An Investigation of an Optical Multiple Pulse Position Modulation Link over a Dispersive Optical Channel

By Konstantinos Nikolaidis

Beng (Hons) in Electronic and Communication Engineering
University of Huddersfield 2003

A dissertation submitted in part fulfillment for the degree of Doctor of Philosophy in Electronic and Optical Communication Engineering

November 2008

ABSTRACT

Digital Pulse Position Modulation (digital PPM) is a modulation format that codes n bits of PCM into a single pulse that occupies one of 2^n time slots. Various studies over the last three decades have shown that such a scheme can offer an improvement in receiver sensitivity of 5-11 dB when compared to Pulse Code Modulation (PCM). Such an increase in sensitivity can be exploited beneficially in both long haul applications and the multi-user environment. However, this improvement results in a considerable increase in the final data rate of the original PCM, and this makes implementation difficult.

Alternative methods have been proposed, such as multiple PPM, dicode PPM, differential PPM and overlapping PPM, that reduce transmission bandwidth while maintaining an increased sensitivity. Dicode and multiple PPM (MPPM) are the most bandwidth efficient of these formats and MPPM, the subject of this thesis, offers the best sensitivity without the large bandwidth increase. In this scheme, multiple pulses per frame are used, with the pulse positions being determined by the original PCM word.

The main concern of this thesis is a full and detailed investigation of an Optical MPPM link operating over a dispersive optical channel. As the analysis of any $\binom{X}{Y}$ multiple PPM system, in which X denotes the number of data slots and Y the number of pulses, is extremely time-consuming, a novel automated solution was designed to predict the equivalent PCM error rates of specific sequences and simplify the task. An original

mathematical formulation is developed using the Maximum Likelihood Sequence Detection (MLSD) scheme. Using the equivalent PCM error rates of specific sequences generated from software, a full simulation of an MPPM optical link can be produced and the results show the effectiveness of the MPPM format over PPM. A measure of coding quality is proposed that accounts for efficiency of coding and bandwidth expansion. Original results are presented for a $\binom{12}{Y}$ MPPM system, considered as very efficient and examined by many authors, showing that the most efficient systems are in the middle of the family. A methodology to predict the Bit Error Rate (BER) of any MPPM system is also proposed. The results obtained confirmed the results obtained from the full mathematical analysis.

The effects of linear increment, linear decrement, Gray code and random mapping of data on the performance of a $\binom{12}{Y}$ multiple PPM system are also examined. Simulations show that the Gray code is the most effective as it minimizes the Hamming distance between adjacent multiple PPM words.

Further experiments showed that system performance can be obtained exclusively with the use of software making the analysis simpler and minimizing the time consumption. A novel algorithm is presented and results obtained using this method, agree with those obtained using a full mathematical model.

Certain mappings can either enhance or degrade the final total error probability of the system and hence affect the sensitivity of the MLSD scheme. In this thesis the author also suggests a methodology of how to predict and generate an optimum or close to optimum mapping. The methodology is based on minimizing the occurrence of dominant error sequences. Detailed results show the effectiveness of this mapping routine.

For the first time also, high order MPPM codes are considered for analysis. All the experiments completed for the $\binom{12}{Y}$ MPPM system are repeated for a range of MPPM systems (with 4, 7, 15, 17, 22, 28 and 33 slots operating over a plastic optical fibre (POF) channel showing again that the most efficient systems are in the middle of the family. Close to optimum mappings are also presented for these MPPM systems. The estimated mappings were found to be far superior to the efficient Gray codes, linear coding and a series of random mappings. To measure these optimum mappings a very efficient mapping is also considered. This mapping minimises the Hamming distance between all MPPM codewords. This mapping allows repetitions of MPPM codewords and cannot be used in a MLSD scheme. Therefore, it is only used for comparisons with the (close to) optimum mappings. It is shown that the optimum mappings are close to ideal.

Other correlation techniques are also considered for optimised detection in the MLSD scheme. Results obtained showed that raised cosine filtering can enhance detection.

Acknowledgements

The author wishes to thank, “a beautiful mind”, the director of studies, Dr M.J.N. Sibley, for his valuable support throughout the PhD completion and especially for his editing reviews.

Many thanks also to my second supervisor, “a true mentor”, Dr P.J. Mather, for his support, goodwill and help when needed, especially with the first publication.

Vicky, for her love, support and understanding all these years.

Dedicated to my parents.

Report Organization

In this thesis, a detailed analysis of the MPPM format is presented when operating in a highly dispersive optical channel. *Chapter 1* presents the background and motivation of this work. *Chapter 2* shows the necessary theory used for this project (error types, error probability and MLSD). *Chapter 3* demonstrates the design of the software solution. A modified traditional methodology is applied. The requirement analysis is done through scenarios and diagrams to determine the needs or conditions to meet for the new software solution. The implementation phase of the software is also presented. The software is implemented through an Object Oriented Programming Language (C++). The main algorithms are described and test results are also demonstrated. The program is split into three main areas (interface, main body, functions). The main body consists of seven algorithms which are explained from data flow diagrams. Testing results obtained from the software alongside test patterns used to verify the software are also presented. *Chapter 4* discusses the mathematical models used to simulate a MLSD scheme used in an optical MPPM link with the use of a matched filter. The mathematical models use error rates from specific sequences calculated from the software solution. An original method of predicting the sensitivity of any MPPM system, and results, are presented for a $\binom{12}{Y}$ MPPM scheme. A methodology to predict the Bit Error Rate of any MPPM system is also proposed and results obtained for a $\binom{12}{Y}$ MPPM system are compared with the results obtained from the mathematical models. *Chapter 5* discusses the effects of data

mapping on the overall sensitivity of a MPPM system and results from mapping experiments are presented for a $\binom{12}{Y}$ MPPM scheme. *Chapter 6* describes a simplified fully automated (software) solution to predict the sensitivity of any MPPM system without the use of complex mathematical models. Also, a new methodology is proposed of how to obtain an optimum or close to optimum mapping for any MPPM system. Results are presented for a $\binom{12}{Y}$ MPPM system. *Chapter 7* presents a full analysis with experimental results of high order MPPM systems and discusses and demonstrates optimum (without redundancy) mappings for higher order MPPM systems. *Chapter 8* demonstrates other correlation techniques used in a MLSD scheme alongside results for a $\binom{12}{Y}$ MPPM scheme. *Chapter 9* presents a discussion of the main points of the research and *Chapter 10* presents the conclusions of this project and outlines possible further work.

Several appendices are also included, after the references section, to present figures, tables, software printouts, mathematical models, publications and the project plan.

Table of Contents

List of Figures

List of Tables

List of Notations and Abbreviations

(In alphabetical order)

ADPCM	Adaptive Differential Pulse Code Modulation
AFE	Analogue Front-End
AHD	Averaged Hamming Distance
α_i	the minimum amount of photons
b	the received pulse energy in photons per bit
BE	bandwidth expansion
$\mathrm{BE}_{\mathrm{norm}}$	the normalized bandwidth expansion
BER	Bit Error Rate
bps	bits per second
B_s	the MPPM slot rate
CCPPM	Colour Coded Pulse Position Modulation
CPU	Central Processing Unit
C\W	Codeword
dBm	decibel referenced to 1mW
DE	Digital Equalizer
dPPM	differential Pulse Position Modulation
DFE	decision-feedback equalizer
ER	Erasure Error
erfc	the complementary error function

FA	False Alarm
f_n	the normalised fibre bandwidth to the PCM data rate
$G(j\omega)$	is the transfer function of the equalising filter
$h_p(t)$	received pulse shape
$H_p(\omega)$	the complex conjugate of the received pulse
I	bit
I_1	shape of the output pulse
IC	Integrated Circuit
IFI	Inter Frame Interference
ISI	Inter Symbol Interference
K	number of pulses
KHz	Kilo Hertz
Kbit/sec	Kilo Bits per second
L	the frame bits
L_k	the known bits
L_u	the unknown bits
M	bits
Mb	the number of PCM codewords
MFC	Microsoft Foundation Class Library
MLSD	Maximum Likelihood Sequence Detection
Mp	the number of MPPM codewords
MPPM	Multiple Pulse Position Modulation
MSB	Most Significant Bit

N	number of slots
n	number of slots
n	number of encoded PCM bits
O/P	Output
OPPM	Overlapping Pulse Position Modulation
PAM	Pulse Amplitude Modulation
P_e	the erasure error probability
$P_{erasure}$	the erasure error probability
P_f	the false alarm error probability
P_{false_alarm}	the false alarm error probability
PCM	Pulse Code Modulation
PCM_BITS	encoded PCM bits
PFM	Pulse Frequency Modulation
PM	Pulse Modulation
$\mathrm{ph}_{\mathrm{PCM\ norm}}$	the normalized sensitivity in photons per PCM bit
POF	Plastic Optical Fibre
PPM	Pulse Position Modulation
p/s	pulses per second
P_s	the wrong slot error probability
PSE	Probability of Symbol Error
PTM	Pulse Time Modulation

P_{TOTAL}	the total error probability
PWM	Pulse Width Modulation
P_{wrong_slot}	the wrong slot error probability
Q_e	fraction defined as $\frac{v_{pk}-v_d}{\sqrt{\langle n_o^2 \rangle}}$
Q_f	fraction defined as $\frac{v_d-v_o(t_d)}{\sqrt{\langle n_o^2 \rangle}}$
Q_s	fraction defined as $\frac{T_s}{2}\frac{slope(t_d)}{\sqrt{\langle n_o^2 \rangle}}$
rms	root mean square
ROM	Read Only Memory
R_T	is the mid-band transimpedance
$slope(t_d)$	the slope of the received pulse at the threshold crossing instant t_d
S_o	preamplifier (double sided) noise at input
S/W	Software
T_b	the PCM bit time
t_d	the threshold crossing time
T_f	frame period
THD	Total Hamming Distance

T_s	the slot width
Ts	the frame period
t_s	the MPPM slot period
v	the threshold parameter
V_d	the receiver output at the threshold crossing t_d
VHDL	Very High (Speed) Description Language
V_o	the voltage level of the slot
V_{pk}	the peak receiver output
V_{po}	the peak receiver output
weight_{BE}	the bandwidth weighting, 1- weight_{sens}
weight_{sens}	the sensitivity weighting, 0 to 1, in steps of 0.1
WCS	Worst Case Scenario
WS	Wrong Slot
X	number of slots
Y	number of pulses
$Z_T(j\omega)$	a dominant transimpedance pole
$\langle n_o(t)^2 \rangle$	the mean square receiver output noise
$\frac{T_S}{\tau_R}$	the number of uncorrelated samples/time slot

α	pulse (variance) shape
α'	pulse (variance) shape
η	efficiency factor
ω_p	is the -3 dB bandwidth of the preamplifier

Chapter 1

Introduction

1.1 Background and Motivation

Electronic information may be transmitted from one point to another using either analogue or digital communication techniques [1]. In analogue communications the three key parameters of a carrier signal are its amplitude, phase and its frequency, all of which can be modified (modulated) in accordance with a low frequency information signal to obtain the modulated signal. In analogue modulation, the modulation is applied continuously in response to the analogue information signal.

In digital modulation, an analogue carrier signal is modulated by a digital bit stream. The modulation of pulses is called Pulse Modulation (PM) [2]. PM is the process of using some characteristics of a pulse (amplitude, width, position) to carry a narrowband analogue signal over an analogue lowpass channel as a two-level quantized signal, by modulating a pulse train. Common modulation formats are:

i) Pulse Amplitude (PAM)

ii) Pulse Density (PDM) or Width (PWM)

iii) Pulse Frequency (PFM)

iv) Pulse Code (PCM)

v) Sigma-delta modulation (ΣΔM)

vi) Continuously variable slope delta modulation (CVSDM), also called Adaptive-Delta Modulation (ADM)

vii) Pulse Time or Position (PTM)

1.1.1 PAM

In PAM (presented in figure 1.1 - A) the amplitude of the pulses carries the information. This basic scheme can be made more sophisticated by using several amplitude levels. For example, signal bits can be grouped into twos, i.e. 00, 01, 10 and 11 and four different amplitude levels can be used for each of these groups. This scheme is known as Quadrature Pulse Amplitude Modulation (QPAM or QAM). Direct-sequence spread spectrum (DSSS) is based on pulse-amplitude modulation

1.1.2 PDM

PDM, is another form of modulation used to convert an analogue signal into a digital signal. In a PDM signal, the relative density of the pulses is proportional to the magnitude of the analogue (input) signal. Pulse-width modulation (PWM) is a special case of PDM. It is widely used in motor control.

1.1.3 PFM

PFM (figure 1.1 - B) is a method of pulse modulation in which the modulating wave is used to frequency modulate a pulse-generating circuit. For example, the pulse rate may be 8000 pulses per second (p/s) when the signal voltage is 0. The pulse rate may step up to 9000 p/s for maximum positive signal voltage, and down to 7000 p/s for maximum negative signal voltage. This method of modulation is not used extensively because the PFM generation circuitry is complicated. It requires a stable oscillator that is frequency modulated to drive a pulse generator. Unlike PDM (and PWM), in which the width of square pulses is varied at constant frequency, PFM is accomplished using fixed-duration pulses and varying the repetition rate.

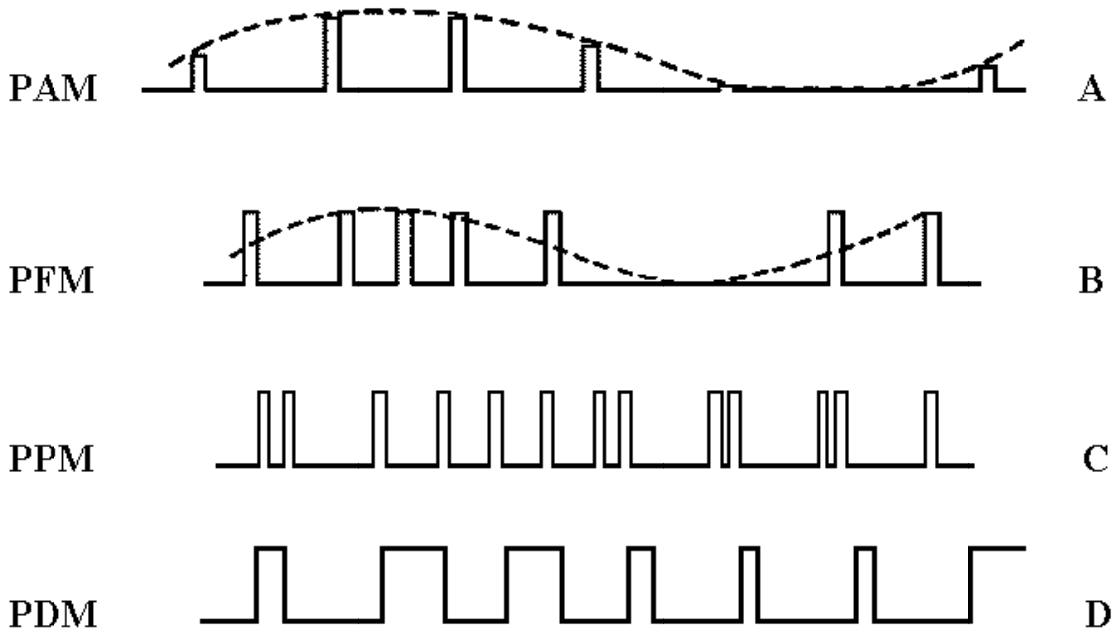


Figure 1.1: PAM (A), PFM (B), PPM (C) and PDM (D) (www.iec.org).

1.1.4 PCM

PCM (figure 1.2) is a general scheme for transmitting analogue data in a binary form independent of the complexity of the analogue waveform. With PCM all forms of analogue data such as video, voice, music and telemetry can be transferred. To obtain PCM from an analogue waveform at the source (transmitter), the amplitude of the analogue signal is sampled at regular time intervals. The sampling rate, is several times the maximum frequency of the analogue waveform (Nyquist rate). The amplitude of the analogue signal at each sample is rounded off to the nearest binary level (quantisation) and represented by a binary word of two, three or more binary bits. At the receiver, a pulse code demodulator converts the binary numbers back into pulses having the same quantum levels as those in the modulator. These pulses are further processed to restore the original analogue waveform.

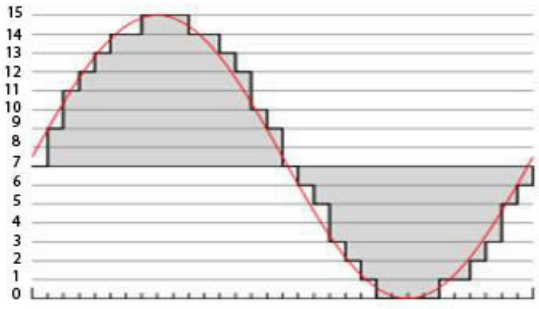


Figure 1.2: PC Modulation (www.iec.org).

Adaptive Differential Pulse Code Modulation (ADPCM) is a technique defined by the International Telecommunication Union (ITU) for converting sound or analogue information to binary information by taking frequent samples of the audio signal and expressing the value of the sampled audio modulation in binary terms. This produces a lower bit rate and is sometimes used to effectively compress a voice signal, allowing both voice and digital data to be sent where only one would normally be sent. ADPCM is a variation of pulse code modulation (PCM) that only sends the difference between two adjacent samples. ADPCM is used to send audio on fiber-optic long-distance lines as well as to store audio along with text, images, and code on a data storage medium. It is also used in digital cordless telephones and radio/wireless local loop.

1.1.5 ΣΔM

The Sigma-Delta (ΣΔ) modulation is a method for encoding high resolution signals into lower resolution signals using pulse-density modulation. This technique has found increasing use in a range of modern electronic components, such as analogue-to-digital and digital-to-analogue converters, frequency synthesisers, switched mode power supplies and motor controls. One of the earliest and most widespread uses of delta-sigma modulation is in data conversion. An ADC or DAC circuit which implements this technique can easily achieve very high resolutions while using low-cost CMOS processes, such as the processes used to produce digital integrated circuits; for this reason, even though it was first presented in the early 1960s, it is only in recent years that

it has come into widespread use with improvements in silicon technology. Almost all analogue integrated circuit vendors offer delta sigma converters.

1.1.6 CVSDM

Continuously variable slope delta modulation (CVSD or CVSDM) is a voice coding method. It is a delta modulation with variable step size (i.e. special case of adaptive delta modulation), first proposed by Greefkes and Riemens in 1970 [3]. CVSD encodes at 1 bit per sample, so that audio sampled at 16 KHz is encoded at 16 Kbit/s. The encoder maintains a reference sample and a step size. Each input sample is compared to the reference sample. If the input sample is larger, the encoder emits a 1 bit and adds the step size to the reference sample. If the input sample is smaller, the encoder emits a 0 bit and subtracts the step size from the reference sample. The encoder also keeps the previous N bits of output ($N = 3$ or $N = 4$ are very common) to determine adjustments to the step size; if the previous N bits are all 1s or 0s, the step size is doubled. Otherwise, the step size is halved. The step size is adjusted for every input sample processed. The decoder reverses this process, starting with the reference sample, and adding or subtracting the step size according to the bit stream. The sequence of adjusted reference samples are the reconstructed waveform, and the step size is doubled or halved according to the same all-1s-or-0s logic as in the encoder. Adaptation of step size allows one to avoid slope overload (step of quantization increases when the signal rapidly changes) and decreases granular noise when the signal is constant (decrease of step of quantisation).

CVSD is sometimes called a compromise between simplicity, low bitrate, and quality. Bitrates are 9.6 to 128 Kbit/s.

1.1.7 PTM

The time characteristics of pulses may also be modulated. Two time characteristics may be affected, the time duration of the pulses and the occurrence (position) of the pulses. The main PTM techniques are:

i) Pulse Position Modulation (PPM)

In 1949, Golay [4] published a paper that considered the use of digital PPM (referred as QPPM) as the practical way of approaching the Shannon limit. Many other authors after him [5]-[66] continued his research on QPPM (referred to as PPM). Optical Digital PPM (referred in some papers as ODPPM) is an early form that codes n bits of PCM data into a single, narrow, high-energy pulse which occupies one of $n=2^M$ (where M is the number of encoded PCM bits) allowed pulse positions or slots in a frame of duration T_n (as presented in figure 1.3). A guard band is left at the end of each frame and this defines a modulation depth, m. The first level of PPM synchronisation consists of identifying the slot boundaries and frequency. The second level consists of identifying the frame frequency. The last and most important part in synchronizing PPM signals is the establishment of the proper frame phase from the incoming data stream. Thus

synchronizing a PPM system is quite complicated. Generally it can be stated that efficiency improves with increasing the number of slots because more bits are being conveyed.

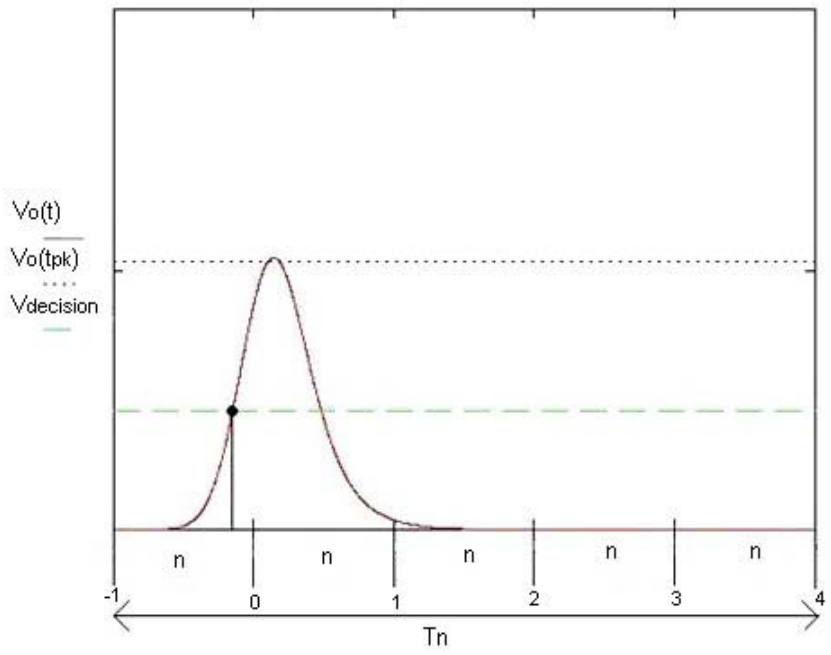


Figure 1.3: An Optical PPM frame encoding 2 PCM bits. Note the guard interval at the end of the frame.

McAulay and Sakrison, [5] developed a hybrid PPM/PM modulation system. They showed that for a fixed signal having a specified bandwidth the PPM/PM modulation can perform significantly better than straight PPM at higher values of the input SNR. Karp and Gagliardi, [6] considered some design aspects of an optical M-ary PPM communication system using photon counters at the receiver. The system considered transmits monochromatic optical energy in one of *M* time intervals, and the receiver determines the photon count in each interval and performs a maximum-

likelihood test to determine which signal has been received. Blachman, [8] studied the spectrum of a time-division-multiplexed (TDM) pulse-position-modulated (PPM) signal. The spectrum consists of three parts - the continuous part, which is simply the sum of the continuous parts of the spectra of the separate channels; the lines and harmonics of the maximum overall pulse-repetition frequency (PRF). Simple expressions were found for all three parts under the assumption that *N*-*M* channels are empty (input lines have no data to send). *N* are unmodulated and the remaining *M* contain independent, identically uniformly distributed pulse positions. A complicated expression was also found for the variances of the lines in the last part of the spectrum under a random choice of the modulated and unmodulated channels, and a simple approximation was presented for it. The author also obtained the signal's continuous spectrum, which was simply the sum of the continuous parts of the spectra of the individual modulated channels and was independent of which channels they were.

The same author also studied [9] the signal-to-noise-ratio (SNR) performance of a pulse-position-modulation (PPM) receiver when the input SNR exceeded the threshold, but below it the output SNR deteriorated on account of false pulses due to noise. A formula was obtained for the output SNR as a function of the input SNR that was valid below as well as near and above the threshold. From it the threshold was easily determined, and it was found to be higher than previously indicated. The paper concluded with a new, simpler derivation of Rice's result as well as a resolution of the demodulator output into its various components. The relation between the time-bandwidth (TB) product and the threshold peak SNR was also obtained. The latter turned out to be about

17 dB for values of TB in the range of 10-15. Thus, the SNR threshold was found in PPM reception to be considerably higher.

Muoi and Hullett, [10] developed an optimum receiver structure for an optical PPM system and from this a simple sub-optimum receiver was proposed which offered significant SNR improvement and yet required few circuit changes. For a FET preamplifier, the SNR improvement was 10.8 dB with a p-i-n photodiode and 4.7 dB with an avalanche photodiode. For a BJT preamplifier, the SNR improvement was 3.1 dB and 2.3 dB with the p-i-n and avalanche diodes, respectively. Garrett, [14] analysed the receiver sensitivity of an optical PPM system over a slightly dispersive channel, where both "wrong slot" and "false alarm" errors are important. It was shown that receiver sensitivity of better than 100 photons per binary bit-time was theoretically possible using direct detection and un-coded PPM. Ideal heterodyne detection could reduce this to below 5 photons per binary bit-time. Timing extraction and a digital modulation method were discussed.

Gagliardi and Prati, [15] investigated laser pulse stretching in optical PPM formats in terms of performance degradation and decoder design alternatives. Several methods were considered for combating the pulse stretching, including pulse equalization, extended pulse integration, and pulse shape matching. Performance of these methods was compared for the case of exponential stretching and Gaussian statistics. The methods included pulse equalization, and spread pulse energy integration and matching. Optimal performance, therefore minimum PSE (probability of symbol error), occurred

only if the pulse shape was exactly matched, but the performance can be approached by a piece-wise (logarithmic and exponential function) integrate-and-correlate decoding.

Prati, [16] examined the maximum probability decoding for a stretched-pulse, PPM, direct-detection optical communication system when the spreading factor of the received laser pulse was unknown. Based on a discrete count model, joint pulse spreading estimators and decoders were derived for both Poisson and avalanche photodetection (APD) cases. Performance was evaluated in terms of probability of error for pulse decoding.

Ling and Gagliardi, [18] examined the slot clocking design associated with a direct detection, photodetecting optical PPM system. Several types of practical slot synchronizers were also considered. A basic design involving analogue correlators and slot gating was presented, along with an indication of its performance. Several alternative designs were also presented, including digital synchronizers in which time samples were used for loop control. The advantage in digital systems is that more extensive processing could be handled in software, allowing the loop to perform closer to the ideal. Design procedures for digital clocking were presented, and optimal laser pulse shaping and filtering were discussed. Performance in terms of loop models and tracking error variance was included. It has been shown that accurate, reliable, and implementable slot clocking tracking loops can be designed for PPM decoder processors. These loops can be designed without the necessity of decoding decision feedback, thereby eliminating delay lines and complicated storage hardware. The primary disadvantage was to make binary slot

decisions fairly accurately, which produced slightly poorer performance then predicted by idealized tracking theory. Some alternative designs were possible, eliminating the binary decision making, but they became noisier and generally had limited linear tracking range. An alternative digital design was also considered, in which the early-late gate integrators were replaced by A/D conversion, and sample differencing was used to generate error voltages. The digital system can be optimised by properly designing the preloop filter and shaping the transmitted laser pulse so as to minimize tracking variance. The use of software processing made it easier for the digital clock loop to be decision-directed by the PPM decoding, whereas an equivalent analogue loop would require accurate delay lines to achieve the same performance.

Charbit and Bendjaballah, [20] calculated the capacity for a Poisson channel with a source noise as modeled by Pierce by bounding the error probability. Based on Chernoff's bound properties, a more general method was developed, yielding a formula for the channel capacity.

Cryan et. al, [28] investigated the PPM potential for coherent optical fibre communications channel. They presented a thorough performance and optimisation analysis. Comparisons, at a wavelength of 1.5 μm, were made with shot-noise limited coherent PCM (homodyne and heterodyne ASK, FSK, and PSK) over a range of fibre bandwidths and varying PPM word size. They concluded that, for moderate to high fibre bandwidths, homodyne digital PPM could achieve an improvement in sensitivity of typically 5 dB over homodyne PSK PCM. Cryan [29] also considered the simplification

of the receiver structure, and an algorithm was developed for calculating the receiver sensitivity when sub-optimum pre-detection filters were employed. Original results were presented for the sub-optimum detection of heterodyne n-ary PPM, and they illustrated that the receiver complexity can be significantly simplified at a cost of only 0.9 dB degradation in sensitivity. Finally, the algorithm was used to model an experimental 16-ary heterodyne PPM system, predicting a sensitivity of -65.5 dBm when operating with a slot duration of 20 ns. This is within 2.8 dB of that measured practically, and 5.6 dB short of the shot noise limit due to a limited local oscillator power of 14 μW. This represents an improvement of 18.2 dB over an equivalent direct detection 16-ary PPM system.

Advani and Georghiades, [30] demonstrated jointly optimal receivers that make decisions in the absence of symbol synchronisation and analysed a pulse-position modulation, optical direct-detection channel. It was seen that jointly optimal receivers were superior to conventional receivers that had separately designed synchronisation and decision subsystems. However, their performance advantage was significant only at very low signal levels. Simulation results indicated that the much less complicated receivers that observed histogram data performed as well as receivers that observed the complete sample-path at a rather small number of bits per slot.

Cryan et. al, [33] showed from experiments that the optimum sensitivity in a PPM system occurs when the contributions from the error sources are equal. Above this optimum, sensitivity is maximised by balancing False Alarm errors with Wrong Slot

errors. Below the optimum, it is maximised by balancing Erasure with False Alarm errors.

The use of coding techniques (Reed-Solomon, [41] Viterbi, [42] Trellis, [43] and convolutional codes [44]) for an optical fibre PPM channel was considered by Garrett [45] in 1981. RS and convolution coding were discussed by McEliece [46]-[47] for free space optical communications. Divsalar and Gagliardi [48] considered an optical-RF relay deep space communication link that transmits optical PPM data from spacecraft to an optical relay that then retransmits the data via microwave to ground. It was generally advantageous to use Reed-Solomon encoding over the PPM optical link for improved error correction. Several demodulating schemes were also considered. Yichao et al [51] discussed repeatered optical fibre communication with line-coded digital PPM in 1985. Since then several papers [52]-[64] have been published in the area of line, RS and convolution coding for direct and coherent detection of digital PPM over optical fibre channels. The conclusions from these papers are that RS coding offers increased receiver sensitivity, and that there exists an optimum code rate for a particular system. Combining RS and convolution coding to form an inner and outer code system, also yields advantages in system performance.

Cryan et. al [58] also presented a performance and optimisation analysis for both uncoded homodyne digital PPM and digital PPM employing Reed-Solomon error-correction codes. The system performance for a range of fibre bandwidths and PPM symbol sizes was analysed, and it was shown how the predetection filter may be

configured in order to minise the three error sources and achieve maximum transmission efficiency (nats/photon). Results were presented at a bit rate of 565 Mbit/s and a wavelength of 1.5m, comparing both uncoded and coded homodyne digital PPM with shot-noise-limited coherent PCM. It was shown that there are optimum PPM symbol sizes, fibre bandwidths and Reed-Solomon code rates at which to operate. The conclusion was that uncoded digital PPM offers an improvement of 5 dB over homodyne PSK PCM, and that the Reed-Solomon error-correction coded system offered a 4 dB improvement over uncoded PPM, when operating at the optimum $3/4$ code rate.

Cannone et. al, [59] studied the performance of convolutionally coded pulse position modulation (PPM) systems in the presence of slot synchronisation errors, for the shot-noise-limited photon-counting receiver and the avalanche photodetection (APD) receiver. Both hard and soft (δ-max) demodulation results were given, and two soft-decision metrics were investigated. The effect of slot synchronisation errors on the performance of an interleaved PPM system, where the pulses are arranged in a non-contiguous way in order to increase performance, using convolution was evaluated for both photon counting and APD receivers with hard and soft demodulation and Viterbi decoding. The results indicated that the performance of coded PPM systems with hard demodulation is in general significantly affected by imperfect slot synchronisation unless the value of the normalized loop bandwidth is less than 10^{-3}.

In 1992 Massarella and Sibley completed experimental work [61] in the area of IFI and Reed-Solomon code error correction for an optical PPM channel. This work has

shown that the effects of IFI in the first time slot can be completely eradicated. The use of RS coding has also been experimentally investigated with conclusions drawn on the random and burst error correction ability of such codes with PPM channels.

Lee and Kahn, [64] analyzed the performance of trellis-coded pulse-position modulation with block decision-feedback equalization (BDFE) and parallel decision-feedback decoding (PDFD) on indoor, wireless infrared channels. They showed that the reduced complexities of BDFE and PDFD as compared to maximum-likelihood sequence detection allow for better codes whose increased coding gain more than compensates for the penalty due to suboptimal detection. They also quantified these net gains in performance over a range of dispersive channels, indicating where BDFE and PDFD provided the best performance. Thus, for TC 16-PPM, BDFE provided the best performance. For TC 8-PPM, BDFE provided the best performance for normalized delay spreads $D_T<0.2$, but because of significant penalties due to decision errors in BDFE, PDFD provided the best performance for $D_T>0.2$.

ii) Differential PPM (dPPM)

The synchronisation difficulties of PPM can be solved using differential PPM (dPPM) [67]-[70] whereby each pulse position is encoded relative to the previous pulse. Thus the receiver must only measure the difference in the arrival time of successive pulses. It is possible to limit the propagation of errors to adjacent symbols, so that an error in measuring the differential delay of one pulse will affect only two symbols,

instead of causing all successive measurements to be erroneous. dPPM was proposed by Zwillinger [68] as a way of increasing throughput for a band-limited and average-power-limited optical channel. Shirokov and Bukhinnik [68] also considered dPPM when transmitted over optical fibre channels. In particular they evaluated the reliability of such systems. Peile [69] completed a theoretical analysis for dPPM with error correcting codes combined with interleaving techniques.

Shiu and Kahn [70], presented expressions for the error probability and power spectral density of DPPM. They showed that for a given bandwidth, dPPM requires significantly less average power than pulse-position modulation (PPM). They also examined the performance of dPPM in the presence of multipath intersymbol interference (ISI). They found that the ISI penalties incurred by PPM and dPPM exhibited very similar dependencies upon the channel rms (root mean square) delay spread. They also discussed the use of chip-rate and multichip-rate equalization to combat ISI. Finally, they described potential problems caused by the nonuniform bit-rate characteristic of dPPM, and proposed several solutions. They considered several receiver structures, including a simple, unequalized hard-decision receiver, an MLSD, a chip-rate decision-feedback equalizer (DFE), and a multichip-rate DFE (the MLSD comprises an Analogue Front-End (AFE) IC and a Digital Equalizer (DE) IC which are packaged in a multichip module). They concluded that dPPM always achieved high power efficiency and lower hardware complexity than PPM. These made dPPM a favorable candidate to replace PPM in many applications. Using a simple model for the indoor wireless infrared channel, they found that the ISI penalties of dPPM were essentially determined by the ratio of the rms delay

spread to chip duration. DFE was discovered as an effective technique to combat ISI. The same authors derived the PSD of dPPM signals assuming that the transmit pulse shape was rectangular. The PSD did not approach zero at dc. Thus, if highpass filtering is employed to reduce the effect of fluorescent light noise, dPPM signals are subject to greater distortion than PPM signals.

iii) Overlapping PPM (OPPM)

In 1984 Bar-David and Kaplan [71] and in 1999 Shalaby, [72] published some work on Overlapping PPM (OPPM). This modulation scheme was found to allow multiple positions per pulse width as well as fractional modulation indices (number of pulse widths per frame). The associated theoretical analysis showed that, for low data rates, OPPM offers a 20 percent advantage over conventional PPM in nats/photon.

iv) Colour Coded PPM (CCPPM)

Colour coded PPM (CCPPM) was proposed by Davidson and Bayoumi in several papers [73]-[76]. This technique uses a different optical non-overlapping centre frequency for each individual PPM data slot. It was found that this system offered higher energy efficiency in nats per average number of received photons per pulse than an ordinary PPM system. Gagliardi and Kim [21] also considered CCPPM combining laser diodes, operated in different wavelengths, in conjunction with a pulse position modulated (PPM) format (several laser diode sources that were combined into a single beam for

digital PPM transmission). They considered three different system architectures. In the first design a single PPM data pulse was represented by several optical frequencies which were combined before the digital PPM modulator. The second design also used a single beam, but each optical wavelength was modulated by digital PPM before the optical combiner. They considered the data rate performance of the systems based on the PPM coding level, number of wavelengths used, optical SNR and losses associated with the beam optics.

v) Multiple PPM (MPPM)

The use of digital pulse position modulation using multiple pulses (figure 1.4) per time frame was considered by Lee and Schroeder [77] in 1977. This paper gave analytic and computer simulation results for a pulse-interval modulation (PIM) system that represented a discrete PPM system. They also determined the laser power required to achieve a given bit error rate. This modulation format was then theoretically investigated by Gol'dsteyn and Frezinski [78] in 1978 when it was transmitted over a channel containing regenerators. Marougi and Sayhood [79] published a paper in 1983 that had a complete noise performance analysis and Fyath et al [80] investigated the spectral properties for timing extraction purposes. Since then several papers have been published in the area of Pulse Interval and Double Header PI Modulation [81]-[83].

The noise immunity of Multiple PPM (MPPM) was investigated by Yemin and Petrich [84] when operating over a dispersive channel i.e. pulse broadening. They derived

error probabilities for incorrect reception of the signal assuming a Gaussian received pulse shape and threshold crossing detection. Sugiyama and Nosu [85] proposed a detailed noise performance of a $\binom{12}{2}$ multiple PPM, optical fibre system in the presence of erasure errors. A Maximum Likelihood Sequence Detector (referred as MLSD) was used as the decoder-detector and this same scheme is also used in this thesis. They concluded that multiple PPM is more efficient than digital PPM in terms of power and bandwidth utilization (MPPM reduces the transmission bandwidth by half), resulting in a best predicted sensitivity of 0.58 bits/photon compared to the 0.5 bits/photon for digital PPM, both operating at an error rate of 1 in 10^{-9}. They also proposed a practical method of finding the "optimum" (equivalent PCM) mapping. By constantly changing the mapping they recorded the final error probability. Depending on the rate of change of the error probability, they changed the mapping until the error probability was not reduced any more. This mapping was theoretically named as the optimum.

Majumder et al [87] considered the performance degradation of coded MPPM systems due to slot synchronisation error for an APD type receiver, and obtained expressions for receiver degradation with the timing jitter variance.

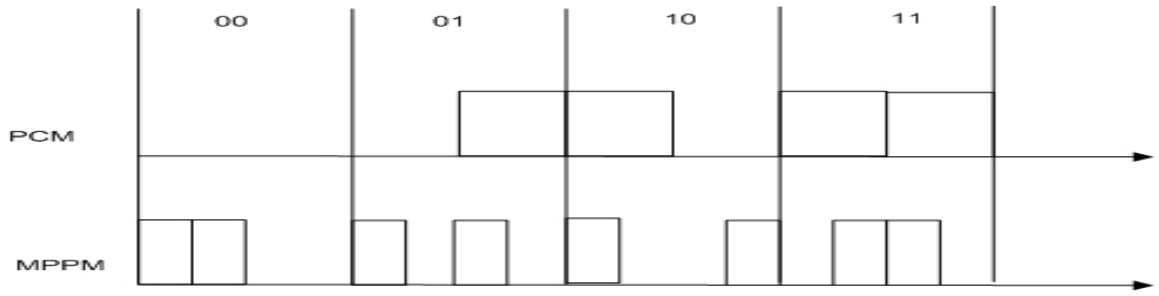


Figure 1.4: Conversion of PCM data to multiple PPM.

Park and Barry [92] examined the performance of multiple PPM and its variants PPM and OPPM on ISI channels with additive white Gaussian noise. The error probability and channel capacity results indicated that, although PPM modulation schemes were extremely power efficient across ISI-free channels, their power efficiency dropped dramatically when ISI was present. The same authors also investigated the effect of dispersion on multiple PPM [100] for indoor wireless infrared communication. They concluded that a partial-response pre-coding at the transmitter reduces the ISI span to two baud periods, which reduced the complexity of the receiver significantly, providing a good balance between performance and complexity.

Velidi and Georghiades [93], investigated the synchronisation properties of slot-synchronized multiple pulse position modulation (MPPM) sequences. They derived a bound on the probability of MPPM symbol synchronisation and identified synchronisable MPPM symbols, which, when periodically inserted in the data stream, can remove an observed performance floor.

Sibley [94] presented an original performance analysis of a $\binom{12}{2}$ multiple PPM system with a slope detection system coupled with a classical matched filter, MLSD scheme, to combat inter-symbol interference. The author concluded that this multiple PPM scheme (used in Plastic Optical Fibre - POF) had a 7.36 dB advantage over PCM when operating under wide bandwidth conditions. When all consecutive pulses were replaced by three-pulse sequences to reduce the effects of ISI and IFI at low bandwidths, it was shown that such a hybrid 2/3 pulse system gave a sensitivity of -22.74 dBm at a channel bandwidth of 0.7 times the PCM bit rate. This represented a 3.61 dB improvement over the original two-pulse MPPM system.

Cryan and Sibley [95] simplified the receiver design by employing raised cosine filtering to eliminate ISI. They showed that very good performance can be achieved by using a simple first-order preamplifier in cascade with a 3rd order Butterworth pre-detection filter, both with their bandwidths set at 0.6 times the MPPM slot rate. When operating with a POF bandwidth of 0.7 times the data rate, it was shown that both the ideal raised cosine scheme and the simple Butterworth pre-detection filter system offered improvements in sensitivity of 7.6 dB and 8 dB respectively over the more complex MPPM MLD system.

The use of error reduction codes, such as Reed-Soloman (RS) to further increase receiver sensitivity, was proposed by Atkin and Fares [96] in 1989. They analyzed the performance of a RS coded multiple PPM system using an avalanche photodiode (APD)

and predicted 0.1 nats/photon, compared to the 0.03 nats/photon for an equivalent (RS) PPM system, both operating at an error rate of 1 in 10^{-9}. Since then several papers [97]-[103] have been published using coding techniques in MPPM.

Herro et al [97] and Takahashi et al [98], in 1989, analysed the use of error correcting codes with MPPM modulation. The main conclusion drawn from these papers was that a Reed-Soloman (RS) coded MPPM system achieves an energy efficiency of more than twice that of an RS coded digital PPM system.

Park and Barry [101] presented new trellis codes based on multiple-pulse-position modulation (MPPM) for wireless infrared communication. They assumed that the receiver uses maximum-likelihood sequence detection to mitigate the effects of channel dispersion, which were modeled using a first-order lowpass filter. Compared to trellis codes based on PPM, they concluded that the new codes were less sensitive to multipath dispersion and offered better power efficiency when the desired bit rate was large, compared with the channel bandwidth. Thus, for the trellis-coded $\binom{17}{2}$ MPPM (where the bit rate equals the bandwidth), required 1.4 dB less optical power than trellis-coded 16-PPM having the same constraint length. They also concluded and showed that a $\binom{12}{2}$ combination is particularly efficient.

Garrido-Balsells, Garcıa-Zambrana and Puerta-Notario [102] presented a novel rate-adaptive transmission scheme using block coding of variable Hamming weight. This

coding scheme is based on the MPPM modulation technique, where codewords with different Hamming weight are allowed (vw-MPPM), including the all-zero one and providing better performance in terms of bit error rate (BER) if compared with other transmission methods, specially with Infrared Data Association (IrDA) standards. They also studied [103] the spectral characterisation of the vw-MPPM. The spectral evaluation was realised using codeword correlation matrices, obtaining an expression including the continuous and discrete parts of the spectrum. Additionally the existence of an oscillating behavior was showed, having a lower relevance as the output codeword length n was increased, and providing a smoother power spectral density.

vi) Dicode PPM (DiPPM)

In dicode signaling, proposed by Sibley [104]-[105], data transitions from logic zero to logic one are coded as +V and transitions from logic one to logic zero are coded as –V. As shown in figure 1.5, a zero signal is transmitted if there is no change in the PCM signal. The positive pulse can be regarded as setting the data to logic one (pulse SET), whereas the negative pulse resets the data to logic zero (pulse RESET). In dicode PPM, these SET and RESET signals are converted into two pulse positions in a data frame. Thus a PCM transition from zero to one produces a pulse in slot R. If the PCM data is constant, no signal is transmitted (although two guards slots have been used in this system, to reduce the effects of Inter-Symbol Interference – ISI - this depends on the channel characteristics. If there is minimal ISI, zero guard slots could be used). In this particular system, four slots are used to transmit one bit of PCM, and so the line rate is

four times that of the original PCM: a considerable reduction in speed compared to digital PPM. As the bandwidth requirement is much smaller then digital PPM, dicode PPM could be used in dense wavelength division multiplexing (DWDM) systems.

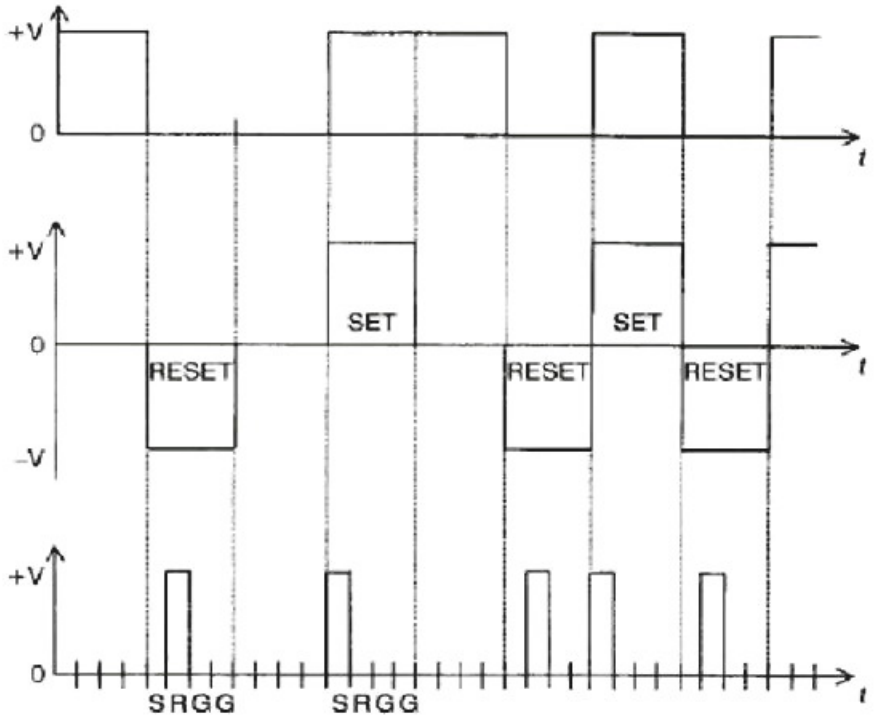


Figure 1.5: Conversion of PCM data (top trace) into dicode (middle trace) and dicode PPM (bottom trace) [104].

1.1.8 Maximum Likelihood Detection, Error Probability and Avalanche Photodiodes

Many authors, [6], [7], [64], [85] and [101], considered a Maximum Likelihood Detector as a decoder. Mohanty, [106] in 1974 derived the maximum likelihood detector for pulse-position-modulated (PPM) signals in Laguerre communications. A decision-directed maximum likelihood estimator for the delay of PPM signals was discussed. An adaptive estimator based on decision criteria was also derived on the assumption of very

high SNR. A minimum mean-square estimator was also derived. A Maximum Likelihood Sequence Detector (MLSD) was also used in this work.

The performance analysis of a digital optical encoding scheme is done by calculating or estimating the error probabilities in the digital optical receiver. Mansuripur and Goodman, [107] applied the Gram-Charlier series method to the calculation of error probabilities in digital optical receivers. This method allowed the calculation of "exact" error probabilities including the effects of avalanche noise, thermal noise, and arbitrary post-detection processing filter.

Gagliardi and Prati, also investigated [108] the output voltage of an optical receiver which is statistically a mixture random variable, composed of the sum of a discrete count variable and a continuous Gaussian thermal noise variable. Based on some computers analyses, it was shown that threshold crossing probabilities using the mixture density can be reliably approximated by integrations of an equivalent continuous Gaussian density. The conclusions applied in optical communication receivers with APD where such mixture densities arose. The validity of being able to use tabulated Gaussian density integrals (erf functions) greatly aided communication analysis in on-off keyed and pulse position modulated encoding where error probabilities appeared as such integrals.

Hayat et al [109] computed bit-error rates for an on-off keying optical communication system using avalanche photodiodes (APD). Using an exact analysis they

showed that the presence of dead space (minimum distance that a newly generated carrier must travel in order to acquire sufficient energy to become capable of causing an impact ionization in the multiplication region of the APD) enhances the performance at relatively low data rates. Using a Gaussian approximation technique with the exact mean and variance, they demonstrated that dead space degrades the performance at high rates, since it is responsible for longer tails in the impulse response function of the APD, which, in turn, increased the effect of intersymbol interference.

1.2 Introduction to the Problem

This thesis is looking at MPPM for two reasons. The first one is because the MPPM format remains the most efficient of all the modulation formats proposed [94]. MPPM combines the advantages of the PPM format as far as the receiver sensitivity is concerned, but without the big bandwidth expansion as shown in figure 1.6.

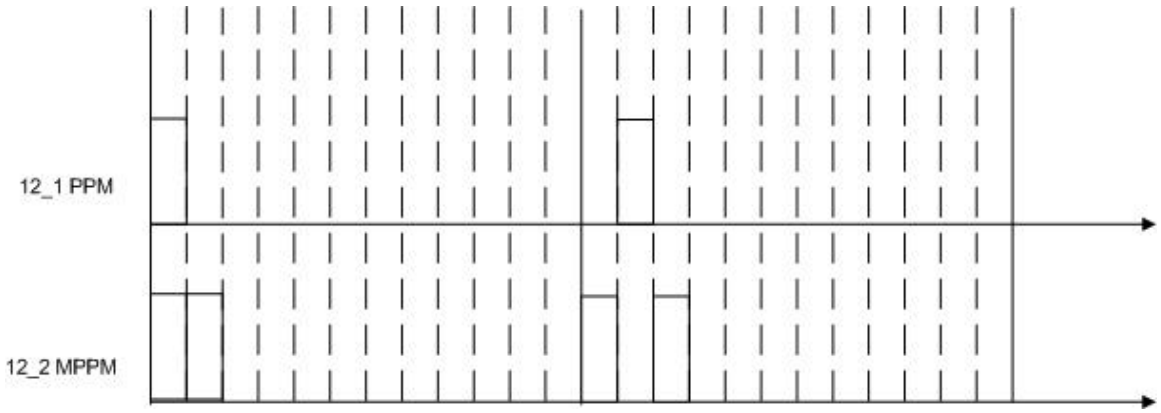


Figure 1.6: Two (2) frames of a 12-1 PPM ([1] and [2] frame) and 12-2 MPPM ([1,2] and [1,3] frame) systems. The MPPM system can encode 6 PCM bits (66 possible pulse combinations) instead of 3 (12 pulse combinations) being encoded by the PPM format (50% better encoding capability). Thus, to encode 6 PCM bits using a PPM format a 64-1 PPM system should be used (5.33 times more bandwidth expansion/frame).

The second reason is that although various authors [77]-[103] worked with the MPPM format, many issues have not yet been fully explored. One of these issues is the complexity for a performance analysis of MPPM systems (they suffer from the same error sources as PPM). This, alongside the geometrical increase in the MPPM data range (referred to here as codewords), makes the investigation of this format difficult, especially for large MPPM systems. For example, a $\binom{16}{1}$ MPPM system can encode 4 PCM bits, whilst the $\binom{16}{6}$ MPPM system can encode up to 12 PCM bits (4096 different MPPM codewords need to be considered, and a much larger number of error sequences, for a full analysis). This problem can be overcome with the design and development of an

automated (software) solution where sensitivity can be predicted (simulated) in a time efficient way, even for high order MPPM systems.

Another issue is the performance analysis considered so far. Some authors, [84]-[93] and [96]-[103], considered only the receiver sensitivity (without considering the bandwidth expansion) as the efficiency factor, whereas others considered only some types of errors [85]. This work presents, for the first time, a full and detailed analysis. All types of errors are considered, ISI and IFI (inter-frame interference) are also considered and the manner in which the erasure, wrong-slot and false-alarm errors affect system performance. Also, in this performance analysis, the methodology proposed considers both sensitivity and bandwidth expansion. This helps to choose the most efficient system (from the same or different system families) according to link specifications.

Several authors also considered the use of redundancy to enhance receiver sensitivity [96]-[103]. Nevertheless, redundancy comes with a cost in bandwidth expansion (especially when redundancy is used for Error Correction). Redundancy can be ignored if there is a mapping (referred here as optimum) where it is more immune (or less sensitive) to errors and thus decreases the final error probability. A main concern of this research was the development of a methodology to obtain an optimum mapping that decreases the error probability and hence increases receiver sensitivity. The results of this research aid in the prediction of an optimum mapping (automatically and time efficiently) for any MMPM system and consequently the use of redundancy may not be necessary limiting the bandwidth expansion. The finding of this optimum mapping is very

complicated, considering the vast amount of possible mappings a MPPM system can have. For example, a $\binom{12}{2}$ MPPM system can encode 6 PCM bits (2^6=64 MPPM codewords) meaning a total of 64 factorial, 64!, different mappings.

1.3 Research Objectives

Below is a list of objectives set prior and during the research period.

1) Investigate an Optical MPPM link over a dispersive optical channel. The effects of receiver noise and channel dispersion should be accounted for and the manner in which the erasure, wrong-slot and false-alarm errors affect system performance should be studied. The receiver/decoder should use slope detection (i.e. a classical matched filter) and a Maximum Likelihood Sequence Detector (MLSD).
2) Propose a performance analysis generated from this investigation.
3) Develop a novel automated solution to predict (the equivalent PCM error rates of specific MPPM sequences and therefore) the performance of any MPPM system. This objective was set because the investigation of MPPM systems is extremely time-consuming.
4) Simulate an MPPM decoder through the use of mathematical models.
5) Investigate the effects of various mappings, without the use of redundancy, like linear, random and Gray Codes.

6) Propose a methodology to be able to measure the (Bit) Error Probability of any MPPM system.
7) Investigate other correlation techniques (i.e. raised cosine filtering) in the decoder.
8) Propose an optimum mapping that will minimize the total error probability.

The work has led to the following publications (presented in appendix F):

1) "Investigation of an Optical Multiple PPM Link over a Highly Dispersive Optical Channel", K.Nikolaidis, M.J.N.Sibley, IET Optoelectronics, June 2007, Volume 1, Issue 3, p. 113-119.
2) "Theoretical Investigation into the Effects of Data Mapping in an Optical multiple PPM Link", K.Nikolaidis, M.J.N.Sibley, Electronics Letters, September 2007, Volume 43, Issue 19, p. 1042-1044.
3) "Optimum Mapping in an Optical Multiple PPM link using a Maximum Likelihood Sequence Detection Scheme", K.Nikolaidis, M.J.N.Sibley, IET Optoelectronics, February 2009, Volume 3, Issue 1, p. 47-53.
4) "Investigation of Higher Order Optical Multiple PPM Links over a Highly Dispersive Optical Channels", K.Nikolaidis, M.J.N.Sibley, accepted IET Optoelectronics paper.

Chapter 2

Theory

As previously described Pulse Position Modulation is a form of signal modulation in which M message bits are encoded by transmitting a single pulse in one of 2^M possible time-shifts. This is repeated every *T* seconds, such that the transmitted bit rate is *M/T* bits per second.

As shown in figure 2.1 for MPPM, if *k* pulses are transmitted in a timeframe of *n* slots, the number of combinations are $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ compared to PPM (where there are no combinations of position). Ideally, $\binom{n}{k}$ should be a power of two to ease implementation but for $k > 1$ this is rarely the case. For example, a $\binom{12}{2}$ multiple PPM system uses a 12-slot frame with 2 data pulses to code 6 bits of PCM data. This gives 64 valid multiple PPM frames instead of the available 66. If linear mapping is used, the PCM word 000000 is translated to a codeword with pulses in slots 1 and 2 (referred as [1,2] MPPM codeword) and the PCM word 111111 is translated to the [10,11] codeword.

If higher order codes are used, such as $\binom{12}{6}$, further reductions in bandwidth can be achieved because it becomes possible to encode up to 9 bits ($\binom{12}{6} = 924 \approx 2^9 = 512$).

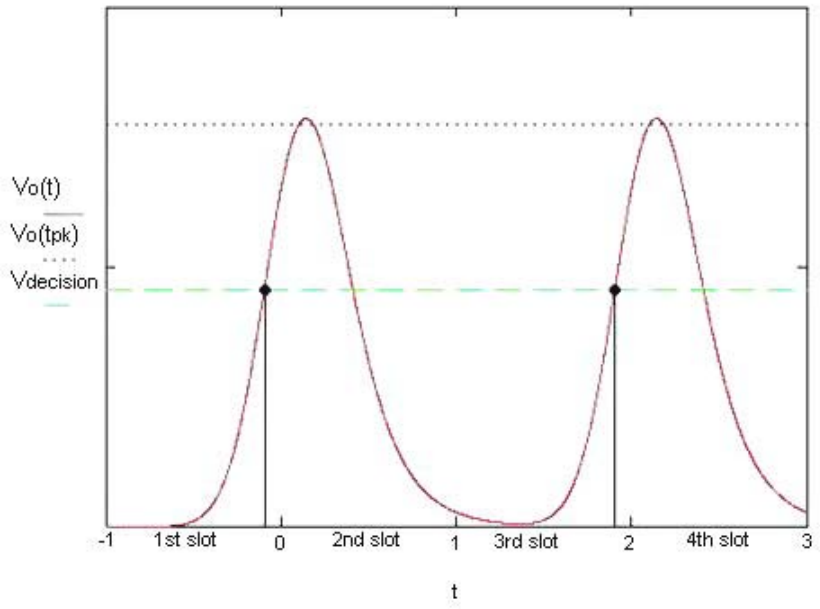


Figure 2.1: An Optical MPPM frame with pulses in slots 1 and 3 [1,3].

The general features of an optical fibre system employing MPPM are shown in figure 2.2. A PCM source of information provides the input to the system. The PCM input of M bits in a frame of duration T_f. A PCM to MPPM coder converts these M information bits into k pulses in a timeframe of n slots. A guard interval can be left at the end of each frame to allow for fibre dispersion and hence avoid inter frame interference.

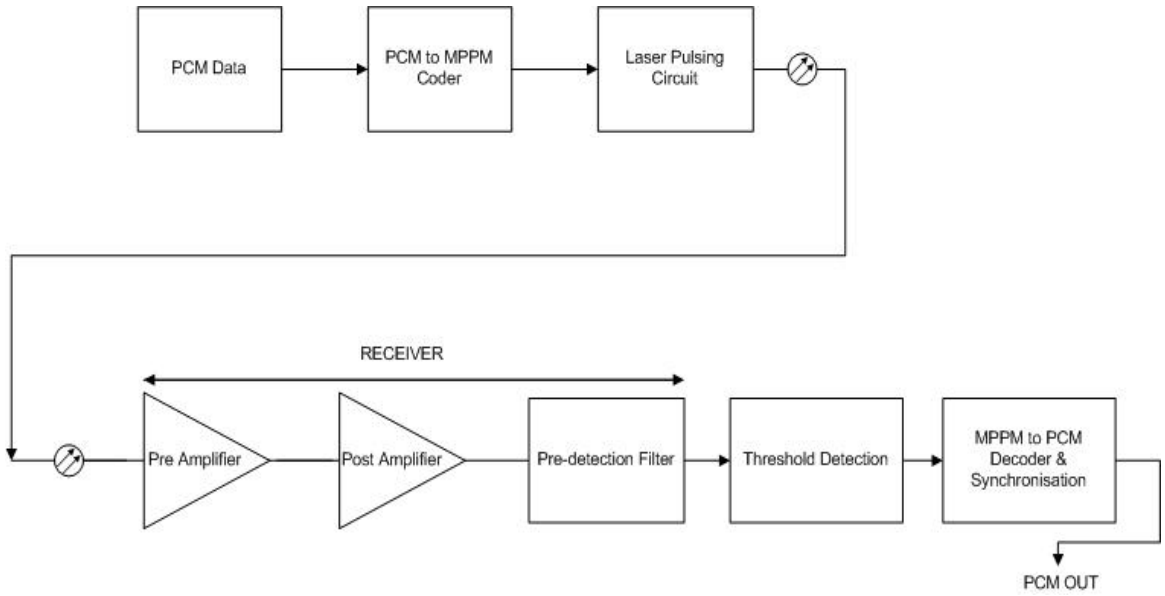


Figure 2.2: The optical fibre digital MPPM system features.

Error Probability (or Probability of Error) is a term used in mathematics and statistics. In electronics the error probability is the ratio of the number of bits, elements, characters, or blocks incorrectly received to the total number of bits, elements, characters, or blocks sent during a specified time interval. The most commonly encountered error ratio is the bit error rate (BER) which is the number of erroneous bits received divided by the total number of bits transmitted.

Maximum likelihood estimation (MLE) is a popular statistical method used for fitting a mathematical model to data. Maximum Likelihood Detection (MLD) is a (hard) detection scheme used to minimise ISI. In this research a Maximum Likelihood Sequence Detection (with threshold crossing) scheme is used (as demonstrated below). In this hard decision decoding scheme, the decoder detects all the threshold crossings as the MPPM

word to be decoded. If the number of the detected pulses is less than the number of pulses of the MPPM system (erased pulse) then the word is decoded according to the MLSD output. The average binary error probability due to an erasure can be determined by mapping the impact of all possible erasures and averaging over all of the MPPM frames. The rest error types are treated accordingly.

2.1 Pulse Detection Errors

As with digital PPM, multiple PPM systems suffer from three types of error, erasure, false alarm and wrong-slot. The following three sections present expressions for their respective error probabilities [13]-[14].

2.1.1 Erasure

Erasure errors (ER), presented in figure 2.3, are generated by noise present at the decision (sampling) time causing the amplitude of the pulse to fall below the threshold voltage. The probability of this occurring, *Pe*, is given by (1):

$$P_e = 0.5\mathrm{erfc}\left(\frac{Q_e}{\sqrt{2}}\right) \tag{1}$$

where ***Qe*** is given by (2):

$$Q_e = \frac{v_{pk} - v_d}{\sqrt{\langle n_o{}^2 \rangle}} \tag{2}$$

where the symbols have the following meanings:

V_o = the voltage level of the slot

$V_{pk} = V_o(t_{pk})$ the peak receiver output,

$V_d = V_o(t_d)$ the receiver output at the threshold crossing time t_d,

$\langle n_o^2 \rangle$ the mean square receiver output noise.

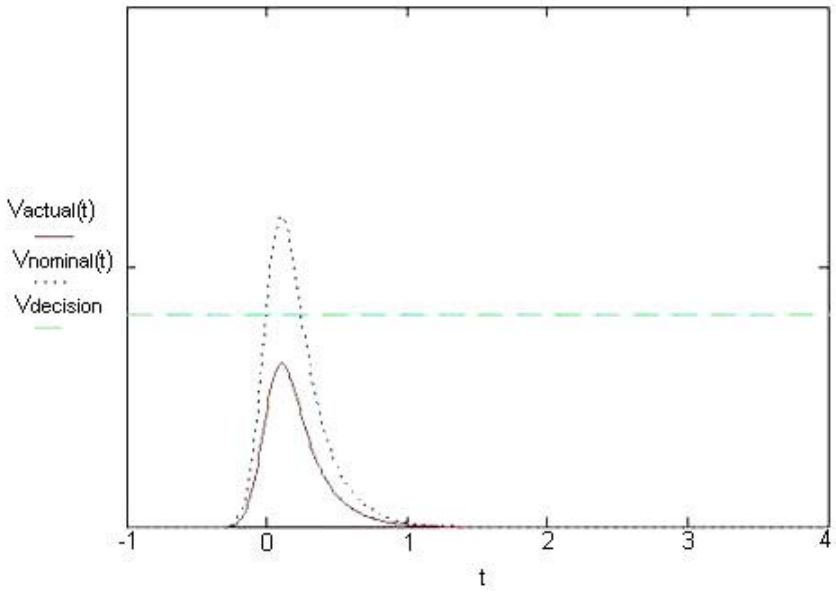


Figure 2.3: An Erasure error.

When an Erasure error occurs, one pulse is removed from the multiple PPM frame. Thus, if the frame has only 2 pulses, as with $\binom{12}{2}$ multiple PPM, only one pulse is present (only one threshold crossing). In this case the number of resultant PCM errors is found (as described by Sibley [94] and shown in table 2.1) by summing the logic 1s for each individual bit and averaging over the number of code-words. A PCM bit is assigned to logic one if its weighting is greater than 0.5, a logic 0 if it is less than 0.5, or undefined 'U' for worst case scenarios (WCS) when equal to 0.5. The bit-by-bit comparison

between the original code-word and the averaged MLSD codeword gives the average error per PCM bit for a specific multiple PPM sequence. For example, consider the codeword [1,3] which decodes to the 000001 equivalent PCM word (if a linear mapping is used). If an ER error occurs on the second pulse of the code-word caused by noise present at the decision (sampling) time the MLSD will detect the [1,?] codeword. Averaging all the codewords with a pulse in slot 1 (as shown in table 2.1) the MLSD will decode the [1,?] codeword to 000000. This generates 1 error between the original codeword [1,3] and the averaged [1,?] codeword. Considering that 6 PCM bits can be encoded the average error/PCM Bit for this MPPM sequence is 0.17.

PCM bits	**[1,?]**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**	**Errors/PCM Bit**
	[1,2]	0	0	0	0	0	0	
	[1,3]	0	0	0	0	0	1	
	[1,4]	0	0	0	0	1	0	
	[1,5]	0	0	0	0	1	1	
	[1,6]	0	0	0	1	0	0	
	[1,7]	0	0	0	1	0	1	
	[1,8]	0	0	0	1	1	0	
	[1,9]	0	0	0	1	1	1	
	[1,10]	0	0	1	0	0	0	
	[1,11]	0	0	1	0	0	1	
	[1,12]	0	0	1	0	1	0	
AVERAGE		0/11	0/11	3/11	4/11	5/11	5/11	
		0	0	0.27	0.36	0.45	0.45	
Average PCM (MLSD O/P)	[1,?]	0	0	0	0	0	0	
Original Word	[1,3]	0	0	0	0	0	1	
Error Bits	XOR	0	0	0	0	0	1	**1/6**
								0.17

Table 2.1: Decoding for Erasure error using MLSD.

2.1.2 False Alarm

For the False-Alarm Error (FA), in figure 2.4, noise in an empty slot could cause a threshold violation and so a pulse could be detected in an empty slot. The probability of this occurring is given by (3):

$$P_f = \frac{T_s}{\tau_R} 0.5\mathrm{erf}\left(\frac{Q_f}{\sqrt{2}}\right) \quad (3)$$

where,

$$Q_f = \frac{v_d - v_o(t_d)}{\sqrt{\langle n_o^{\,2} \rangle}} \quad (4)$$

and

V_o = the voltage level of the slot,

$\frac{T_S}{\tau_R}$ = the number of uncorrelated samples/time slot,

τ_R = the time at which the autocorrelation function of the filter has become small,

$V_o(t_d)$ = the signal voltage in the slot considered, which can be non-zero due to the effects of ISI,

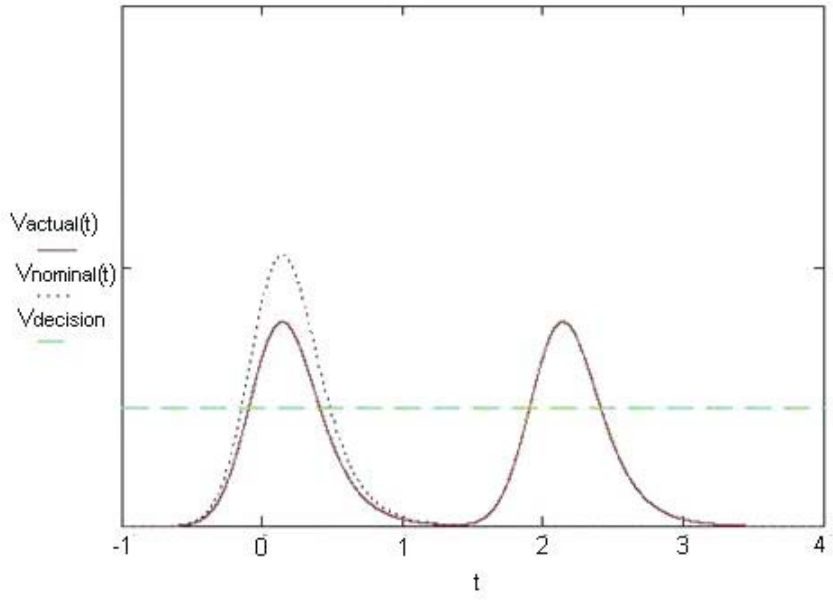


Figure 2.4: A False Alarm error.

In the case of a FA error, an extra pulse is detected in a frame. The treatment is identical to that used by Sibley [94] (presented in table 2.2). For example, consider again the codeword [1,3]. If a FA error occurs (three threshold crossings are detected) in an empty slot between the pulses the MLSD will detect [1,2,3]. Averaging all the codewords with two pulses in any of the slots 1, 2 and 3 (as shown in table 2.2) the MLSD will decode the [1,2,3] codeword to 000001. This generates 0 errors between the original codeword [1,3] and the averaged [1,2,3] (and 0 errors/PCM Bit for this MPPM sequence).

PCM bits	[1,2,3]	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Errors/PCM Bit
	[1,2]	0	0	0	0	0	0	
	[1,3]	0	0	0	0	0	1	
	[2,3]	0	0	1	0	1	1	
AVERAGE		0/3	0/3	1/3	0/3	1/3	2/3	
		0	0	0.33	0	0.33	0.67	
Average PCM (MLSD O/P)	[1,2,3]	0	0	0	0	0	1	
Original Word	[1,3]	0	0	0	0	0	1	
Error Bits	XOR	0	0	0	0	0	0	**0/6**

Table 2.2: Decoding for False Alarm error using MLSD.

2.1.3 Wrong-Slot

Noise on the leading or falling edge of a pulse can cause it to appear either before or after the current slot (figure 2.5). To minimize this error, detection should occur at the centre of the current slot. Hence the probability of a Wrong-Slot error (WS), *Ps*, is given by (5):

$$P_s = 0.5\mathrm{erfc}\left(\frac{Q_s}{\sqrt{2}}\right) \tag{5}$$

with *Qs* defined by (6):

$$Q_s = \frac{T_s}{2}\,\frac{\mathrm{slope}(t_d)}{\sqrt{\langle n_o{}^2\rangle}} \tag{6}$$

where,

T_s = the slot width,

$slope(t_d)$ = the slope of the received pulse at the threshold crossing instant t_d,

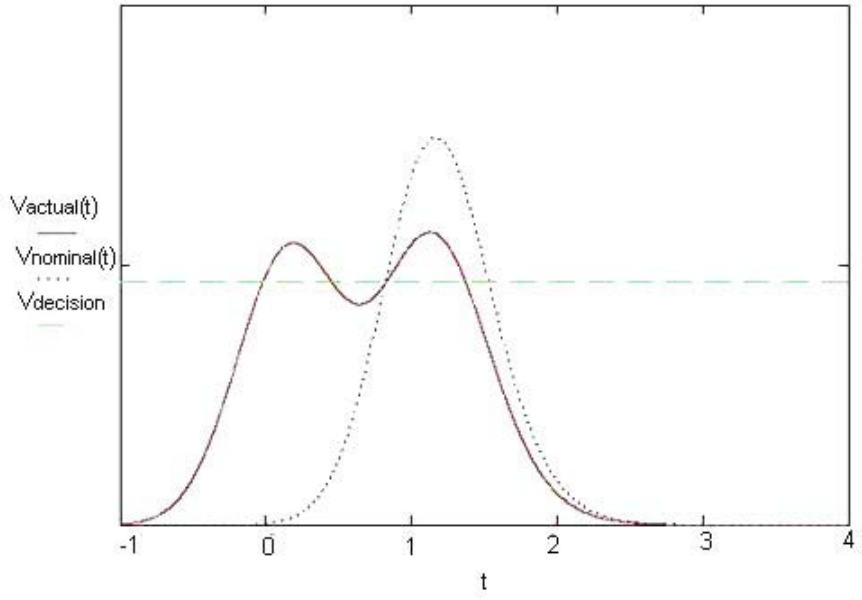


Figure 2.5: A Wrong Slot error cause by left dispersion of the pulse.

When a WS error occurs, a pulse can be detected immediately before or after the correct slot depending on the size of the dispersion and receiver noise as described by Garrett [13]-[14]. For example, consider the codeword [3,5] which decodes to the 010110 equivalent PCM word (if a linear mapping is used). If a WS error occurs on the first pulse of the code-word causing it to appear too early, the detected code-word will be [2,3,5]. If the noise and dispersion causes the pulse to appear one slot later, the code word [4,5] results. Similarly, if the WS error occurs on the 2nd pulse, the possible detected code-words might be [3,4,5] or [3,6]. Thus four possible PCM words result as shown in table

2.3. The output of the MLSD is treated as described by Sibley [94]. The bit-by-bit comparison between the original code-word and the averaged MLSD codeword gives the average error per PCM bit for a specific multiple PPM sequence and the total wrong slot error is found by averaging all possible WS code-words. A problem occurs in characterising an error as a WS error when IFI occurs on the first or last pulse in a multiple PPM frame. If a WS error occurs on a pulse in the first slot of the frame, a false pulse could occur in the last slot of the preceding frame. This would appear to give a False-Alarm error in the frame before the one under consideration. If a WS error occurs on a pulse in the last slot of the frame, the pulse could effectively move into the first slot of the following frame. Thus, the original pulse is lost and the treatment is similar to an Erasure error.

PCM bits		Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Errors/PCM Bit
First Pulse Left	[2,3]	0	0	1	0	1	1	
Error (2,3,5)	[2,5]	0	0	1	1	0	1	
	[3,5]	0	1	0	1	1	0	
AVERAGE		0/3	1/3	2/3	2/3	2/3	2/3	
		0	0.33	0.66	0.66	0.66	0.66	
Average PCM	[2,3,5]	0	0	1	1	1	1	
Original Word	[3,5]	0	1	0	1	1	0	
Error Bits	XOR	0	1	1	0	0	1	**3/6**
First Pulse Right Error (4,5)	[4,5]	0	1	1	1	1	0	
Original Word	[3,5]	0	1	0	1	1	0	
Error Bits	XOR	0	0	1	0	0	0	**1/6**
Second Pulse Left	[3,4]	0	1	0	1	0	1	
Error (3,4,5)	[3,5]	0	1	0	1	1	0	
	[4,5]	0	1	1	1	1	0	
AVERAGE		0/3	3/3	1/3	3/3	2/3	1/3	
		0	1	0.33	1	0.66	0.33	
Average PCM	[3,4,5]	0	1	0	1	1	0	
Original Word	[3,5]	0	1	0	1	1	0	
Error Bits	XOR	0	0	0	0	0	0	**0/6**
Second Pulse Right Error (3,6)	[3,6]	0	1	0	1	1	1	
Original Word	[3,5]	0	1	0	1	1	0	
Error Bits	XOR	0	0	0	0	0	1	**1/6**
Av. Error/PCM Bit for [3,5]								**5/24=0.20833**

Table 2.3: Decoding for Wrong-Slot error using MLSD.

2.2 ISI and IFI effects

In order to calculate the error probability and the effects of ISI and IFI, Sibley [94] considered specific pulse sequences such as *0* and *1* (called standard error) and 1*0*, 11*0*, *1*1, 1*1*, 1*0*1, 10*1* and 10*1*1 caused by ISI or IFI of adjacent slots with the symbol in error being represented in *italics*. The error probability was determined by applying the MLSD decoding and then weighting by the probability that the particular sequence occurs. This process can be very tedious and time consuming. For example, in a $\binom{12}{2}$ multiple PPM system, 12 erasure, 220 false alarm and 232 wrong slot errors need to be considered. This becomes even worse if the system is large such as $\binom{22}{5}$.

Therefore, an automated analysis is needed (referred to as complete analysis because it considers every possible error sequence). However, if this methodology is expanded to larger systems, a very large number of sequences need to be considered so making this analysis again very prolonged. The solution was the development of a new analysis (referred as simplified analysis) that only considers the most important and common sequences in any $\binom{X}{Y}$ multiple PPM system. This simplification can be achieved because some particular sequences rarely occur and so have a negligible effect on the equivalent PCM error rate. Some sequences can be further grouped and so only few sequences (discussed in the next chapter) need to be considered to predict the

performance of any $\begin{pmatrix} X \\ Y \end{pmatrix}$ multiple PPM system, even when the effects of ISI and IFI are accounted for.

Chapter 3

Design, Implementation and Testing

One of the deliverables of this research was an automated solution that can calculate the PCM error probabilities of specific error sequences taking into account inter-symbol (ISI) and inter-frame interference (IFI) using a MLSD scheme. A hardware solution was investigated first using a Higher (Abstract) Description Language (like VHDL) because of the high speed performance of hardware. Unfortunately, a total generic hardware design [114]-[115] cannot be developed using a hardware approach (inputs and outputs need to be clearly stated beforehand). Attempts to implement smaller generic hardware designs for a range of values (i.e. for $100<X<2$, where X is the number of slots) failed in the synthesis process (topology problems cause of space shortage in the FPGA solution, and vast synthesis and implementation time). Hence, the development of a generic and more versatile implementation was only through software.

3.1 Software Design

In order to design a complicated software solution, a design methodology [116]-[124] should be chosen first. A modified waterfall design methodology, as shown in figure 3.1 (appendix A), was adopted. The simplicity of this methodology and the fact that the software is mainly algorithmic (no use of classes and objects), with low probability of concept or vast design changes, makes this choice the most appropriate.

The main difference with the traditional waterfall design methodology is that testing is implemented for every part (algorithm) of the software, instead of just at the end of the implementation phase, to minimize the possibility of failure.

The requirement analysis is implemented through possible scenarios and different software diagrams. Therefore, a possible scenario (showing the specifications of the solution) for the software is: "The user will be able to find the equivalent PCM error rates of specific or generic sequences of any MPPM system that affect the final error probability in a MLS Detection scheme. The user will enter the characteristics of the MPPM system: 1) the number of slots (X), 2) the number of pulse(s) (Y), 3) the error type (Erasure, False Alarm, Wrong Slot), 4) the choice of mapping between Linear Increment\Decrement, Gray Codes, Random mapping and other predefined mappings (in some cases the range or starting data are needed), 5) the choice of the MLS Detection algorithm (to consider specific or generic error sequences). The generated results (error rates or MPPM/PCM codewords/mappings referred as statistics) will be displayed or saved. The user can exit the program at any time by entering the "escape sequence" (-1) at any time. The program will check the validity of the user inputs and will display error messages and instructions. For some data (like Pascal's Number, encoded PCM bits) the program will automatically generate results but the user will have to enter them manually for lower risk of failure."

A top-down analysis was followed for the design phase and a bottom up for the implementation phase. A Use Case (level-0 and 1 detail) and Sequence diagrams were

developed using MagicDraw©. The diagrams presented in figures 3.2, 3.3 and 3.4 (presented in appendix A), are implemented using the Unified Modeling Language (UML©) [125]-[126] to identify the complete (and any hidden) functionality of the program. From the diagrams the user should be able to:

1) Choose Error Type:
    i) Erasure.
    ii) False Alarm.
    iii) Wrong Slot.
2) Choose Performance Algorithm:
    i) Complete (complete analysis).
    ii) 2-Pulse (simplified analysis).
3) Choose Mapping:
    i) Linear Increment\Decrement.
    ii) Gray Codes.
    iii) Random.
    iv) Read (a predefined user) Mapping.
4) Other:
    i) Enter MPPM System\Mapping\Performance Algorithm Details.
    ii) Enter Number of Slots\Pulse(s)\encoded PCM Bits.
    iii) Enter Number of Erasure\False Alarm\Wrong Slot Sequences.
    iv) Enter Range of Data or Start Number.

The program should be able to:

1) Create Success/Failure Messages.
2) Display or Save various Statistics (PCM equivalent error rates).
3) Produce Instructions.
4) Check Input Data.

Where, level-0 detail is represented by decimal and level-1 by Latin numbering.

From the sequence diagram the user:

1) First enters the MPPM System Details:
   i) *X* (Number of Slots), *Y* (Number of Pulses), Error Type, number of encoded PCM bits, number of Erasure\False Alarm\Wrong Slot Sequences
2) then chooses mapping and
3) enters range or starting data
4) and finally chooses action:
   i) display\save Sequences\Rates (Statistics).

The main purpose of the software (s\w) is to simulate a MPPM optical link with the use of a MLSD. The software solution (presented in appendix C) is implemented

through algorithms as presented in figure 3.5 (appendix A). PCM data words generated from different mapping algorithms are encoded into MPPM code-words. If an error occurs in the MPPM code-words, the MLSD algorithm generates all possible error codewords alongside the averaged (MLSD) data. The straight comparison between (MLSD) averaged and original data generates the number of equivalent PCM errors for specific error sequences. These numbers, expressed as probabilities (equivalent PCM error rates), are used in mathematical models to find the (total) error probability of a system.

The software was developed as a Windows32© application (to ease implementation of interface) using Visual C++© version 6. The software program is divided into 3 main parts:

1) Interface (a console and a visual interface were implemented).
2) Main Program (where the main algorithms and processing are implemented).
3) Functions (Printing Functions).

The program can theoretically calculate the equivalent PCM error rates of any $\binom{X}{Y}$ MPPM system. But, for large X, Y numbers (or large X-Y difference), the limitation is the computer memory, the software data types (numbers up to 32 binary bits can only be used) and the processing time (all these are affected by the PC characteristics). Therefore, all algorithms are time and space (memory usage) analyzed [127]-[131] and then, (if possible) optimised [127]. The program is very versatile i.e. it can be easily

changed for calculating error rates for more than one error per frame (error/frame) or to detect new error sequences. The possibility of more than one error/frame is very small, and that is why it was neglected in this research.

The implementation and testing of the program was carried out on a Personal Computer (PC) with the following characteristics:

1) AMD© 32-bit 2GHz CPU.
2) 1 GB of RAM.
3) 1 TB of hard drive disc space.

If a PC with better characteristics (especially in CPU and RAM memory) is used a better performance will be achieved. The Console32 Interface is presented in figure 3.6 (appendix A) and the Visual Interface (implemented as a Microsoft© Foundation Class Library-MFC- application) is presented in figure 3.7 (appendix A). The two Interfaces are totally (function) equivalent, despite some small differences in the design.

The software simulates a MLS Detection scheme. In this system, registers are created according to the multiple PPM system using Dynamic Memory Allocation [127] (generic registers and arrays without predefined size can be created) and every possible code-word of the system is generated. The size of the $\binom{X}{Y}$ MPPM system will be given by Pascal's Number. This number gives the maximum number of combinations between two numbers. Thus,

$$\binom{X}{Y} = \left(\frac{X!}{Y!(X-Y)!} \right) \tag{7}$$

where,

$$\left(\frac{X!}{Y!(X-Y)!} \right) = \left(\frac{X*(X-1)*(X-2)*(X-3)...*1}{(Y*(Y-1)*(Y-2)*(Y-3)...*1)*((X-Y)*(X-Y-1)*(X-Y-2)...*1)} \right)$$

MPPM systems with encoding capabilities above 32-bits (2^{32} = 4,294,967,296 data range) are ignored, as this is considered to be a hardware implementation limit. A function was constructed to generate Pascal's Number. If Pascal's Number is above 12! (factorial) because of the bit overflow there isn't a software data type that can describe a number greater than 32-bits. Recursion was always a problem in the software because it can easily get out of hand in terms of memory-space. A more efficient way to achieve recursion is through iteration, but that can also fail if *X* and *Y* can be any integer. The (time and space) optimization of the algorithm was solved by canceling common factorials. More details are:

i) If Y<X-Y.

$$\binom{X}{Y} = \left(\frac{(X-Y)!*(X-Y+1)*(X-Y+2)...*X}{(Y!*(X-Y)!} \right) = \left(\frac{(X-Y+1)*(X-Y+2)...*X}{Y!} \right)$$

ii) Else Y>X-Y.

$$\binom{X}{Y} = \left(\frac{Y!*(Y+1)*(Y+2)...*X}{(Y!*(X-Y)!} \right) = \left(\frac{(Y+1)*(Y+2)...*X}{(X-Y)!} \right)$$

The only possibilities now that the function will fail are:

i) If Y!>12! or $(X-Y+1)*(X-Y+2)...*X$ >12!.

ii) If (X-Y)!>12! or $(Y+1)*(Y+2)...*X$ >12!.

In a software program the compiler reads a program line by line (multithreading programming is still not popular because of the complexity involved). This is also the case why hardware solutions are faster. If for every line of code the compiler needs *T* seconds (where *T* is the CPU clock time) the previously described function (algorithm) has the following characteristics:

Algorithm 1: Pascal's Number Calculation

Time Estimation:

Best Case: (4+(2*Y))*T Worst Case: (4+(2*(X-Y)))*T

where T is the CPU period, X the number of slots and Y the number of pulses

Space Estimation: 240 bits used

More details for the time and space analysis of the software are presented in the software printout (comments) in appendix C.

Ideally, the size of the $\binom{n}{k}$ MPPM system should be a power of two to ease implementation but for $k > 1$ this is rarely the case. Therefore, the predefined (power) function can be used to generate the correct MPPM system size (where *PCM_BITS* is the encoding capability of the system). For example, a $\binom{12}{2}$ multiple PPM system uses a 12-slot frame with 2 data pulses to code 6 bits of PCM data. This gives 64 valid multiple PPM frames (calculated with the predefined function) instead of the available 66 (calculated with the user defined function previously described).

Having the size of the $\binom{X}{Y}$ MPPM system ($2^{PCM\_BITS} * X$), registers are created according to the multiple PPM system and every possible code-word of the system is generated, where digital 1s and 0s represent occupied and empty multiple PPM slots as showed in figure 3.8 (appendix A). As an example, the [1,2] code-word in a $\binom{12}{2}$ multiple PPM system is represented as 110000000000. The rest of the code-words are represented accordingly. The algorithm created to generate the MPPM mapping is showed in figure 3.9 (Data Flow diagram - appendix A). The algorithm inputs are:

i) X, Y (number of slots and pulses).

ii) Size (of the MPPM array-Pascal's Number or 2^{PCM_BITS}).

For the $\binom{X}{Y}$ MPPM system (with *PCM_BITS* encoding capability) the algorithm will have the following steps:

1) An array of size $X * 2^{PCM\_BITS}$ is created.
2) The first Y slots are set to 1 (to represent the 1st row).
3) The last two pulses (named *Y2* and *Y1*) are pointed.
4) If *Y1* is not in the last slot (X):
   i) All the slots until pulse *Y2* are copied in the next row and *Y1* is Serial Right Shifted (SRS) by one slot.
   ii) Step 4-i) is repeated until *Y1* is on slot X (last slot).
5) If *Y1* cannot be shifted further, all the slots before *Y2* are copied, *Y2* is SR Shifted and a pulse is added on the slot right of *Y2*.
6) Steps 3 to 5 are repeated (increasing the number of pulses added when step 5 is repeated) and
7) exit when there is no possible shifting operation (all pulses are shifted at the end of the row without any empty slots between the pulses).

The algorithm has the following characteristics:

Algorithm 2: MPPM Mapping

Time Estimation: $2^{\text{PCM BITS}}*(X+2)*T$

where PCM BITS is the number of encoded PCM bits

Space Estimation: 256 bits used

A sample of testing is presented in table 3.1 for a variety of MPPM systems.

System	MPPM Register (maximum response<1sec)	
	Codeword	Software Representation
12-1	[1]	100000000000
	[7]	000000100000
12-2	[1,4]	100100000000
	[2,8]	010000010000
	[10,11]	000000000110
12-5	[1,2,3,4,5]	111110000000
	[2,3,5,6,7]	011011100000
	[6,7,8,9,10]	000001111100
	[7,8,9,10,11]	000000111110
4-2	[1,2]	1100
	[1,4]	1001
	[3,4]	0011
7-5	[1,2,3,4,5]	1111100
	[2,3,4,5,7]	0111101
22-11	[1,2,3,4,5,6,7,8,9,10,11]	11111111111000000000000
	[2,4,6,9,10,11,12,16,19,21,22]	01010100111100010010011
	[3,4,5,7,9,10,11,15,17,18,19]	00111010111000101110000

Table 3.1: Testing results from the MPPM mapping algorithm.

From table 3.1 it is clear that the algorithm generates accurate and time efficient results (below<1sec). The next step was to create the data mapping algorithm (with range

of numbers between 0 and 2^{PCM_BITS}). All mappings are without redundancy and the final numbers are converted into binary form. More details are:

i) Linear Increment/Decrement is implemented by a simple Up/Down software counter (user enters the starting value-default is zero).

ii) The random mapping is generated using the predefined random function that generates totally random numbers (the Windows® clock is used as a "seed" value to the function for totally random values). Every random number is checked for repetition (if the number is repeated, it is then discarded and a new number is generated).

iii) The program can read a predefined user mapping (using a pointer) in the (Windows®) notepad.

iv) Gray Codes are created using the algorithm where bit *I* of a Gray Code is 0 if bits *I* and *I*+1 of the corresponding binary codeword are the same, or bit *I* is 1 (when *I* is the Most Significant Bit (MSB), bit *I*+1 of the binary codeword is considered to be 0).

The decimal to binary conversion algorithm is presented in figure 3.10 (appendix A). The algorithm uses successive divisions and modulus divisions of powers of 2. The algorithm has the following characteristics:

Algorithm 3: Decimal to Binary Conversion

Time Estimation: 3*PCM BITS*$2^{PCM\ BITS}$*T

Space Estimation: 48 bits used

A sample of test results obtained from the Data Mapping (register with size $2^{PCM\_BITS} * PCM\_BITS$) algorithm is presented in table 3.2.

Codeword	Data Register (maximum response<1sec)				
	Linear Increment	Linear Decrement	Random	User Defined Mapping	Gray Codes
[1,2]	000000	111111	000001	010100	000000
[1,3]	000001	111110	000010	100101	000001
[1,4]	000010	111101	010001	101101	000011
[1,5]	000011	111100	011000	110101	000010
[1,6]	000100	111011	100001	011101	000110
[1,7]	000101	111010	101000	011111	000111
[1,8]	000110	111001	100000	010011	000101
[1,9]	000111	111000	010000	010101	000100
[1,10]	001000	110111	001000	010100	001100
[1,11]	001001	110110	000100	100101	001101
[1,12]	001010	110101	000000	101101	001111
[2,12]	010100	101011	100100	100010	011110
[3,5]	010110	101001	101100	110011	011101
[3,8]	011001	100110	110100	101011	010101
[4,7]	100000	011111	111100	111011	110000
[8,11]	111010	000101	111101	100011	100111

Table 3.2: Testing results from the Data Mapping algorithm.

Therefore, a new register (referred as encoder) is created integrating the PCM data and MPPM mapping (with size equal to the sum of the two integrated registers). As,

an example consider the [1,2] codeword of a $\binom{12}{2}$ MPPM system if a linear increment (starting from zero) mapping was used. The encoder mapping will be 110000000000|000000 where the first 12 digits represent the MPPM word [1,2], and the last 6 digits represent the encoded PCM data. Thus, the encoder register was created to simulate "a real" MPPM encoder where PCM data are encoded to the transmitted MPPM data.

Table 3.3 demonstrates a sample of test results of the encoder register (with size $2^{PCM\_BITS} * (X + PCM\_BITS)$) using three different mappings.

Codeword	**Decoder Register (maximum response<1sec)**		
	Linear Increment	**Linear Decrement**	**Gray Codes**
[1,2]	110000000000|000000	110000000000|111111	110000000000|000000
[1,3]	101000000000|000001	101000000000|111110	101000000000|000001
[1,4]	100100000000|000010	100100000000|111101	100100000000|000011
[1,5]	100010000000|000011	100010000000|111100	100010000000|000010
[1,6]	100001000000|000100	100001000000|111011	100001000000|000110
[1,7]	100000100000|000101	100000100000|111010	100000100000|000111
[1,8]	100000010000|000110	100000010000|111001	100000010000|000101
[1,9]	100000001000|000111	100000001000|111000	100000001000|000100
[1,10]	100000000100|001000	100000000100|110111	100000000100|001100
[1,11]	100000000010|001001	100000000010|110110	100000000010|001101
[1,12]	100000000001|001010	100000000001|110101	100000000001|001111
[2,3]	011000000000|001011	011000000000|110100	011000000000|001110
[2,4]	010100000000|001100	010100000000|110011	010100000000|001010
[2,5]	010010000000|001101	010010000000|110010	010010000000|001011
[2,7]	010000100000|001111	010000100000|110000	010000100000|001000
[3,5]	001010000000|010110	001010000000|101001	001010000000|011101

Table 3.3: Testing results from the encoder algorithm.

The total error probability (P_{TOTAL}) is given by the sum of the error probabilities (8) of the three error types. Thus, the three error types should be analyzed first.

$$\boldsymbol{P_{TOTAL} = P_{erasure} + P_{false\_alarm} + P_{wrong\_slot}} \quad (8)$$

3.1.1 Erasure Errors

The analysis of the ER error prerequisites the creation of the ER averaged (MLSD) mapping. This register (with size of $\binom{X}{Y-1}$*(*X*+*PCM_BITS*)) contains all possible erasure code-words with the corresponding average data. As an example, consider again the $\binom{12}{2}$ MPPM system using a Linear (starting from 0) mapping for PCM data. This system has 12 ($\binom{12}{1}$) possible erasure code-words (from [1,?] to [12,?]). For the [1,?] codeword all code-words with a pulse in slot 1 are averaged. The MLSD for the [1,?] codeword is 000000 (the treatment is similar to that presented in chapter 2.2.1). Thus, in the ER (averaged) mapping the first row of the register is 100000000000|000000, where the first 12 bits represent the [1,?] erasure MPPM codeword (with only one pulse in slot 1) and the rest of them the averaged PCM data. As previously described a PCM bit is assigned to logic one if it's weighting is greater than 0.5, a logic 0 if it is less than 0.5, or undefined 'U' for worst case scenarios (WCS) when equal to 0.5. In this case the maximum number of errors is generated from the MLSD.

The algorithm that implements the ER MLSD register is presented in figure 3.11 (appendix A). The algorithm can be described through the following steps:

1) An array of size $\binom{X}{Y-1}$*(X+PCM_BITS) is created.
2) Every MPPM codeword (from the encoder register) is scanned from every ER MPPM codeword (from the ER averaged register).
3) If the two codewords are similar to *X*-1 slots (1 error bit per frame) the PCM equivalent data of the MPPM codeword are stored.
4) If *X*-*Y*+1 MPPM codewords are found (scanned) then every PCM bit of the ER averaged register (MLSD) is averaged (weighted) according to PCM data from the scanned MPPM codewords.
5) Steps 2 and 3 are repeated for every ER averaged codeword and
6) Exit if all the codewords are averaged.

The algorithm has the following characteristics:

Algorithm 4: ER MLSD

Time Estimation:

(PASC2*(2*X) + 2 + (2*PASC2) + (PCM BITS*PASC2))*T

where PASC2 is the number of erasure combinations

Space Estimation: 208 bits used

A sample of tests is showed in table 3.4.

ER C/W	BIT POSITION												Averaged PCM Data
	11	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
[1,?]	1	0	0	0	0	0	0	0	0	0	0	0	000000
[2,?]	0	1	0	0	0	0	0	0	0	0	0	0	000000
[3,?]	0	0	1	0	0	0	0	0	0	0	0	0	011001
[4,?]	0	0	0	1	0	0	0	0	0	0	0	0	100100
[5,?]	0	0	0	0	1	0	0	0	0	0	0	0	101110
[6,?]	0	0	0	0	0	1	0	0	0	0	0	0	100110
[7,?]	0	0	0	0	0	0	1	0	0	0	0	0	110101
[8,?]	0	0	0	0	0	0	0	1	0	0	0	0	111000
[9,?]	0	0	0	0	0	0	0	0	1	0	0	0	111100
[10,?]	0	0	0	0	0	0	0	0	0	1	0	0	111*22*
[11,?]	0	0	0	0	0	0	0	0	0	0	1	0	111*22*1
[12,?]	0	0	0	0	0	0	0	0	0	0	0	1	111110
[1,2,?]	1	1	0	0	0	0	0	0	0	0	0	0	0000*02*
[3,5,?]	0	0	1	0	1	0	0	0	0	0	0	0	1101*2*00
[7,8,?]	0	0	0	0	0	0	1	1	0	0	0	0	1111001
[1,2,3,?]	1	1	1	0	0	0	0	0	0	0	0	0	00000000
[7,9,10,11,?]	0	0	0	0	0	0	1	0	1	1	1	0	101000001

Table 3.4: A sample of ER codewords alongside the averaged data from the MLSD in a $\binom{12}{2}$, $\binom{12}{3}$, $\binom{12}{4}$ *and* $\binom{12}{5}$ *systems. Number (2) in italics symbolizes Worst Case Scenario (WCS).*

Therefore, the software so far has created 2 registers: 1) a MPPM (encoder) register (that holds the MPPM code-words plus the equivalent PCM data) and 2) the averaged ER MLSD register (that holds all the ER MPPM code-words plus the averaged PCM data. The only thing missing is the number of generated (PCM) ER errors (between averaged and original PCM data). Thus, a new register was needed called ER PCM error (with size of ($(X+2^{PCM\_BITS}+Y) * 2^{PCM\_BITS}$)). The last Y digits represent the ER equivalent

PCM errors (the algorithm that implements this register is presented in figure 3.12 - appendix A). For example, for the $\binom{12}{2}$ system (using a Linear Increment mapping) the first row of the corresponding register is 110000000000000000 0 0. The first 12 digits represent the MPPM codeword [1,2] (2 pulses and 10 empty slots), the following 6 digits represent the encoded PCM data, and the last two digits represent the number of errors between original data and averaged data from the MLSD. Hence, for the [1,2] codeword the last two digits show that there are no errors generated if an erasure error is generated in the first, [1,?], or the second, [?,2], pulse. The algorithm (which is implemented in parallel with the previous algorithm for time optimization) can be illustrated through the following steps:

1) Every MPPM codeword (from the encoder register) is scanned from every ER MPPM codeword (from the ER averaged register).
2) If the two codewords are similar to X-1 slots (1 error bit per frame) the PCM equivalent data of the MPPM codeword are compared (X-OR) with the averaged data of the ER MPPM codeword.
3) The number of the generated (error) bits is stored in the correct position in an array of size $(X+2^{PCM\_BITS}+Y) * 2^{PCM\_BITS}$.
4) Exit if all the codewords are compared.

The algorithm has the following characteristics:

Algorithm 5: ER PCM

Time Estimation:

Best Case: $(6+(2*PASCAL2))*2^{PCM\ BITS}*T$

Worst Case: $(6+(2*PASCAL2)+(3*PASC2*PCM\ BITS))*2^{PCM\ BITS}*T$

Space Estimation: 368 bits used

A sample of test results of the ER PCM error register, are presented in table 3.5.

Codeword [x,y]	ER PCM error Register (maximum response<1sec)	Error in the 6-Bit PCM word if Pulse 1 or 2 is ERASED	
		1 [?,y]	2 [x,?]
[1,2]	110000000000000000 0 0	0	0
[1,3]	101000000000000001 2 1	2	1
[1,4]	100100000000000010 3 1	3	1
[1,5]	100010000000000011 4 2	4	2
[1,6]	100001000000000100 2 1	2	1
[1,7]	100000100000000101 2 2	2	2
[1,8]	100000010000000110 5 2	5	2
[1,9]	100000001000000111 5 3	5	3
[1,10]	100000000100001000 4 1	4	1
[1,11]	100000000010001001 4 2	4	2
[1,12]	100000000001001010 3 2	3	2
[2,3]	011000000000001011 2 3	0	0
[2,4]	010100000000001100 2 2	2	1
[2,5]	010010000000001101 3 3	3	1
[2,7]	010000100000001111 4 4	4	2
[3,5]	001010000000010110 3 4	2	1
[10,11]	000000000110111111 2 3	2	2

Table 3.5: A sample of results of the ER PCM register for a 12-2 MPPM system.

As a threshold crossing detector is assumed, erasure errors can only occur in the slot containing the pulse. Static noise (and left-dispersion) can make a single pulse to be erased (right dispersion is considered as a Wrong Slot error). This error, referred to as a standard error, occurs when the adjacent slots of an optica l pulse are empty (00100). The other error sequences are caused by ISI and IFI. Therefore, for a $\binom{12}{2}$ MPPM system the possible erasure sequences are demonstrated in table 3.6.

System	ER Error Sequences
$\binom{12}{2}$	***1***
	11
	111
	1***1***
	11***1***
	10***1***
	110***1***
	1010***1***
	10***1***1
	101***1***
	1101
	1010***1***

Table 3.6: Erasure Error Sequences considered by a MLSD in a 12-2 MPPM system with the symbol in error being represented in italics.

It is also evident that if a larger MPPM system is examined, a larger number of error sequences is needed. The algorithm (presented in figure 3.13 - appendix A) referred to as original, or complete (because most of the error sequences are considered), locates a

possible ER sequence and calculates the equivalent PCM error rate of the sequence, by dividing the total number of PCM error bits and then weighting by the probability that the particular sequence occurs. Therefore, for the $\binom{12}{2}$ MPPM system, every ER error sequence should be divided by 64 (valid MPPM codewords) and then by 6 (number of encoded PCM bits) to obtain the equivalent error rate per PCM bit of the sequence. The complete algorithm is illustrated in the following steps:

1) For every row of the ER PCM register, pulses are located (possible erasure errors).
2) For every pulse, the distance of the pulse from the previous or the next frame is checked (for IFI).
3) If the pulse is isolated, the error is a standard error, otherwise other pulses are located (for ISI) and the correct sequence is detected.
4) The number of PCM errors for every sequence is stored.
5) When all ER PCM rows are scanned the total error sum of every sequence is then weighted by the probability that the particular sequence occurs.
6) These probabilities are stored and the algorithm exits.

The algorithm has the following characteristics:

Algorithm 6: ER Sequence Complete

Time Estimation:

Best Case: 64*T Worst Case: ((64*T)+((4*(2*Y+3))T)+(2*Y*T))*N

where N is the number of erasure sequences

Space Estimation: (240+(32*Y+3))*N bits used

The complete algorithm is followed by a full mathematical analysis. This process can be very tedious and time consuming (924 combinations need to be considered just for ER errors in a $\binom{12}{7}$ multiple PPM system). The solution is that a number of sequences can be neglected, while others can be grouped with other sequences. For example, 101*1* is considered as 1*1*, 1*1*1 as 1*1* or *1*1 (depending on ISI or IFI) and 10*1*1 is considered as *1*1 (10*1*1 is considered as a Wrong Slot sequence only). These sequences are grouped because they have a very low probability to produce an Erasure error (right dispersion of a pulse) and grouped with more common sequences, or they have a small impact on the final error probability. Furthermore, sequences with trailing pulses are even more simplified. As a result, error sequences like 11111*1*, 1111*1* and 111*1* can be grouped and considered as 11*1*. This is due to the fact that the impact of more than two pulses (in this case in an erased pulse) is equivalent to the impact generated by two adjacent pulses. Because of this characteristic, the simplified algorithm is called a 2-pulse algorithm.

Considering grouping and the low error probability of some sequences, only 7 sequences (for an ER error for MPPM systems that can encode 32 PCM bits) shown in

table 3.7 need to be considered to predict the ER performance of any $\binom{X}{Y}$ multiple PPM system, even when the effects of ISI and IFI are accounted for.

Erasure Error Sequences	2-Pulse
	1
	11
	1***1***
	11***1***
	10***1***
	110***1***
	1010***1***

Table 3.7: Erasure Sequences considered from the simplified algorithm for an X-Y MPPM system with symbol in error being represented in italics (1 alone is called a standard error).

3.1.2 False Alarm Errors

False alarm errors can only occur in the time (empty) slots preceding the pulse. The treatment for False Alarm Errors is identical to that used for an ER error (same algorithms are used). The only differences are:

i) in the FA MLSD Register (table 3.8) where every possible FA code-word is generated, an extra pulse is added and not deleted, and,

ii) in the FA PCM error register, errors are generated for every empty slot and not for every pulse of the frame (table 3.9).

Original Codeword	FA Codewords	FA MLSD Register (maximum response<1sec)
[1,2]	[1,2,3]	111000000000|000001
	[1,2,4]	110100000000|000000
	[1,2,5]	110010000000|000001
	[1,2,6]	110001000000|000100
	[1,2,7]	110000100000|000101
	[1,2,8]	110000010000|000000
	[1,2,9]	110000001000|000001
	[1,2,10]	110000000100|000000
	[1,2,11]	110000000010|000001
	[1,2,12]	110000000001|000000

Table 3.8: A sample of results of the FA MLSD register in a 12-2 MPPM system.

Therefore, as an example, consider the [1,2] codeword of a $\binom{12}{2}$ MPPM system where 10 possible FA codewords can be generated. For a FA error in slot 3, the [1,2,3] codeword is received. The MLSD register for this codeword is 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1, where the first 12 bits represent the [1,2,3] MPPM (FA) codeword, and the 6 remaining bits the MLSD (averaged) data. In the FA PCM register, the same codeword [1,2] is represented as 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 2 0 1 0 1 0. The first 12 bits represent the MPPM codeword, [1,2], the following 6 bits represent the encoded PCM data of the [1,2] MPPM codeword and the rest of the bits represent the PCM FA errors generated between original and (averaged) MLSD data. Hence, for a FA error in slot 3, the [1,2,3] codeword is received, generating 1 PCM error. The rest of the bits are represented accordingly.

Codeword [x,y]	FA PCM error Register (maximum response<1sec)	FA PCM errors for a FA error in the Empty Slot									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
[1,2]	1100000000000000|1011201010	1	0	1	1	2	0	1	0	1	0
[1,3]	1010000000000000|0011001111	0	0	1	1	0	0	1	1	1	1
[1,4]	1001000000000000|1201100010	1	2	0	1	1	0	0	0	1	0
[1,5]	1000100000000000|1012110212	1	0	1	2	1	1	0	2	1	2
[1,6]	1000010000000000|0111112122	0	1	1	1	1	1	2	1	2	2
[1,7]	1000001000000000|0121010001	0	1	2	1	0	1	0	0	0	1
[1,8]	1000000100000000|2311010322	2	3	1	1	0	1	0	3	2	2
[1,9]	1000000010000000|2121011322	2	1	2	1	0	1	1	3	2	2
[1,10]	1000000001000000|1121130111	1	1	2	1	1	3	0	1	1	1
[1,11]	1000000000100000|1021122102	1	0	2	1	1	2	2	1	0	2
[1,12]	1000000000010000|2210130112	2	2	1	0	1	3	0	1	1	2

Table 3.9: A sample of results of the FA PCM register in a 12-2 MPPM system.

For the $\binom{12}{2}$ MPPM system, every FA sequence should be divided by 10 (number of possible FA codewords for every MPPM codeword), then by 64 (valid MPPM codewords) and finally by 6 (number of encoded PCM bits) to obtain the equivalent error rate per PCM bit of the sequence. The sequences considered for FA errors using the simplified algorithm are demonstrated in table 3.10.

False Alarm Error Sequences	2-Pulse
	0
	1***0***
	11***0***
	1***0***1
	11***0***1

Table 3.10: False Alarm Sequences considered from the simplified algorithm for an X-Y MPPM system with symbol in error being represented in italics (0 alone is called standard error).

3.1.3 Wrong Slot Errors

When a WS error occurs, a pulse can be detected immediately before or after the correct slot depending on the size of the dispersion and receiver noise as described by Garrett [3, 4]. A problem occurs in characterising an error as a WS error when IFI occurs on the first or last pulse in a multiple PPM frame. If a WS error occurs on a pulse in the first slot of the frame, a false pulse could occur in the last slot of the *preceding* frame. This would appear to give a False-Alarm error in the frame before the one under consideration. If a WS error occurs on a pulse in the last slot of the frame, the pulse could effectively move into the first slot of the *following* frame. Thus the original pulse is lost and the treatment is similar to an Erasure error.

Therefore, a WS error can be treated as Erasure (ER MLSD register is needed), False Alarm (FA MLSD register is also needed) or as a "normal" WS error (where the treatment is the straight comparison between original and received data). The WS PCM

register is presented in table 3.11. As an example consider the [1,2] codeword of a $\binom{12}{2}$ MPPM system where 4 different WS errors can occur. The [1,2] codeword is represented in the WS MLSD register as 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1. The first 12 bits represent the [1,2] MPPM codeword, the following 6 bits represent the encoding PCM data and the rest of the bits represent the PCM errors generated from the 4 possible WS errors. Therefore, only the right dispersion of the second pulse generates 1 PCM error.

Codeword [x,y]	**WS MLSD Register (maximum response<1sec)**	**WS PCM Errors for a WS error in the Pulse**			
		1		**2**	
		Dispersion		**Dispersion**	
		Left	**Right**	**Left**	**Right**
[1,2]	110000000000000000|0001	0	0	0	1
[1,3]	101000000000000000|0202	0	2	0	2
[1,4]	100100000000000000|0321	0	3	2	1
[1,5]	100010000000000000|0313	0	3	1	3
[1,6]	100001000000000000|0211	0	2	1	1
[1,7]	100000100000000000|0202	0	2	0	2
[1,8]	100000010000000000|0311	0	3	1	1
[1,9]	100000001000000000|0314	0	3	1	4
[1,10]	100000000100000000|0311	0	3	1	1
[1,11]	100000000010000000|0302	0	3	0	2
[1,12]	100000000001000000|0422	0	4	2	2

Table 3.11: A sample of results of the WS PCM register for a 12-2 MPPM system.

The sequences considered for WS errors using the simplified algorithm are demonstrated in table 3.12.

Wrong Slot Error Sequences	2-Pulse
	1
	11
	1***1***
	11***1***
	10***1***
	110***1***
	10***1***1
	10***1***11
	110***1***1
	110***1***11
	1***1***1
	1***1***11
	11***1***1
	11***1***11

Table 3.12: Wrong Slot Sequences considered from the simplified algorithm for an X-Y MPPM system with symbol in error being represented in italics (1 alone is called a standard error).

For the $\binom{12}{2}$ MPPM system, every WS sequence should be divided by 64 (valid MPPM codewords) and then by 6 (number of encoded PCM bits) to obtain the equivalent error rate per PCM bit of the sequence.

3.2 Testing

The testing strategy as previously stated was to test the software during implementation (every algorithm, functionality, of the software was tested separately)

using test patterns (presented in table 3.13 - appendix B). The test patterns [127] considered were in the following sections:

1) Interface / Control of Inputs
2) Data Inputs / Outputs
3) Result Handling

However, the software was tested thoroughly at the end of the implementation phase and account taken of any different results (deviations) between the two algorithms used for the MLSD scheme. Table 3.14 presents results for a $\binom{12}{2}$ MPPM system using both software algorithms alongside results taken without the use of software (complete analysis presented in tables 3.15, 3.16 and 3.17 - appendix B). From the results obtained it is clear that both algorithms give correct and similar results. In addition, the time taken for a complete non-automated analysis is now dramatically reduced using the computerized method.

Although the processing time between the two algorithms (complete and simplified) is almost the same for the systems of the $\binom{12}{Y}$ family, the difference becomes apparent when larger systems of the family are examined. For example, for the largest system $\binom{12}{6}$, 188 different sequences are considered in the complete algorithm, in contrast to 26 needed from the simplified.

ERASURE			
Sequences	**SOFTWARE ALGORITHMS (maximum response<1sec)**		**Non Automated Analysis (max response<15min)**
	Complete	**Simplified (2-Pulse)**	
P***e***	0.634	0.634	0.639
Pe1***1***	0.0593	0.0582	0.0593
Pe10***1***	0.0716	0.0716	0.0709
Pe110***1***	0.000366211	0.000366211	0.000366211
Pe***1***1	0.004923502604166	0.004923502604166	0.00501869
Pe1010***1***	0.0004069	0.0004069	0.0004069
FALSE ALARM			
Sequences	**SOFTWARE ALGORITHMS (maximum response<1sec)**		**Non Automated Analysis (max response<30min)**
	Complete	**Simplified (2-Pulse)**	
P***f***	0.205	0.205	0.2049
Pf111***0***	0.0000366	0.0000366	0.00003672
Pf11***0***	0.00293	0.00293	0.00293
Pf1***0***	0.0311	0.0303	0.0311
Pf1***0***1	0.00268	0.00268	0.00268
WRONG SLOT			
Sequences	**SOFTWARE ALGORITHMS (maximum response<1sec)**		**Non Automated Analysis (max response<45min)**
	Complete	**Simplified (2-Pulse)**	
P***s***	0.844	0.844	0.844
Ps1***1***	0.0479	0.0479	0.0479
Ps11***1***	0.00146	0.00146	0.00146
Ps***1***1	0.0996	0.0996	0.0996
Ps10***1***	0.0801	0.0801	0.0801
Ps110***1***	0.000732	0.000732	0.000732
Ps10***1***1	0.00146	0.00146	0.00146

Table 3.14: Sequence Error Rates in a 12-2 MPPM system using both software and non software analysis.

As there is little difference between the two analyses, the simplified version was used in the simulations. In table 3.18, time and space analysis is presented for the whole software solution (alongside some important characteristics) and the two main

algorithms. As it can be seen, the simplified algorithm is much faster and at the same time uses less memory. The data processing in the registers is done serially for memory usage reasons. If speed is the key issue, then the register processing can be turned to parallel easily. Therefore, adding a parallel process always results in halving the processing time.

Software Characteristics			**Complete Algorithm**	**2-Pulse Algorithm**
Lines of Code	7325		2262	870
No. of functionality points	20		20	
No. of algorithms	10		10	
No. of functions	10		10	
Space Analysis	Best Case	Worst Case	92 Integers 21 Floats 4 Chars 11 Reg 5 Files	92 Integers 21 Floats 4 Chars 11 Reg 5 Files
	92 Integers 21 Floats 4 Chars	11 Registers 5 Files		

Time Analysis	$((4+(2*Y))*T)+$ $((6*T)*X*PASC+(2*T))+$ $((PASC*(X*T+T))+T)+$ $(((X*T+T)*PASC)+(2*T))+$ $((4*T+(2*(X-Y)*T))+(4*T+2*Y*T))+$ $((2^{PCM\_BITS})*(X+2)*T)+$ $((3*PCM\_BITS)*(2^{PCM\_BITS})*T)+$ $((PASC2*(2*X) + 2 + (2*PASC2) +$ $(PCM\_BITS*PASC2))*T)+$ $((6+(2*PASCAL2))*(2^{PCM\_BITS})*T)+$ $64*T$	$(240+(32*Y+3))*4N$	$(23*T)+((3*Y+2)*4T)$

Table 3.18: Time and space analysis for the software solution and the two main algorithms (where N is the number of error sequences, and T is the clock period of the CPU).

Chapter 4

Mathematical Analysis

4.1 Receiver Configurations

In order to evaluate the error probabilities, the output voltage, $V_o(t)$, and the mean square receiver output noise $\langle n_o(t)^2 \rangle$ are required, and these, in turn, depend upon the received pulse shape, the type of preamplifier employed, the associated noise power spectral density, and the type of equalisation filter employed.

Let the received pulse shape, $h_p(t)$, have the following property (rectangular pulse) [95]:

$$\int_{-\infty}^{\infty} h_p(t)dt = 1 \tag{9}$$

The transmission medium in an optical link is the optical fibre. Plastic optical fibre (POF) is an optical fibre which is made out of plastic (in contrast to glass the more difficult to use fibre). POF is usually used in small scale networks because the high loss property limits the transmission distance to short value. Two basic forms of plastic optical fibre are currently available – step-index POF and graded-index POF (GI-POF). Of these, GI-POF offers the highest bandwidth and is used in this scheme. The impulse response of a GI-POF can be approximated to a Gaussian [13]-[14] and so

$$h_p(t) = \frac{e^{-\frac{t^2}{2\alpha^2}}}{\alpha\sqrt{2\pi}} \tag{10}$$

with a Fourier transform of [95]

$$H_p(\omega) = e^{-\frac{(\alpha\omega)^2}{2}} \tag{11}$$

Note that the received pulse variance is linked to the fibre bandwidth by [132]

$$\alpha = \frac{\sqrt{2\ln 2}T_b}{2\pi f_n} \tag{12}$$

where T_b is the PCM bit-time, and f_n is the fibre bandwidth normalised to the PCM data rate. In this research a dominant pole transimpedance preamplifier is assumed (stable operation), with transfer function, $Z_T(j\omega)$, given by [132]

$$Z_T(j\omega) = \frac{R_T}{1 + j\left(\frac{\omega}{\omega_p}\right)} \tag{13}$$

where R_T is the mid-band transimpedance and $\boldsymbol{\omega_p}$ is the –3 dB bandwidth of the preamplifier.

In order to make comparisons it is assumed that both the PCM and MPPM systems are operating at a data rate of 1 Gbit/s. For received pulse energy *b* the receiver output voltage is [132]

$$v_o(t) = \frac{bR}{2\pi}\int_{-\infty}^{\infty} H_p\left(j\omega\right) Z_T\left(j\omega\right) G\left(j\omega\right) e^{j\omega t} d\omega \qquad (14)$$

where $G(j\omega)$ is the transfer function of the equalising filter.

4.2 Matched Filter

In the case of the classical matched filter, the equaliser response has the form [132]

$$G(j\omega) = H_p\left(\omega\right)^* \qquad (15)$$

where $H_p(\omega)^*$ is the complex conjugate of the received pulse and represents the matched filter. By substituting (15) to (14) the output voltage is then given by [132]

$$v_o(t) = \frac{b\eta q R_T}{2\pi}\int_{-\infty}^{\infty} \frac{\left| e^{-\frac{(\alpha\omega)^2}{2}} \right|^2}{\left(1 + j\frac{\omega}{\omega_p}\right)} e^{j\omega t} d\omega$$
$$= b\eta q R_T \frac{\omega_p}{2} e^{(\alpha\omega_p)^2} e^{-\omega_p t} \operatorname{erfc}\left(\alpha\omega_p - \frac{t}{2\alpha}\right) \tag{16}$$

and the noise at the filter output is given by [132]

$$\left\langle n_o(t)^2 \right\rangle = R_T^2 S_0 \frac{\omega_c}{2} e^{(\alpha\omega_p)^2} \operatorname{erfc}\left(\alpha\omega_p\right) \tag{17}$$

4.3 Target System

As a (general) target system the $\binom{12}{Y}$ MPPM family is selected. It is selected because systems of the family are already used (and considered very efficient) from several authors ([85], [94] and [100]). As a family it can encode from 3 to 9 PCM bits and considers all the error sequences presented above (for other systems a larger or smaller number of error sequences can be considered). The specifications of the target system are: a 1.2 GHz bandwidth PINBJT receiver is assumed (Philips TZA 3043) with a low frequency input equivalent noise current spectral density of 16×10^{-24} A^2/Hz (double-sided). An operating wavelength of 650 nm [94], and an ideal photodiode quantum efficiency of 100% were taken (a typical efficiency is around 70%).

In the mathematical analysis a classical matched filter was used in combination with slope detection and maximum likelihood detection (MF-MLD). The final error probability is equal to the sum of all error probabilities (erasure, false alarm and wrong slot). The bit rate is set to 10^9 in bits per second (bps) and the PCM bit time to 10^{-9} seconds. The slot time is given by the multiplication of the PCM bit time and the number of PCM bits (*PCM_BITS*) divided by the number of slots (*X*). Therefore for the $\binom{12}{2}$ MPPM system the slot time is equal to $5*10^{-10}$ (seconds). The number of like symbols in PCM is set to 10. The quantum energy is given by $1.6 * 10^{-19}$ (atoms) and the wavelength of operation λ is equal to $1.55 *10^{-6}$ (meters).

For a given set of parameters (figure 4.1), the pulse shapes, derivates (slopes) and the noise were found and the number of photons per bit, *b*, calculated. A threshold parameter, *v*, was defined as

$$v = \frac{V_d}{V_{pk}} \tag{18}$$

where the symbols have the following meanings:

V_{pk} the peak receiver output,

V_d the receiver output at the threshold crossing time t_d.

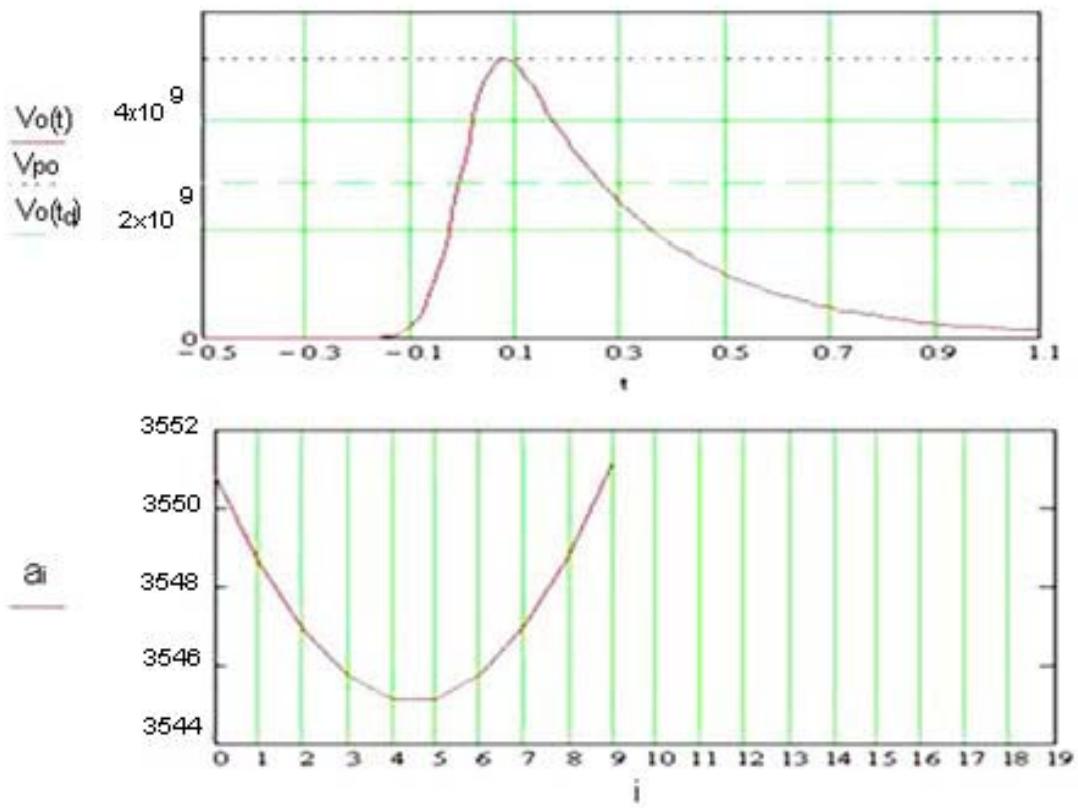


Figure 4.1: A received pulse for a normalized bandwidth of 10, where V_{po} is the peak receiver output, V_{pk} , $V_o(t_d)$ is the receiver output, V_d , at the threshold crossing time td. The threshold parameter, v, is set to 0.5, and a_i is the minimum amount of photons (in this case 3545) needed for a pulse detection for the specific MPPM system.

The threshold parameter, *v*, is nominally set to 0.5 (threshold voltage of half the peak amplitude). The pulse should be located in the centre of the slot (to minimise WS and ER errors) and so the boundaries of the slot will be $(-\frac{Ts}{2}, \frac{Ts}{2})$. Note also that, for consecutive pulses, t_d is defined as

$$t_{d+N} = (N\text{-}1)*T_s \qquad (19)$$

where N is the number of pulses ($N \geq 1$). For low normalised bandwidth values (f_n) the threshold parameter (v) and the decision time (t_d) are changed. It is clear that by changing the threshold parameter some errors are minimized. Thus, Erasure errors can be minimized by lowering the threshold point of detection. Alternatively, False Alarm errors can be minimized by increasing the threshold point of detection.

All the experiments (implemented using MATHCAD© Professional) considered transmission through plastic optical fibre with a Gaussian impulse response [13]-[14]. The channel bandwidth was normalised to the PCM bit-rate and varied between 100 and 1.2. Operation below 1.2 was impossible (a_i is huge) due to the high levels of ISI and IFI causing sequences such as 110, 1101 and 1110 to be received as 10, 101 and 10 respectively as shown in figure 4.2.

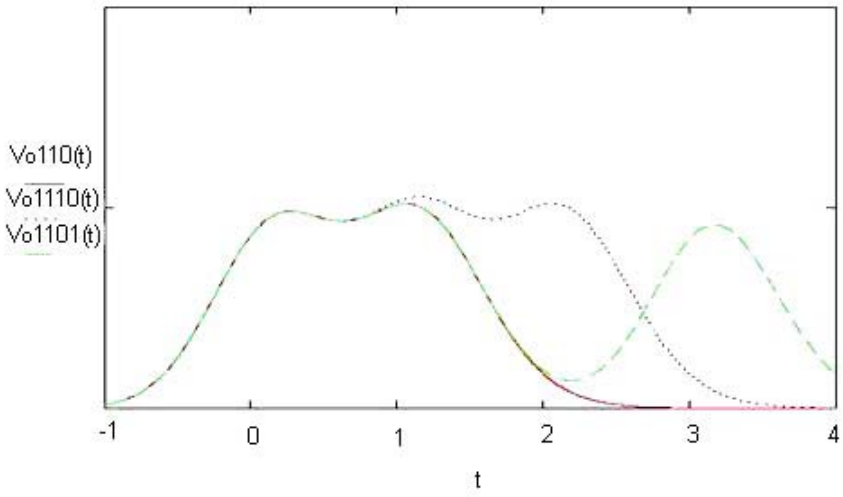


Figure 4.2: 1110, 110 and 1101 multiple PPM sequences with f_n=1.

As can be seen, there is significant ISI between adjacent slots and so, in order to detect the slopes correctly (no clear peaks of adjacent pulses), the average power must be increased, especially at low bandwidth. The level of dispersion is dependent upon the type of equalisation filter employed, and, although matched filtering is optimal at high fibre bandwidths where ISI is not a problem, it gives poor performance at the low fibre bandwidths due to ISI. The simplified sensitivity analysis for a $\binom{12}{2}$ multiple PPM system is presented in appendix D.

4.4 Sensitivity Results and Coding Quality

Table 4.1 details the variation in the number of photons per multiple PPM pulse, for a PCM bit error rate of 1 in 10^9, as the channel normalised bandwidth, f_n, varies from 100 to 1.2. All the systems of the 12-*Y* MPPM family are presented at a particular bandwidth, and each set compares results from the complete and simplified analyses. As can be seen, there is little difference between the two analyses and so the simplified version was used in the simulations. It is apparent from the table that the optimum coding level ranges from 12-2 for high bandwidths, to 12-1 for low bandwidths. It is also apparent that, as the number of pulses increases sensitivity is degraded (more errors are generated). Another observation is that the difference between best and worst sensitivity is not very large until very low normalized bandwidth (f_n) values.

f_n	Best sensitivity (photons per pulse)		Optimum coding level	Worst sensitivity (photons per pulse)		Worst coding level
	Simplified	Complete		Simplified	Complete	
100	2763	2763	12-2	2845	2853	12-10
80	2782	2782	12-2	2868	2871	12-10
60	2821	2821	12-2	2933	2935	12-10
40	2903	2903	12-2	2989	2993	12-10
20	3103	3103	12-2	3204	3205	12-10
10	3501	3501	12-2	3621	3622	12-10
5	4257	4257	12-2	4411	4413	12-10
4	4642	4642	12-2	4810	4812	12-10
3	5312	5312	12-2	5500	5504	12-10
2	6994	6994	12-2	7232	7237	12-10
1.8	7744	7744	12-2	8006	8010	12-10
1.5	10030	10030	12-1	11910	11910	12-10
1.2	42380	42380	12-1	101000	101000	12-10

Table 4.1: Best and worst case sensitivity (using the simplified and complete models) and corresponding coding level for varying normalised channel bandwidth.

Bit Error Rate (BER) is also a factor of efficiency in an MPPM system. Sugiyama and Nosu [85] developed a methodology to predict the BER of an MPPM link when erasure errors occurred. The author also suggests a methodology that can be expanded to False Alarm and Wrong Slot errors to predict the total BER of an MPPM link when all types of error are considered. The methodology is based on the number of known and unknown (MPPM) bits when an error occurs. Hence, if the number of bits restored from one signal is L, then L_k (fixed number) of these bits are always known, and the remainder L_u, remain unknown because of an optical pulse error. Considering that a pulse and an empty slot are equal probable, the error probability of an erasure error is given by (20)

$$P_a = 0.5\ (Lu/L) \tag{20}$$

When an erasure error occurs in an $\binom{N}{K}$ MPPM link, the number of known bits is $L_k = K\text{-}1$. Hence, the number of unknown bits L_u can trigger $M_p = N\text{-}K\text{+}1$ MPPM sequences. The L_u bits can trigger $M_b = 2^{Lu}$ PCM sequences. Because, $M_p = M_b$,

$$2^{Lu} = N\text{-}K\text{+}1 \tag{21}$$

Thus,

$$L_u = \log_2(N\text{-}K\text{+}1) \tag{22}$$

By substituting (22) to (20)

$$P_{er} = 0.5\ (\log_2(N - K + 1)/L) \tag{23}$$

L bits can trigger 2^L sequences or else $\binom{N}{K}$. Therefore,

$$2^L = \binom{N}{K} \tag{24}$$

and accordingly

$$L = \log_2 \binom{N}{K} \tag{25}$$

As a result, (23) can be written as

$$P_{er} = 0.5 \left(\log_2 (\mathrm{N} - \mathrm{K} + 1) \Big/ \log_2 \binom{N}{K} \right) \tag{26}$$

For a False Alarm error in an $\binom{N}{K}$ MPPM link, K+1, pulses are detected. Consequently,

$$P_{fa} = 0.5 \left(\log_2 (\mathrm{K} + 1) \Big/ \log_2 \binom{N}{K} \right) \tag{27}$$

For a Wrong Slot (left dispersion) error of an MPPM link the treatment is the same as a false alarm error (an extra pulse is generated). Hence,

$$P_{ws\_l} = 0.5 \left(\log_2 (\mathrm{N} - \mathrm{K} + 1) \Big/ \log_2 \binom{N}{K} \right) \tag{28}$$

For a Wrong Slot (right dispersion) error of an MPPM link $L = K$ and $L_u = \log_2 K$. Thus,

$$P_{ws\_r} = 0.5 \left(\log_2(\mathrm{K}) \Big/ \log_2 \binom{N}{K} \right) \tag{29}$$

The total BER is equal to the sum of all error probabilities (30) weighted to the occurrence of each error type. As an example, consider the erasure, *Per*, for the $\binom{12}{2}$ MPPM system. This error probability should be weighted with 128 (2 pulses/MPPM frame * 64 MPPM frames). For the False Alarm error the error probability should be weighted with 640 (10 slots/frame * 64 frames). Finally, for the wrong slot error (both left and right dispersion) the error probability should be weighted with 256 (2 pulses/frame * 2 errors-left and right dispersion- * 64 frames).

$$\boldsymbol{BER} = \boldsymbol{P}_{er\_weighted} + \boldsymbol{P}_{fa\_weighted} + \boldsymbol{P}_{ws\_l\_weighted} + \boldsymbol{P}_{ws\_r\_weighted} \tag{30}$$

The BER results in a $\binom{12}{Y}$ MPPM system are presented in table 4.2. From the results obtained it is clear that the most efficient systems are in the middle of the family.

Systems	P_{er}	P_{fa}	P_{ws_l}	P_{ws_r}	WEIGHTING			Total BER
					ER	FA	WS	
12_1	0.5	0.139	0.139	0	8	88	2	0.134
12_2	0.286	0.131	0.131	0.083	128	640	4	0.056
12_3	0.213	0.129	0.129	0.102	384	1152	6	0.039
12_4	0.177	0.13	0.13	0.112	1024	2048	8	0.03
12_5	0.156	0.134	0.134	0.121	2560	3584	10	0.026
12_6	0.142	0.142	0.142	0.131	3072	3072	12	0.023
12_7	0.134	0.156	0.156	0.146	3584	2560	14	0.022
12_8	0.13	0.177	0.177	0.168	2048	1024	16	0.022
12_9	0.129	0.213	0.213	0.204	1152	384	18	0.024
12_10	0.131	0.286	0.286	0.275	640	128	20	0.03
12_11	0.139	0.5	0.5	0.482	88	8	22	0.109

Table 4.2: BER calculation in a 12-Y MPPM system.

These representations of the data do not take into account the coding efficiency of the systems – a $\binom{12}{1}$ system can code up to 3 PCM bits while a $\binom{12}{2}$ system can code up to 6 PCM bits. Thus the $\binom{12}{2}$ system could be regarded as more bandwidth efficient. In order to explore this, a bandwidth expansion, *BE*, parameter is defined

$$BE=\frac{X}{n} \tag{31}$$

where *X* is the number of multiple PPM slots in a frame and *n* is the number of *PCM BITS* encoded.

A process of normalisation is now applied to the sensitivities and bandwidth expansion so that all values are expressed between 0 and 1. To achieve this, the number of photons required for each coding level are divided by the largest number of photons per pulse for that particular bandwidth (worst case sensitivity of the 12-10 system - table 4.1), and all bandwidth expansions are divided by the largest BE (12/3 = 4 for all f_n). These normalized results are shown in table 4.3.

f_n	Normalised sensitivity	Normalised BE	Coding level
100	0.971	0.5	12-2
80	0.970	0.5	12-2
60	0.962	0.5	12-2
40	0.971	0.5	12-2
20	0.969	0.5	12-2
10	0.967	0.5	12-2
5	0.965	0.5	12-2
4	0.965	0.5	12-2
3	0.966	0.5	12-2
2	0.967	0.5	12-2
1.8	0.967	0.5	12-2
1.5	0.842	1	12-1
1.2	0.420	1	12-1

Table 4.3: Normalized sensitivity, BE and optimum coding level of a 12-Y MPPM system.

A weighted sum approach (used when performing a sum, integral, or average in order to give some elements more of a "weight" than others) can now be applied [110] to the multiple PPM systems (at every value of f_n) in terms of sensitivity and bandwidth expansion, with a range of weights between 0 and 100% in steps of 10%. A weighting of 0% means bandwidth expansion is the most important parameter (and hence power can be spared) while a weighting of 100% means sensitivity is more important (and hence

bandwidth can be spared). A 50% weighting is where the two parameters are equally important and is a key point of interest. Thus, using (32), an efficiency factor, η, can be defined as

$$\eta = 1 - \left(\mathrm{ph}_{\mathrm{PCM\ norm}} \times \mathrm{weight}_{\mathrm{sens}} + \mathrm{BE}_{\mathrm{norm}} \times \mathrm{weight}_{\mathrm{BE}}\right) \qquad (32)$$

where,

$\mathrm{ph}_{\mathrm{PCM\ norm}}$ is the normalized sensitivity in photons per PCM bit

$\mathrm{BE}_{\mathrm{norm}}$ is the normalized bandwidth expansion

$\mathrm{weight}_{\mathrm{sens}}$ is the sensitivity weighting = 0 to 1 in steps of 0.1, and

$\mathrm{weight}_{\mathrm{BE}}$ is the bandwidth weighting = 1- $\mathrm{weight}_{\mathrm{sens}}$

A system is considered to be 100% efficient if it has the best sensitivity and the lowest BE (coding 9 PCM bits). On the other hand, a system is 0% efficient if it has the lowest sensitivity and the bandwidth expansion of the $\binom{12}{11}$ system (only 3 PCM bits can be encoded). Table 4.4 shows the efficiency map for a 12-*Y* multiple PPM system across the range of f_n.

	BE is more important					Equal Weight	Sensitivity is more important				
f_n	0-100	10-90	20-80	30-70	40-60	50-50	60-40	70-30	80-20	90-10	100-0
100	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-2
80	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-2	12-2
60	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-2	12-2
40	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-2	12-2
20	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-2	12-2
10	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-2	12-2
5	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-2	12-2
4	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-2	12-2
3	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-2	12-2
2	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-2	12-2
1.8	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-2	12-2
1.5	12-5	12-5	12-5	12-5	12-5	12-5	12-5	12-2	12-2	12-2	12-1
1.2	12-5	12-5	12-5	12-5	12-5	12-5	12-1	12-1	12-1	12-1	12-1

Table 4.4: Efficiency map (as defined by equation 11) for a 12-Y multiple PPM system.

As can be seen, the $\binom{12}{6}$ multiple PPM system is generally the most efficient system. It is only when the bandwidth is extremely low that the coding level changes to $\binom{12}{5}$. Both systems code 9 bits of PCM data but the normalised bandwidth expansion for the $\binom{12}{5}$ code is slightly lower. When the sensitivity weighting is greater than 90%, the optimum coding level is generally $\binom{12}{2}$. In this situation, bandwidth expansion is not important and so the higher sensitivity, lower codes can be used. The simulation predicts that $\binom{12}{1}$ (i.e. digital PPM) should be used at very low bandwidths. This is because there is a very high level of pulse dispersion leading to an increased level of ISI and IFI and, as

digital PPM only has one pulse in a frame, it will not be affected in the same way as a two pulse frame.

Figure 4.3 shows a surface plot for a 12-Y multiple PPM system operating with a normalized bandwidth of 30. In this figure, the efficiency factor, η, has been plotted on the vertical axis to demonstrate the optimum. This figure clearly shows that the $\binom{12}{6}$ coding system is the most efficient except at the extremes, when sensitivity is more important than operating speed (above 80-20 weighting). In this instance smaller systems are more efficient, such as $\binom{12}{2}$.

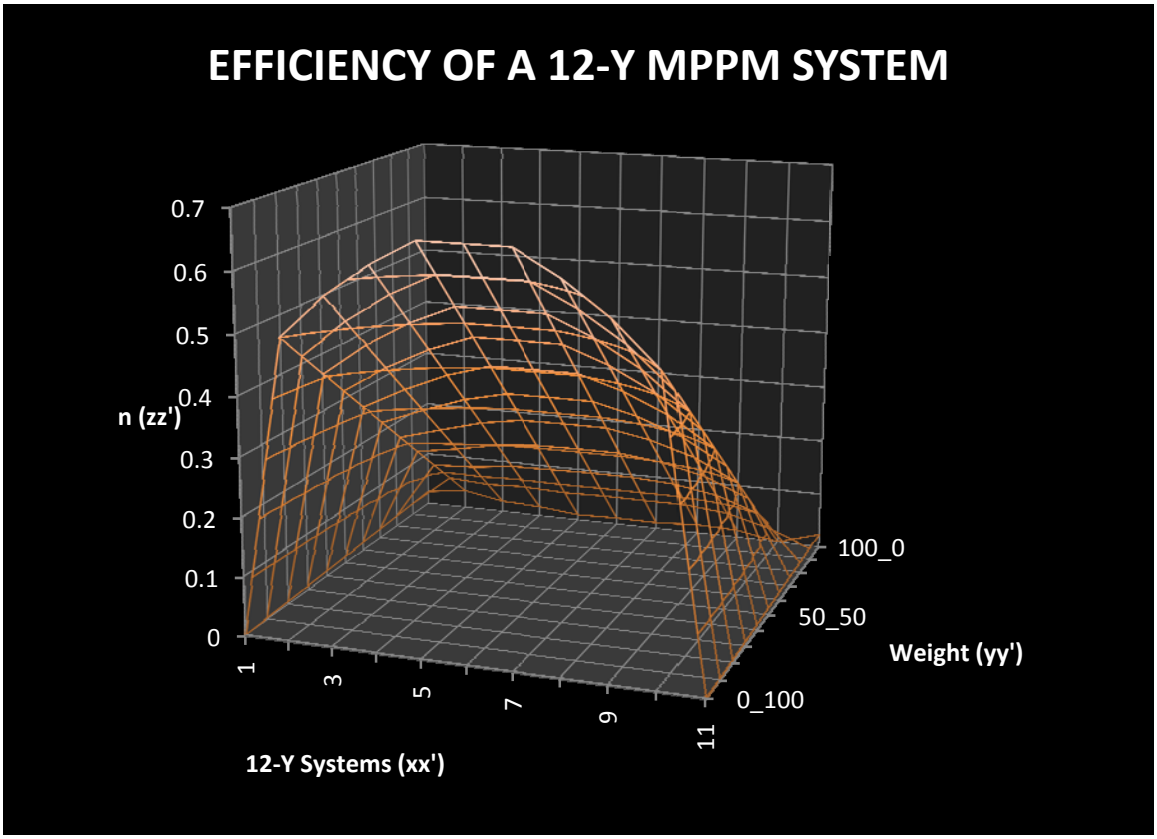


Figure 4.3: Efficiency (surface weighted sum) plot of a 12-Y multiple PPM system for a normalized bandwidth of 30 (in the yy' axis from 0 to 49, bandwidth expansion is dominant from 51 to 100 sensitivity is dominant with equality in 50).

Chapter 5

Theoretical Investigation into the Effects of Data Mapping

Table 5.1 details the variation in the number of photons per PCM bit as the normalised channel bandwidth, f_n, varies from 100 to 1.2 for $\binom{12}{Y}$ systems using Linear Increment (LI) coding.

Linear Increment mapping											
Sensitivity (photons/PCM bit)											
Fn	**12_1**	**12_2**	**12_3**	**12_4**	**12_5**	**12_6**	**12_7**	**12_8**	**12_9**	**12_10**	**12_11**
100	930	921	1199	1401	1567	1877	2190	2802	3610	4740	10234
80	938	927	1207	1411	1579	1891	2207	2823	3637	4777	10314
60	952	940	1224	1431	1601	1917	2238	2863	3689	4843	10457
40	979	968	1260	1473	1648	1973	2303	2947	3797	4985	10762
20	1047	1034	1348	1576	1764	2113	2466	3155	4065	5340	11528
10	1180	1167	1522	1780	1993	2387	2786	3565	4594	6035	13020
5	1435	1419	1854	2168	2429	2909	3395	4342	5595	7352	15833
4	1565	1547	2023	2365	2650	3173	3703	4735	6101	8017	17237
3	1790	1771	2315	2706	3032	3629	4237	5415	6975	9167	19657
2	2356	2331	3047	3561	3993	4775	5575	7119	9162	12053	25663
1.8	2608	2581	3373	3941	4421	5285	6173	7880	10140	13343	28347
1.5	3347	3533	4766	5665	6433	7793	9170	11650	14953	19850	37547
1.2	14127	27390	39450	47555	54461	66327	78167	99250	126900	168333	316653

Table 5.1: Sensitivity in photons/PCM bit of a 12-Y MPPM system using a Linear Increment mapping for a target (PCM) error probability of Pe = 10^{-9}.

Tests [111] were carried out to examine the effects of data mapping for four different mappings – Linear Increment, Linear Decrement, Gray Code and Random – that operate without redundancy for every version of the $\binom{12}{Y}$ family. The results are shown in table 5.2, which details the improvement in sensitivity (photons/PCM bit) obtained over Linear Increment mapping (taken as the reference).

Maximum improvement in sensitivity (photons/PCM bit)											
f_n	12_1	12_2	12_3	12_4	12_5	12_6	12_7	12_8	12_9	12_10	12_11
100	-2	0	-2	-2	-1	-4	0	0	-3	-18	-18
80	-2	0	-2	-2	-1	-5	0	0	-3	-18	-22
60	-2	0	-1	-2	-1	-5	0	0	-3	-18	-22
40	-2	0	-2	-2	-1	-5	0	0	-3	-18	-22
20	-3	0	-2	-2	-1	-5	0	0	-4	-20	-22
10	-4	0	-2	-2	-1	-6	0	0	-4	-23	-26
5	-7	0	-2	-2	0	-7	0	0	-4	-30	-48
4	-9	0	-3	-3	-2	-8	0	0	-5	-33	-55
3	-10	0	-3	-3	-2	-9	0	0	-5	-38	-88
2	-12	0	-4	-3	-3	-12	0	0	-4	-55	-187
1.8	-13	0	-5	-3	-3	-13	0	0	-5	-62	-220
1.5	0	-30	-26	-95	0	-20	-8	-10	-26	-133	0
1.2	0	-793	-321	-1630	-6	-173	-156	-80	-450	-1500	0

Table 5.2: Maximum improvement of sensitivity in photons/PCM bit compared to Linear Increment mapping (improvement implies reduction in the number of photons/PCM bit). A zero number means that Linear Increment remains the most efficient mapping.

Table 5.3 presents an efficiency map as the channel normalised bandwidth, f_n, varies.

12-Y multiple PPM EFFICIENCY MAP											
f_n	12_1	12_2	12_3	12_4	12_5	12_6	12_7	12_8	12_9	12_10	12_11
100	GC	LI	GC	GC	GC	GC	LI	LI	GC	RD	GC
80	GC	LI	GC	GC	GC	GC	LI	LI	GC	RD	GC
60	GC	LI	GC	GC	GC	GC	LI	LI	GC	RD	GC
40	GC	LI	GC	GC	GC	GC	LI	LI	GC	RD	GC
20	GC	LI	GC	GC	GC	GC	LI	LI	GC	RD	GC
10	GC	LI	GC	GC	GC	GC	LI	LI	GC	RD	GC
5	GC	LI	GC	GC	LI	GC	LI	LI	GC	RD	GC
4	GC	LI	GC	GC	GC	GC	LI	LI	GC	RD	GC
3	GC	LI	GC	GC	GC	GC	LI	LI	GC	RD	GC
2	GC	LI	GC	GC	GC	GC	LI	LI	GC	RD	GC
1.8	GC	LI	GC	GC	GC	GC	LI	LI	GC	RD	GC
1.5	LI	GC	GC	RD	LI	GC	GC	RD	RD	RD	LI
1.2	LI	GC	GC	RD	RD	GC	GC	RD	RD	RD	LI

Table 5.3: Variation in optimum mapping as the channel normalised bandwidth, f_n, varies from 100 to 1.2 for a 12-Y MPPM system. The mappings considered are Linear Increment, LI, Linear Decrement, LD, Gray Code, GC, and Random, RD.

From these results it is evident that eight systems (from $\binom{12}{1}$ to $\binom{12}{6}$ and $\binom{12}{9}$ and $\binom{12}{11}$) enhance their sensitivity if Gray Codes are used (although the percentage changes are small). Two systems ($\binom{12}{7}$ and $\binom{12}{8}$) enhance their sensitivity if Linear Increment/Decrement is used (the LI and LD mappings gave almost the same sensitivity) and Random mapping was most efficient only for $\binom{12}{10}$. Thus it can be seen that Gray Codes are the most efficient mapping for most systems of the $\binom{12}{Y}$ family. This can be

explained from the fact that Gray Codes minimize the Hamming distance between adjacent multiple PPM words, and MLSD is used. Hence, if the Hamming distance is kept to minimum, there are minimum errors between the decoded word and the original data (as shown in Chapter 3). As an example, consider the codeword [2,4] of the $\binom{12}{2}$ MPPM system, which decodes to the 001100 equivalent PCM word (if LI mapping is used). If a WS error occurs on the first or second pulse of the codeword, 7 error bits are generated. This means an average error/PCM bit for [2,4] of 0.29. With Gray coding, [2,4] decodes as 001010 and the number of errors is reduced to 6. The average error/PCM bit is reduced to 0.25. Other results were generated for the rest of the sequences and the other error types in a similar fashion.

Chapter 6

Optimum Mapping in an Optical Multiple PPM link using a Maximum Likelihood Sequence Detection Scheme

---

The previous chapter has shown how coding can alter the performance of an MPPM link. It has also been shown that the performance of an MPPM link can be easily predicted by identifying certain sequences. In this chapter, this work [112] is extended so that the original methodology described here identifies dominant errors which are then used in another algorithm that determines the optimum coding technique for a particular multiple PPM system. The reduction in processing time that comes from this methodology is significant.

6.1 Dominant error sequences in multiple PPM

Figure 6.1 (appendix A) is a data flow diagram showing how the dominant error sequences are identified. Each error sequence is isolated in turn by setting its equivalent PCM error rate to one while the other sequences are set to zero (or very close to zero). Thus only one error source contributes to the system performance at any one time. The software previously described is then used to find the resulting photons/bit for the desired error rate. This is then multiplied by the probability of the particular error sequence

occurring to give the "sequence sensitivity". By considering each error sequence in turn and summing the sequence sensitivities, the total number of photons per bit can be found and the sensitivity of the system can be easily calculated. Comparisons (table 6.1) with the full mathematical model show that this method is accurate.

f_n	12-2 MPPM System Sensitivity (in photons/PCM bit)	
	MATHEMATICAL ANALYSIS	ONLY S/W
100	921	870
80	927	875
60	940	887
40	968	912
20	1034	974
10	1167	1098
5	1419	1331
4	1547	1448
3	1771	1649
2	2331	1767
1.8	2581	2567
1.5	3533	3529
1.2	27390	27389

Table 6.1: Sensitivity results (in photons/PCM bit) for a 12-2 MPPM system using the full mathematical model and the s/w model.

Tests on other $\binom{12}{Y}$ systems (table 6.2) have shown that the dominant error sources and sequence sensitivities are the same in these other systems and so this methodology can be applied to multiple PPM systems in general.

SYSTEM	SENSITIVITY ANALYSIS (in photons/PCM bit)	
	MATHEMATICAL ANALYSIS	ONLY S/W
12-1	930	903
12-2	921	870
12-3	1199	1141
12-4	1401	1342
12-5	1567	1509
12-6	1877	1828
12-7	2190	2146
12-8	2802	2743
12-9	3610	3560
12-10	4740	4678
12-11	10234	10168

Table 6.2: Sensitivity results (in photons/PCM bit) for a 12-Y MPPM system using the full mathematical model and the s/w model for a normalized bandwidth of 100.

Even when other mappings are used (as in table 5.3) with the software model, the same efficiency map is generated as shown in table 6.3 (except from some exceptions represented in bold-italics).

12-Y multiple PPM EFFICIENCY MAP											
f_n	12_1	12_2	12_3	12_4	12_5	12_6	12_7	12_8	12_9	12_10	12_11
100	GC	LI	GC	GC	GC	GC	LI	LI	GC	RD	GC
80	GC	LI	GC	GC	GC	GC	LI	LI	GC	RD	GC
60	GC	LI	GC	GC	GC	GC	LI	LI	GC	RD	GC
40	GC	LI	GC	GC	GC	GC	LI	LI	GC	RD	GC
20	GC	LI	GC	GC	GC	GC	LI	LI	GC	RD	GC
10	GC	LI	GC	GC	GC	GC	LI	LI	GC	RD	GC
5	GC	LI	GC	GC	***GC***	GC	LI	LI	GC	RD	GC
4	GC	LI	GC	GC	GC	GC	LI	LI	GC	RD	GC
3	GC	LI	GC	GC	GC	GC	LI	LI	GC	RD	GC
2	GC	LI	GC	GC	GC	GC	LI	LI	GC	RD	GC
1.8	GC	LI	GC	GC	GC	GC	LI	LI	GC	RD	GC
1.5	LI	GC	***RD***	***LI***	***LI***	GC	GC	***LI***	RD	RD	***GC***
1.2	LI	GC	GC	***LI***	***LI***	GC	GC	***RD***	***LI***	RD	***GC***

Table 6.3: Variation in optimum mapping (using the s/w model) as the channel normalised bandwidth, f_n, varies from 100 to 1.2 for a 12-Y MPPM system. The mappings considered are Linear Increment, LI, Linear Decrement, LD, Gray Code, GC, and Random, RD.

This method of identifying the dominant error sequences and determining the subsequent sequence sensitivity means that the simulation time is reduced considerably as shown in table 6.4. Thus the effects of different mappings can be determined very rapidly so making analysis much faster.

SYSTEM	SENSITIVITY ANALYSIS SCHEMES	
	NO S/W AND MATHEMATICAL ANALYSIS	ONLY S/W
12-1	<30 min	<1 sec
12-2	<110 min	<1 sec
12-3	<140 min	<1 sec
12-4	<200 min	<1 sec
12-5	<380 min	<3 sec
12-6	<380 min	<3 sec
12-7	<380 min	<3 sec
12-8	<200 min	<1 sec
12-9	<140 min	<1 sec
12-10	<110 min	<1 sec
12-11	<30 min	<1 sec
TOTAL	<2100 min	<18 sec

Table 6.4: Time needed to predict the efficiency of the 12-Y system using non-automated methods and only software.

The most significant error sequences were found to be those involving an erasure with the next being those with a false alarm. The wrong slot sequences are only significant when there is considerable pulse dispersion caused by a normalised fibre

bandwidth, f_n, of less than 2. As erasure sequences are the most dominant sequences, the next section describes how to find an optimum, or close to optimum, mapping that minimises their effect.

6.2 Optimum mapping in multiple PPM

To find the optimum mapping until now two approaches were developed. The first approach was to automatically generate (through software) all available mappings of a MPPM system. An algorithm that calculates all the available permutations between a set of numbers (PCM codewords) was required. Various algorithms were developed to generate the maximum number of permutations as shown below:

1) A recursive permutation algorithm by Alexander Bogomolny (www.cut-the-knot.org/do_you_know/AllPerm.shtml)
2) A permutation algorithm based on a string permutation algorithm from the University of Exeter (newton.ex.ac.uk/teaching/resources/jmr/recursion.html)

A new algorithm was developed based on Martin Gardner's Scientific American article [133]. This algorithm gives the fastest known method of listing all permutations. The algorithm is presented in figure 6.2 (appendix A). The software solution is presented in appendix C. However, permutations are geometrically increased. Thus, for a $\binom{12}{2}$

multiple PPM system, 6 PCM bits can be encoded. That means 720 (6!) different mappings. This number becomes larger as the size (X, Y) of the MPPM system increases.

Therefore a new approach had to be developed based on prediction. In a $\binom{12}{2}$ multiple PPM system, 6 PCM bits can be encoded. In order to minimise the error probability, the Hamming Distance (HD) between all equivalent PCM code-words, referred to as the Total Hamming Distance (THD), should be minimised. Taking the all zero code-word as reference, the PCM codewords have the following properties:

i) 6 codewords with HD of 1
ii) 15 codewords with HD of 2
iii) 20 codewords with HD of 3
iv) 15 codewords with HD of 4
v) 6 codewords with HD of 5
vi) 1 codeword with HD of 6

The mapping of PCM codewords onto the multiple PPM codewords must be chosen to minimise the THD. The three sources of error need to be considered in turn, and their effects on the THD minimised by reducing their individual HD.

Consider the $\binom{12}{2}$ detected sequence [1,x] where $12 \geq x \geq 2$. Erasure of any of the second pulses will cause the MLSD detector to receive only the pulse in slot 1 so giving the [1,?] condition. The received codeword could have been any of the 11 codewords in the [1,x] sequence and so the MLSD will average the equivalent PCM data of these codewords to give a PCM word of 000000. Taking this as the reference, table 6.5, shows

that the Erasure Sequence Hamming Distance, ESHD, is equal to 17 for both linear increment and Gray coded data.

Erasure Error [1,?]	Mapping		
	Linear Increment	Gray Codes	Optimum
[1,2]	000000	000000	000000
[1,3]	000001	000001	000001
[1,4]	000010	000011	000010
[1,5]	000011	000010	000100
[1,6]	000100	000110	001000
[1,7]	000101	000111	010000
[1,8]	000110	000101	100000
[1,9]	000111	000100	000011
[1,10]	001000	001100	000101
[1,11]	001001	001101	001001
[1,12]	001010	001111	010001
Averaged [1,?] codeword	000000	000101	000000
Total Hamming Distance	225	225	224
Averaged Hamming Distance	17	17	14

Table 6.5: Total (THD) and Averaged (AHD) Hamming Distance (Number of PCM Errors) of Linear Increment, Gray Codes and "Optimum" mapping of the [1,?] ER averaged codeword.

To obtain the optimum coding regime for this [1,?] condition, PCM codewords should be chosen and assigned to the multiple PPM codewords so that the ESHD, and hence the THD, are minimised. If [1,2] is assigned to 000000, there are 6 other codewords with HD of 1 and a possible 4 out of 15 other codewords with HD of 2. Table 6.5 shows the resulting "optimum" mapping for this particular sequence. The ESHD is now reduced to 14 which is an 18% reduction compared to linear mapping. The erasure sequences [1,?] to [6,?] can also have an ESHD of 14; however the ESHD increases for codes above [6,?], as shown in table 6.6.

Erasure sequence	**Averaged codeword**	**Number of codewords**	**ESHD**
[1,?]	000000	11	14
[2,?]	001011	11	14
[3,?]	010101	11	14
[4,?]	01111 0	11	14
[5,?]	100110	11	14
[6,?]	10110 1	11	14
[7,?]	110011	11	16
[8,?]	11100 0	11	20
[9,?]	110010	11	22
[10,?]	000000	10	21
[11,?]	101001	10	24
[12,?]	011101	9	22

Table 6.6: Erasure averaged codewords for a 12-Y multiple PPM system.

This is because the choice of remaining PCM code-words is limited due to the fact that the lowest HD ones have already been allocated. Thus the MLSD must average the remaining PCM code-words, which have a higher HD, and this results in a large HD for the erasure codewords [7,?] to [12,?]. A complication occurs with the erasure codewords [10,?] and [11,?]. Below [10,?] the number of code-words to be averaged is 11 (table 6.6) and so the MLSD has an odd number of 1s or 0s to consider when making a decision and there is no ambiguity in the result. However, when [10,?] and [11,?] are received, the MLSD will randomly allocate a 1 or a 0 to a data bit because the number of 1s and 0s is the same. This random allocation will generate a large number of errors, and so the optimum mapping should avoid this.

Minimisation of the ESHD means that the errors from some false alarm and wrong-slot sequences will also be reduced. Consider the sequence [1,2]. A false alarm will generate a sequence [1,2,x], where $12 \geq x > 2$. The output of the MLSD will be the average of three PCM codewords corresponding to [1,2], [1,x] and [2,x]. If $8 \geq x > 2$, the code-words [1,2] and [1,x] will have a HD of 1 and so only the [2,x] code-word will have to be chosen (preferably with a HD of 2). If $12 \geq x > 8$, only the code-word [1,2] has a HD of 1 while the other two code-words [1,x] and [2,x] have a HD of 2. In general, for [1,x,y] code-words, where $11 \geq x > 1$, $12 \geq y > 2$ and $x > y$, two out of three codewords have a maximum HD of 2. The missing codewords are allocated to minimise the ESHD. As regards wrong-slot errors, some are already optimised because they appear to be pulse erasures (right shift wrong-slot to adjacent pulse for instance). For the others, Gray codes

are very efficient because a wrong-slot error causes the next codeword to be chosen and, in Gray coded data, the resulting error will be one bit. However, wrong-slot errors are only significant over a small range of low bandwidths and so minimising their effect is not as important as minimising the effect of erasures.

A mapping such as this that targets erasure errors and affects a large number of false alarm and wrong-slot sequences is optimum, or very close to optimum, and table 6.7 shows the mapping generated.

Optimum mapping for the 12-2 MULTIPLE PPM System							
c/w	**PCM**	**c/w**	**PCM**	**c/w**	**PCM**	**c/w**	**PCM**
[1,2]	000001	**[2,8]**	011011	**[4,7]**	111110	**[6,10]**	101100
[1,3]	000100	**[2,9]**	001010	**[4,8]**	011010	**[6,11]**	101101
[1,4]	011000	**[2,10]**	000011	**[4,9]**	011110	**[6,12]**	001101
[1,5]	000110	**[2,11]**	001001	**[4,10]**	010110	**[7,8]**	110001
[1,6]	001100	**[2,12]**	001011	**[4,11]**	011111	**[7,9]**	110010
[1,7]	110000	**[3,4]**	010100	**[4,12]**	011100	**[7,10]**	010011
[1,8]	100000	**[3,5]**	000111	**[5,6]**	101111	**[7,11]**	110011
[1,9]	010000	**[3,6]**	111101	**[5,7]**	100010	**[7,12]**	110111
[1,10]	000000	**[3,7]**	110101	**[5,8]**	101110	**[8,9]**	111010
[1,11]	001000	**[3,8]**	010001	**[5,9]**	110110	**[8,10]**	101000
[1,12]	000010	**[3,9]**	010111	**[5,10]**	100100	**[8,11]**	111000
[2,3]	011001	**[3,10]**	000101	**[5,11]**	100110	**[8,12]**	111001
[2,4]	001111	**[3,11]**	010101	**[5,12]**	100111	**[9,10]**	010010
[2,5]	100011	**[3,12]**	011101	**[6,7]**	111111	**[9,11]**	101010
[2,6]	101011	**[4,5]**	001110	**[6,8]**	101001	**[9,12]**	110100
[2,7]	111011	**[4,6]**	111100	**[6,9]**	100101	**[10,11]**	100001

Table 6.7: The estimated "optimum" mapping for the 12-2 multiple PPM system.

This mapping can also be adapted for larger systems. For example the mapping generated for the erasure codeword [1,?], as shown in table 6.6, can be used in a $\binom{12}{3}$ multiple PPM system to generate the mapping for the erasure codeword [1,2,?] as shown in table 6.8. Here the $\binom{12}{2}$ mapping forms the basis of the expanded mapping with the extra bit, the Most Significant Bit (MSB), set to 0. This same $\binom{12}{2}$ mapping can be used to generate the mapping for the [2,3,?] erasure sequence, but this time the MSB is set to 1 for all codewords. By using this technique, 12 erasure codewords will be generated with a minimum ESHD, and a large number will be generated that are close to minimum. The remaining codewords are selected to minimise the HD as before.

Erasure Error [1,2,?]	Mapping		
	Linear Increment	**Gray Codes**	**Optimum**
[1,2,3]	0000000	0000000	0000001
[1,2,4]	0000001	0000001	0000100
[1,2,5]	0000010	0000011	0011000
[1,2,6]	0000011	0000010	0000110
[1,2,7]	0000100	0000110	0001100
[1,2,8]	0000101	0000111	0000010
[1,2,9]	0000110	0000101	0100000
[1,2,10]	0000111	0000100	0010000
[1,2,11]	0001000	0001100	0000000
[1,2,12]	0001001	0001101	0001000
Averaged [1,2,?] codeword	0000000	0000101	0000000
Averaged Hamming Distance	15	17	12

Table 6.8: Total and averaged Hamming distance for Linear Increment, Gray Code and "Optimum" mapping of the [1,2,?] ER averaged codeword in a 12-3 multiple PPM system.

To demonstrate the importance of mapping in the final error rate table 6.9 shows the percentage change in error rate for a $\binom{12}{2}$ and $\binom{12}{3}$ multiple PPM system, using

linear increment as a base with a PCM error rate of 1 bit in 10^9 bits, at normalised channel bandwidths of 100, 50, 10 and 1.2. It was not possible to obtain results for $f_n < 1.2$, due to excessive ISI and IFI. For normalised bandwidths greater than 10, it can be seen that there is little to be gained by using linear decrement or Gray code. However, if the mapping is random, there is a large deterioration in error rate for both the $\binom{12}{2}$ and $\binom{12}{3}$ systems. The effect of the optimum code can be clearly seen, with significant drops in error rate particularly for the $\binom{12}{3}$ system. This is because the $\binom{12}{3}$ system has 220 possible codewords to code only 128 PCM codewords. Thus, there is an element of redundancy in the code which means that the optimisation routine has a larger range of codewords available to it, and this increases performance.

f_n		100	50	10	1.2
LD	$\binom{12}{2}$	0.0	0.0	-0.2	-0.2
	$\binom{12}{3}$	2.8	0.0	-1.3	0.0
GC	$\binom{12}{2}$	-2.2	0.2	0.8	-50.1
	$\binom{12}{3}$	-2.4	-5.1	-5.6	-21.8
RD1	$\binom{12}{2}$	12.2	12.3	12.6	74.8
	$\binom{12}{3}$	13.6	12.5	11.0	-1.3
RD2	$\binom{12}{2}$	11.3	10.0	11.7	24.6
	$\binom{12}{3}$	10.3	10.6	9.2	5.0
RD3	$\binom{12}{2}$	4.4	7.0	6.5	-0.3
	$\binom{12}{3}$	18.0	15.2	13.7	-6.4
OPT	$\binom{12}{2}$	-20.3	-19.1	-17.0	-24.8
	$\binom{12}{3}$	-23.32	-25.3	-26.0	-30.0

Table 6.9: Percentage change in error rate for a 12-2 and a 12-3 multiple PPM system using linear increment mapping as the reference and a PCM error rate of 1 bit in 109 pulses. The mappings considered are Linear Decrement (LD), Gray Code (GC), Random (RD1, RD2, RD3) and Optimum (OPT).

When operating with $f_n < 10$, there is a large degree of pulse dispersion and this leads to ISI/IFI and an increase in erasure and wrong-slot errors. Indeed, when $f_n = 1.2$, wrong-slot errors are dominant, and so the Gray code offers the best performance for the $\binom{12}{2}$ system. As discussed in the previous paragraph, the $\binom{12}{3}$ system has some unused codewords which can be used to combat ISI/IFI, and this leads to better performance.

Chapter 7

Higher Order multiple PPM Systems and their Optimum Mapping

Thus far, only the $\binom{12}{Y}$ MPPM family has been considered and the effectiveness of coding demonstrated. Here, original results obtained from mapping experiments [113] on the following multiple PPM families: $\binom{4}{Y}$, $\binom{7}{Y}$, $\binom{15}{Y}$, $\binom{17}{Y}$, $\binom{22}{Y}$, $\binom{28}{Y}$ and $\binom{33}{Y}$ are presented. The PCM equivalent data mapping considered for all systems was the Gray code because

Figure 7.1 shows a surface plot for $\binom{4}{Y}$, $\binom{7}{Y}$, $\binom{12}{Y}$ and $\binom{15}{Y}$ multiple PPM systems operating with a normalized bandwidth of 30. This bandwidth was chosen because the dispersion associated with it lies midway between the dispersion, due to the 100 and 1.2 normalized bandwidths. In this figure, the efficiency factor, η, has been plotted on the vertical axis to demonstrate the optimum, and it is clear that the middle coding systems are the most efficient (especially for the 50-50 point of interest where sensitivity is equally weighted with bandwidth expansion). This optimum partly occurs because the middle systems in the families can code more bits and so are considered

more efficient. The exception to this is the $\binom{4}{Y}$ family, where the $\binom{4}{1}$ system is the most efficient system of the family. This is because all systems in this family can only code 2 bits of PCM and so the most efficient system is the one with the lowest number of pulses. When the sensitivity weighting is greater than 80%, the optimum coding level is generally found in smaller systems.

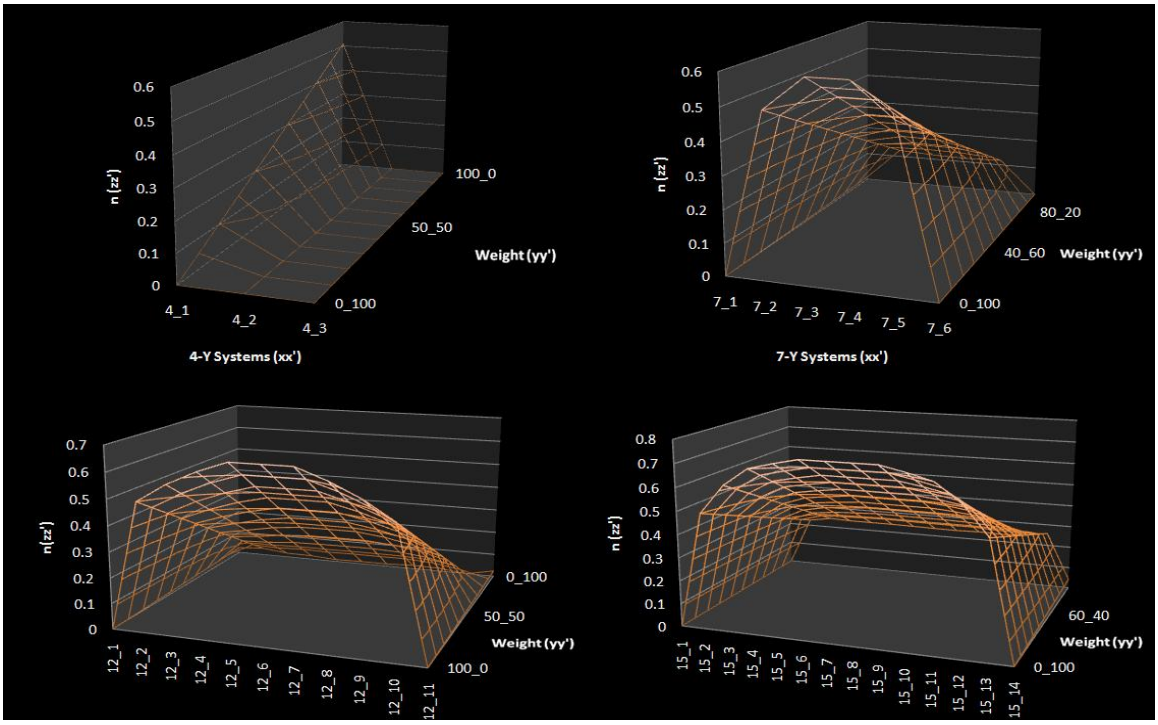


Figure 7.1: Efficiency (surface weighted sum) plot of a 4-Y, 7-Y, 12-Y and 15-Y multiple PPM systems for a normalized bandwidth of 30 (in the yy' axis from 0 to 49, bandwidth expansion is dominant, between 51 to 100, sensitivity is dominant with equality of 50).

Figure 7.2 shows the surface plot for the $\begin{pmatrix}17\\Y\end{pmatrix}$, $\begin{pmatrix}22\\Y\end{pmatrix}$, $\begin{pmatrix}28\\Y\end{pmatrix}$ and $\begin{pmatrix}33\\Y\end{pmatrix}$ multiple PPM families. Again it is clear that the most efficient systems are those in the middle of the family. A comparison test was also performed for multiple PPM systems using linear mapping. The results showed that, as before, the middle systems were the most efficient. In general, Gray codes gave better sensitivities than Linear, except from some systems commonly the smaller or greater systems of the family such as $\begin{pmatrix}4\\1\end{pmatrix}$, $\begin{pmatrix}17\\1\end{pmatrix}$, $\begin{pmatrix}22\\16\end{pmatrix}$, $\begin{pmatrix}22\\20\end{pmatrix}$, $\begin{pmatrix}28\\1\end{pmatrix}$, $\begin{pmatrix}28\\4\end{pmatrix}$, $\begin{pmatrix}33\\1\end{pmatrix}$, $\begin{pmatrix}33\\8\end{pmatrix}$ and from $\begin{pmatrix}33\\28\end{pmatrix}$ to $\begin{pmatrix}33\\31\end{pmatrix}$.

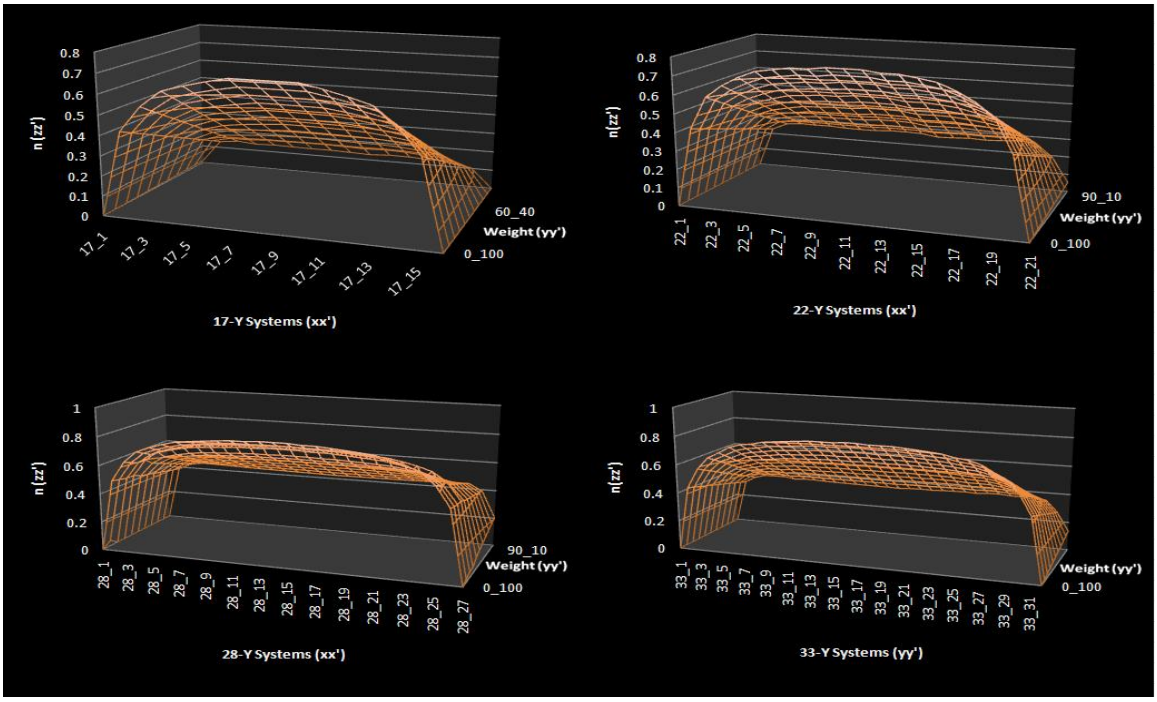


Figure 7.2: Efficiency (surface weighted sum) plot of a 17-Y, 22-Y, 28-Y and 33-Y multiple PPM systems for a normalized bandwidth of 30.

Nikolaidis and Sibley [112] described a methodology of how to obtain an optimum or close to optimum mapping for any MPPM system. The methodology is based on minimising the effects of erasure errors, because these error rates have the greatest impact on the final error probability. Starting from the all zero codeword, the mapping should try to minimise the erasure errors between the weighted codewords and the erasure MLSD weighted codeword. Therefore, the choice of suitable erasure MLSD codewords is vital and very time consuming.

To simplify this process (and fully automate it through software), a practical methodology of finding suitable erasure MLSD codewords that will generate a minimum amount of erasure errors, and hence keep the THD close to minimum, has been suggested [113]. The data specimen of the MPPM system should be divided according to the codewords used from each erasure MLSD codeword. As an example, consider the $\binom{33}{2}$ MPPM system. This system can have 33 erasure MLSD codewords (from [1,?] to [33,?]). Therefore, [1,?] uses 32 codewords, [2,?] 31 codewords, [3,?] 30 codewords, etc. As a result, starting from the all zero codeword, the (number of) codewords used are chosen as the erasure MLSD codewords. So, the [1,?] will have as erasure weighted codeword the all zero codeword (000000000), the [2,?] will have the 32 (000100000), [3,?] the 31 (000011111) and so on.

If the MPPM system has more than 2 pulses, a similar approach is followed. As an example, consider the $\binom{7}{4}$ MPPM system. The system can encode 5 PCM bits and generates 35 erasure MLSD codewords ([1,2,3,?], [1,2,4,?], [2,3,4,?] etc). From these 35 erasure MLSD sequences, six (6) codewords are generated from codewords used by other erasure MLSD codewords. Thus, dividing the data specimen of 32 by 26 (ignoring the 6 erasure MLSD sequences), gives a result of 1.23. Rounding the numbers the erasure MLSD codewords are generated. This approach gave optimum results to the tested systems (as shown in tables 7.5 and 7.6), and was easy to follow. Figure 7.3 (appendix A) shows a part from the methodology used in a $\binom{33}{2}$ MPPM system. Tables 7.1 to 7.4 (appendix B) demonstrate estimated "optimum" mappings for the $\binom{7}{2}$, $\binom{7}{3}$, $\binom{7}{4}$, $\binom{15}{3}$ and $\binom{33}{2}$ MPPM systems (encoding 4, 5, 8 and 9 PCM bits).

Tables 7.5 and 7.6 (appendix B) show the percentage change in error rate for a $\binom{7}{2}$, $\binom{7}{3}$, $\binom{7}{4}$, $\binom{12}{2}$, $\binom{12}{3}$, $\binom{12}{6}$, $\binom{15}{2}$, $\binom{15}{3}$, $\binom{15}{7}$, $\binom{17}{2}$, $\binom{17}{3}$, $\binom{17}{8}$, $\binom{22}{2}$, $\binom{22}{3}$, $\binom{22}{8}$, $\binom{22}{11}$, $\binom{28}{2}$, $\binom{28}{9}$, $\binom{28}{14}$, $\binom{33}{2}$, $\binom{33}{3}$, $\binom{33}{9}$, $\binom{33}{12}$ and $\binom{33}{16}$ multiple PPM system, using linear increment as a base and a PCM error rate of 1 bit in 10^9 bits, at normalised channel bandwidths of 100, 50, 10 and 1.2. It was not possible to obtain results for $f_n < 1.2$ due to excessive ISI and IFI. For normalised bandwidths greater than

10 it can be seen that there is little to be gained by using linear coding or Gray codes. The effect of the optimum code can be clearly seen with significant drops in error rate (in some cases above 30%) especially for large MPPM systems. This is because the large MPPM systems have a large number of possible codewords to code the PCM codewords. Thus there is an element of redundancy in the code which means that the optimisation routine has a larger range of codewords available to it and this increases performance. Even when a series of random mappings were tested there was a large deterioration in error rate for most MPPM systems.

When operating with $f_n < 10$ there is a large degree of pulse dispersion and this leads to ISI/IFI and an increase in erasure and wrong-slot errors. Indeed, when $f_n = 1.2$, wrong-slot [10] errors are dominant and so the Gray code offers the best performance for the MPPM systems.

To measure the efficiency of these mappings an "ideal" mapping is proposed. This mapping is referred as ideal because the THD is minimized. This mapping allows repetitions of MPPM codewords as it is presented in tables 7.7 and 7.8 (appendix B) and cannot be used in a MLSD scheme and therefore, it is only used for comparisons with the (close to) optimum mappings. Table 7.9 and figure 7.4 (appendix B and A respectively) shows the percentage change in error rate for a $\binom{7}{2}$, $\binom{7}{3}$, $\binom{7}{4}$, $\binom{12}{2}$, $\binom{12}{3}$, $\binom{12}{6}$, $\binom{15}{2}$, $\binom{15}{3}$, $\binom{17}{2}$, $\binom{28}{2}$ and $\binom{33}{2}$ multiple PPM system, using linear increment as a base and a PCM error rate of 1 bit in 10^9 bits, at normalised channel bandwidths of 100, 50, 10 and

1.2 using optimum and ideal mappings. The optimum mappings were generally found to have 30% to 50% lower efficiency from the ideal mapping.

Chapter 8

Decoder Optimization using other Correlation Techniques

Rather than use the slope detection normally employed in pulse position modulation systems, Cryan and Sibley [95] proposed a central decision detection with raised cosine filtering in order to eliminate both intersymbol interference and interframe interference. Taking this approach means that the MPPM pulses can spread into adjacent time slots without degrading performance, since the decision point is at the centre of the slot rather than it being a threshold crossing anywhere within the slot. The use of raised cosine filtering ensures that the voltage level due to adjacent pulses is zero at the decision instant, so eliminating ISI. In this case, the output voltage of the ideal raised cosine filter is [95]:

$$v_o(t) = b\eta q R_T \frac{1}{t_s} \frac{\sin\left(\pi \frac{t}{t_s}\right)}{\left(\pi \frac{t}{t_s}\right)} \frac{\cos\left(\pi \frac{t}{t_s}\right)}{\left(1-\left(\frac{2t}{t_s}\right)^2\right)} \tag{33}$$

In figure 8.1, three MPPM patterns were plotted. The same patterns are plotted in figure 4.2 using a raised cosine filtering and, as can be seen, at the central decision points

the contribution from adjacent pulses is zero so eliminating ISI and offering improved performance.

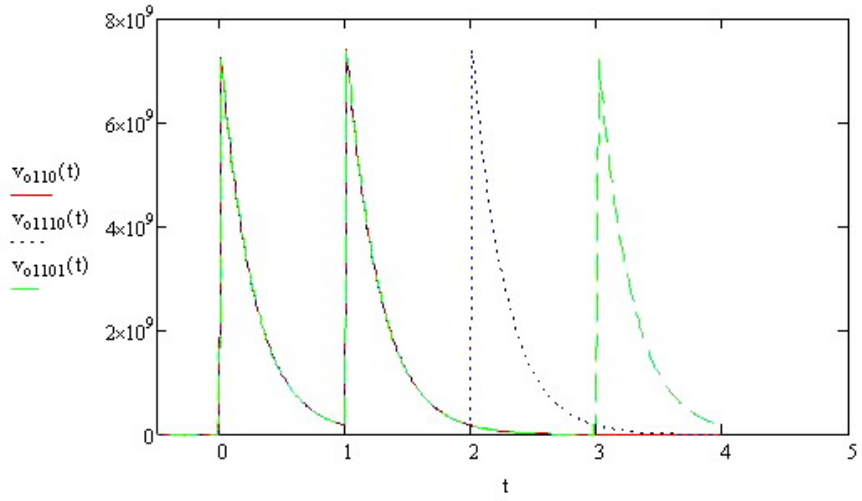


Figure 8.1: 1110, 110 and 1101 multiple PPM sequences with f_n=1.2.

The mean-squared noise voltage using raised cosine equalisation is given by [132]

$$\begin{aligned}\left\langle n_o(t)^2 \right\rangle &= \frac{1}{2\pi}\int_{-\infty}^{\infty} S(\omega)\left|Z_T(j\omega)G(j\omega)\right|^2 d\omega \\ &= \frac{R_T^2}{2\pi}\int_{-\infty}^{\infty} S_0 \left|\frac{\frac{1}{2}\left[1+\cos\left(\omega\frac{t_s}{2}\right)\right]}{e^{\frac{-(\alpha\omega)^2}{2}}}\right|^2 d\omega \\ &= S_0 R_T^2 I_1 B_s\end{aligned} \tag{34}$$

where $B_s = \frac{1}{t_s}$ is the MPPM slot rate and

$$I_1 = \frac{1}{\pi}\int_0^{2\pi} \left| \frac{\frac{1}{2}\left[1+\cos\left(\frac{x}{2}\right)\right]}{e^{\frac{-(\alpha' x)^2}{2}}} \right|^2 dx \qquad (35)$$

Note that the Gaussian pulse variance is normalised to the slot rate such that

$$\alpha' = \alpha / t_s \qquad (36)$$

The I_1 (shape of output pulse) integral is thus independent of the slot time and depends only on the shape of the input pulse that in turn depends upon the normalised fibre bandwidth.

From the results obtained by Cryan and Sibley [95] at the lower fibre bandwidths, where ISI is prevalent, show that the simplified central decision detection combined with raised cosine filtering, gave superior performance from the matched filter scheme. For example, at f_n=0.7 the sensitivity was –26.8 dBm compared to the –19.2 dBm for the more complex Matched Filter-MLD system, representing an improvement of 7.6 dB. As fibre bandwidth increases, the received pulse became narrower and narrower and

eventually it behaved as an impulse. Increasing the bandwidth beyond this point, in this case, f_n=2, led to little further benefit in receiver sensitivity.

Similar results were obtained also from a raised cosine filter combined with a MLSD decoder, as can been seen in table 8.1. For high f_n values, there is little benefit in receiver sensitivity. However, for low f_n values, the improvement is significant. In addition to this, sensitivity results can now be obtained for very low f_n values (below 1.2), in contrast to the Matched filter scheme.

System Sensitivity (in photons/PCM bit)				
	Matched Filter		Raised Cosine Filter	
f_n	12_1	12_2	12_1	12_2
100	930	921	918	899
80	938	927	928	915
60	952	940	939	930
40	979	968	965	960
20	1047	1034	1028	993
10	1180	1167	1167	1155
5	1435	1419	1399	1388
4	1565	1547	1515	1508
3	1790	1771	1720	1704
2	2356	2331	2288	2271
1.8	2608	2581	2578	2548
1.5	3347	3533	3282	3418
1.2	14127	27390	12155	15645
1	X	X	13147	17188
0.8	X	X	15111	22188
0.7	X	X	22135	37897
0.5	X	X	34567	49872

Table 8.1: Sensitivity results for a 12-1 and a 12-2 MPPM system using a Matched and a Raised Cosine filter.

The simplified sensitivity analysis for a $\binom{12}{2}$ multiple PPM system is presented in appendix E.

Chapter 9

Discussion

This thesis has presented the results of an investigation of a MPPM links over a highly dispersive optical channel. The primary objective was to investigate the effects of receiver noise and channel dispersion and the manner in which the erasure, wrong-slot and false alarm errors affect system performance. Three error types (erasure, false alarm and wrong-slot) were considered, caused by noise (ISI and IFI). It was shown from experiments that erasure errors have a greater impact on system sensitivity. Erasure errors can be minimized by lowering the threshold point of detection. Alternatively, False Alarm errors can be minimized by increasing the threshold point of detection. The Wrong Slot errors are only significant in small normalized bandwidth values (f_n below 1.2) and can be neglected. This is logical for a threshold detection scheme because these errors start to occur when the risetime of the pulse is a significant proportion of the slot width.

Another primary objective was to propose a performance analysis. The performance analysis proposed is a Maximum Likelihood Sequence (threshold) Detection scheme. This scheme is based on calculating the probability of specific error sequences. These error sequences are then weighted with the PCM equivalent error rates generated from a software solution that simulates the MLSD scheme. MLSD is generally the optimum detector to recover signals in the presence of ISI, but the complexity of the

MLSD grows exponentially with the channel memory. In this situation, the problem was the large number of sequences to be considered (although it has been shown that erasure errors are the dominant errors the MLSD analysis was implemented so that all error types are analysed and their impact to the final error rate measured. With the MLSD also the importance of the PCM mapping in the final error rate is verified). This made the analysis difficult and time consuming. To overcome this problem a simplified methodology was developed. The methodology is based on grouping similar error sequences and neglecting other error sequences that rarely occur (starting from a 2 pulse system which consist the simplest form of a MPPM system). More detailed the rules used for the grouping are:

1) A pulse (1) can only affect adjacent slots. Therefore, the 100***1*** (with the symbol in error being represented in bold italics) can be considered as a standard error.
2) If consecutive empty slots (more than one) exist, a possible error is only a standard error.
3) An Erasure error can only occur in a pulse (1). In contrast, a False alarm error can only occur in an empty slot (0). A Wrong Slot error can occur either in an empty slot (0) or a pulse (1).
4) The influence of multiple pulses can be approximated with the influence of two consecutive pulses. Therefore error sequences like 1111***1*** can be considered (grouped) as 11***1***.

5) Some error sequences can be neglected because they rarely occur. As an example, consider the 11***1*** sequence in a 12-2 MPPM system (caused by IFI in the [1,2] codeword).

The software was built with an Object Oriented Programming Language but the software was mainly algorithmic. Hence, no Object Oriented Design was needed. As a result, the traditional (modified) waterfall model was chosen as a software development methodology. UML was used for requirements analysis. The software uses Dynamic Memory Allocation to generate the MPPM/PCM/error codewords. Serial processing is used in the algorithms for memory usage reasons. For MPPM systems that encode more than 12 PCM bits, more parallel processes are needed (can be added). Every parallel process added (thread) halves the processing time. The software has a processing limit of MPPM system that can encode up to 32 PCM bits.

From the proposed performance analysis, it was shown that the most efficient systems, if the bandwidth expansion is not considered, are the smaller systems of the MPPM family. This is logical because as the number of pulses is increased, the number of errors generated is also increased. If the bandwidth expansion is considered more important or equally important then the most efficient systems are found in the middle (more bandwidth efficient) systems of the family. To prove this, a methodology referred to as quality factor was developed. This methodology was based on the construction of a (inverse) bath-tube curve combining two different quantities: i) efficiency (measured in photons/detected PCM bit) and ii) bandwidth expansion. From this curve a user can

see the total efficiency of a system referred to as quality. BER is a performance factor used in communication. A methodology to predict the BER of MPPM systems has been also proposed. The results from the BER confirmed, the quality factor, that the systems in the middle of the MPPM systems are more immune to errors.

From the same performance analysis it was also evident about the effectiveness of the MPPM format over the PPM format. In almost every MPPM system examined, the format was superior to the equivalent PPM system (for a 2 pulse MPPM system without even considering the bandwidth expansion).

The effect of data mapping is also considered to this investigation. Experiments showed that Gray codes were very efficient because they minimise the Hamming distance between adjacent codewords (and the probability of a generated error). To be able to minimise the error probability even more, a data mapping that minimises the Hamming distance between all weighted codewords in the MLS Decoder was needed. A methodology (generated from the MLSD algorithm) was described of how to achieve such a mapping. An algorithm identified the dominant error sequences (by isolating the impact of every error sequence), that if minimised, an optimum or close to optimum mapping is obtained. The same algorithm was used with a range of experiments that measured in detail the impact of every error sequence. Doing this, the mathematical models could be substituted by software generating similar results for a certain MPPM system. Therefore, the time consumption for analysis is radically decreased.

Most sensitivity and mapping experiments were done in the efficient 12-2 MPPM system. To generalise the conclusions obtained from this investigation, smaller and higher MPPM codes were also considered. From the experiments, the results obtained showed almost the same pattern (results) as in the 12-Y MPPM family. Again, the middle systems are the most efficient if bandwidth expansion is considered. Some minor exceptions were found in systems were bandwidth expansion does not severely affect the final sensitivity results.

The methodology of how to obtain an optimum or close to optimum mapping has been described in detail. An issue that was not addressed was how to simplify this process (or even automate it) by selecting the correct erased (weighted) MPPM codewords. The solution was the design of a new algorithm that can generate the weighted erased codewords that produce minimum or close to minimum Hamming distances with all the MPPM codewords. The algorithm is simple, by generating the erased codewords according to the data specimen of a specific MPPM system.

To measure these optimum mappings, a bound limit should have been generated (the so called “optimum” mapping was found to be optimum in contrast to the other tested mappings). To verify that these mappings are very efficient a “pseudo”-mapping was generated referred as ideal. The mapping is only generated for comparison reasons with the optimum mappings and cannot be used in a MLSD scheme because they allow repetition of certain codewords. Basically, the idea behind the ideal mapping is that this mapping uses only the all zero codeword, the codeword with only one pulse and

codewords with the one pulse being shifted left or right. This sets the Hamming distance between all codewords (and for every type of error) to 1 (or sometimes to 2). Results obtained showed that the optimum mappings have a 30 to 50% less efficiency from the ideal. This is evidence that the optimum mappings are very efficient coding.

A matched filter has also been selected to be used in the MLSD scheme in the mathematical models. A raised cosine filter was tested for optimisation in a 12-Y MPPM family. In this instance, the system has been shown to offer a sensitivity improvement.

The practical significance of this research is that a designer is able to:

1) Predict the efficiency of any MPPM system.
2) Predict the most efficient system of a MPPM family.
3) Use the most efficient MPPM family or the most efficient system in a MPPM family according to specifications. As an example consider the following: A designer needs to encode 6 PCM bits. Several MPPM systems can encode 6 PCM bits such as: $\binom{12}{2}$, $\binom{13}{2}$, $\binom{14}{2}$, and $\binom{15}{2}$. Which system has the best sensitivity can be easily predicted now through the software solution. On the other hand if the designer wants to encode from 3 up to 9 PCM bits using a specific MPPM family like the $\binom{12}{Y}$ the designer can easily predict which is the most efficient system if bandwidth can be spared (sensitivity is more important) or if sensitivity can be spared (bandwidth is more important) or if both sensitivity and bandwidth are equally important.

4) Measure the BER of any MPPM system.
5) Improve the final error rate of any MPPM system by changing the PCM mapping (and generating the "optimum") and measure this improvement by using the "ideal" mapping.
6) Enhance the efficiency of the MLSD by using a raised cosine filter (especially in very low normalized bandwidths).

Chapter 10

Conclusion and Further Work

The main conclusions of this project are:

- A detailed investigation of an Optical MPPM link operating in a highly disperse optical channel was implemented.

- A novel automated solution designed to predict the equivalent PCM error rates of specific sequences simplified the task (using a full or a simplified analysis). An original mathematical formulation was developed using the Maximum Likelihood Sequence Detection (MLSD) scheme. Using the equivalent PCM error rates of specific sequences generated from software, a full simulation of an MPPM optical link was produced.

- The results indicated the effectiveness of the MPPM format over PPM.

- A measure of coding quality was proposed that accounts for efficiency of coding and bandwidth expansion. This quality factor is a performance factor that can be used in any MPPM system.

- Original results presented for a $\binom{12}{Y}$ MPPM system showed that the most efficient systems were in the middle of the family. Only for very low normalized bandwidths, lower coding was efficient.

- A methodology to predict the Bit Error Rate of any MPPM system was also proposed. The results from the BER confirmed that the systems in the middle of the MPPM systems are the most efficient in terms of sensitivity and bandwidth expansion.

- The effects of linear increment, linear decrement, Gray code and random mapping of data on the performance of a $\binom{12}{Y}$ multiple PPM system were also considered. Simulations using showed that the Gray code was the most effective, although the improvement was not so significant, as it minimizes the Hamming distance between adjacent multiple PPM words.

- Further experiments showed that sensitivity prediction can be obtained exclusively with the use of software, making the analysis simpler and minimizing the time consumption. This prediction was based on the measured impact of every error sequence to the total error probability, and hence to the system's sensitivity.

- A methodology of how to predict the optimum or close to optimum mapping was proposed. The mapping can be reproduced easily for larger systems. The methodology was based on minimizing the (Total) Hamming Distance between the Erased words. Simulations demonstrated the suggested mapping as optimum or close to optimum.

- High order MPPM codes were considered for analysis. The results obtained showed that in most families the most efficient systems were in the middle of the families.

- Close to optimum mappings were presented for high order MPPM systems. The results showed that the estimated mappings were found to be far superior to the efficient Gray codes, linear coding and a series of random mappings.

- An ideal mapping is also proposed. This mapping is referred as ideal because the Hamming Distance (HD) between all code-words is minimized. This mapping allows repetitions of MPPM codewords and cannot be used in a MLSD scheme and therefore, it is only used for comparisons with the (close to) optimum mappings. It was found that the percentage change in error rate of the optimum mappings in contrast to linear coding, are an average of 50% less to the percentage change in error rate of the ideal mappings in contrast to linear coding.

That means that the optimum mappings are (an average) close to the half of the efficiency of the "ideal".

- Other correlation techniques were considered for optimised detection. Results obtained for the $\binom{12}{Y}$ multiple PPM system showed that raised cosine filtering enhances detection.

The author has achieved his objective to investigate MPPM systems on dispersive optical channels. A computerized solution was developed to measure the equivalent PCM error rates of specific MPPM sequences. Using the equivalent PCM error rates of these specific sequences generated from software, a full simulation of an MPPM optical link can be produced. In order to validate the computerized (theoretical) model, this needs to be demonstrated practically.

A methodology, and an algorithm, was proposed of how to predict the optimum or close to optimum mapping. A computerized solution should be produced that could easily generate these mappings for a range of MPPM systems. Other coding techniques [134] should also be considered, and the results obtained should be compared with that obtained from the optimum mapping.

An optimised receiver was tested using a Raised Cosine filter. The results obtained showed that this scheme can give a significant improvement in receiver

sensitivity for low f_n values. The systems tested were from the $\binom{12}{Y}$ MPPM family. The scheme should be tested for various MPPM systems and different mappings in order to fully validate this new scheme.

REFERENCES

1. CALVERT, N.: “Digital Pulse Position Modulation”, PhD Thesis, C.N.A.A, 1988.

2. MASSARELLA, A.J.: “An experimental investigation into the detection of optical digital pulse-position modulation”, PhD Thesis, University of Huddersfield, Octob.1992.

3. GREEFKES, J. A., and RIEMEN, K.: "Code Modulation with Digitally Controlled Companding for Speech Transmission," Philips Tech. Rev., pp. 335-353, 1970.

4. GOLAY, M.J.E: "Note on the theoretical efficiency of information reception with PPM" PIRE, vol. 9, page 1031, 1949.

5. McAULAY, R.J., and SAKRISON, D.J.: “A PPM/PM Hybrid Modulation System”, IEEE Trans. Commun., vol.17, no.4, pp.458-468, 1969.

6. KARP, S. and GAGLIARDI, R.M.: "The design of a Pulse-Position Modulated Optical Communication System", IEEE Trans. in Communications, vol. com -17, Number 6, pp 670-676, December 1969.

7. FORNEY G.D.: "Maximum-Likelihood sequence estimation of digital sequences in the presence of intersymbol interference", IEEE Trans. Inform. Theory, vol.18, pp.363-378, 1972.

8. BLACHMAN, N.M.: "The Spectrum of a TDM PPM Signal", IEEE Trans. Commun., vol.22, no.6, pp.864-866, 1974.

9. BLACHMAN, N.M.: "The SNR Threshold in PPM Reception", IEEE Trans. Commun., vol. com-22, Number 8, pp.1094-1098, August 1974.

10. MUOI, T.V., and HULLETT, J.L.: "Receiver Design for Optical PPM Systems", IEEE Trans. Commun., vol.26, no.2, pp.295-299, 1978.

11. PRATI, G., and GAGLIARDI, R.M.: "Decoding with Stretched pulses in laser PPM communications", vol.31, pp.1037-1045, 1983.

12. GARRETT, I.: "Digital pulse-position modulation for transmission over optical fibre channels with direct and heterodyne detection", IEEE Trans. Commun., vol.31, pp.518-527, 1983.

13. GARRETT, I.: "Digital pulse-Position Modulation over dispersive optical fibre channels", Presented at Int. Workshop on Digital communications, Tirrenia, Italy, 15-19 August 1983.

14. GARRETT, I.: "Digital pulse-position modulation over slightly dispersive optical fibre channels", International symposium on *Information theory*, pp.78-79, St. Jovite, 1983.

15. GAGLIARDI, R.M., and PRATI, G.: "Decoding with Stretched Pulses in Laser PPM Communications", IEEE Trans. Commun., vol.31, no.9, pp.1037-1045, 1983.

16. PRATI, G.: "Joint Pulse Spreading Estimation and Decoding in Stretched Pulse PPM Optical Channels", IEEE Trans. Commun., vol.33, no.8, pp.760-766, 1985.

17. CHEN, C., and GARDNER, C.S.: "Performance of PLL synchronized optical PPM communication system", IEEE Trans. Commun., vol.34, pp.988-994, 1986.

18. LING, G., and GAGLIARDI, R.M.: "Slot Synchronization in Optical PPM Communications", IEEE Trans. Commun., vol.34, no.12, pp.1202-1208, 1986.

19. PIRES, J.J.O., and J.R.F. da ROCHA: "Digital pulse position modulation over optical fibers with avalanche photodiode receivers", IEEE Proc. J., pp.309-313, 1986.

20. CHARBIT M., and BENDJABALLAH, C.: "Probability of Error and Capacity of PPM Photon Counting Channel", IEEE Trans. Commun., vol.34, no.6, pp.600-605, 1986.

21. GAGLIARDI, R.M., and KIM, Y.: "System design for optical PPM communications with diode combining" IEEE Trans. Commun., vol. COM-36, no.2, pp.186-190, 1988.

22. CALVERT, N.M., SIBLEY, M.J.N., and UNWIN, R.T.: "Experimental optical fibre digital pulse-position modulation system", Electron.Lett., 24, pp.129-131, 1988.

23. GARRETT, I., CALVERT, N.M., SIBLEY, M.J.N., UNWIN, R.T., and CRYAN, R.A.: "Optical Fibre digital pulse position modulation", Br.Telecom. Technol. J., vol.7, no. 3, pp.5-11, 1989.

24. DAVIDSON, F., and SUN, X.: "Slot clock recovery in optical PPM communication systems with avalanche photodiode photodetectors", IEEE Trans. Commun., vol.37, pp.1164-1172, 1989.

25. GEORGHIADES, C.N.: "On the synchronizability and detectability of random PPM sequences", IEEE Trans. Inform. Theory, vol.35, pp.146-156, 1989.

26. CALVERT, N.M., SIBLEY, M.J.N., UNWIN, R.T., GARRETT, I., and CRYAN, R.A.: "Optimal filtering of digital PPM transmitted over optical fibre channels", Presented at IEE Colloqium on Electronic Filters, 1989.

27. CRYAN, R.A., UNWIN, R.T., GARRETT, I., SIBLEY, M.J.N., and CALVERT, N.M.: "Optical Fibre digital pulse-position modulation assuming a Gaussian received pulse shape", IEE Proc.J., vol. 137, no. 4, pp.89-96, 1990.

28. CRYAN, R.A., UNWIN, R.T., MASSARELLA, A.J., SIBLEY, M.J.N., and GARRETT, I.: "Coherent detection: n-ary PPM vs. PCM", SPIE Proceedings on Coherent Lightwave Communications, San Jose, California, 1990.

29. CRYAN, R.A., UNWIN, R.T., MASSARELLA, A.J., and SIBLEY, M.J.N.: "Performance analysis of a homodyne digital PPM system", IEE 2nd Bangor Communications Symposium, 1990.

30. ADVANI, M.P., and GEORGHIADES, C.N.: "Jointly Optimal Receivers for the Optical Pulse-Position Modulation Channel", IEEE Trans. Commun., vol.38, no.2, pp.179-186, 1990.

31. MASSARELLA, A.J., and SIBLEY, M.J.N.: "Experimental results on sub-optimal filtering for optical digital pulse-position modulation", Electron. Lett., pp.574-575, 1991.

32. CRYAN, R.A., UNWIN, R.T., MASSARELLA, A.J., SIBLEY, M.J.N., GARRETT, I., and CALVERT, N.M.: "Optical fibre digital PPM: Theoretical and experimental results", Presented at UK-USSR Symp. On Communications and Applications, 1991.

33. CRYAN, R.A., UNWIN, R.T., MASSARELLA, A.J., and SIBLEY, M.J.N.: "A Comparison of Coherent Digital PPM with PCM", European Trans. Telecomm., pp.331-340, 1992.

34. SIBLEY, M.J.N., and MASSARELLA, A.J.: "Detection of digital pulse position modulation over highly/slightly dispersive optical channels", Presented at SPIE Conf. on Video Communications and Fiber Optic Networks, Berlin, 1993.

35. CRYAN, R.A., and UNWIN R.T.: "Optimal and sub-optimal detection of optical fibre digital PPM", IEE Proc. J. Optoelectron., pp.367-375, 1993.

36. SIBLEY, M.J.N.: "Design implications of high-speed digital PPM", Presented at SPIE Conf. on Gigabit Networks, San Jose, CA, 1994.

37. BARRY, J.R.: "Sequence detection and equalization for pulse-position modulation", in Proc. IEEE Int. Conf. Communications (ICC'94), New Orleans, LA, pp.1561-1565, May 1-5, 1994.

38. CRYAN, R.A.: "Optimum and Sub-optimum Detection of Optical Heterodyne n-ary PPM", Journal of Optical Communication, pp.164-169, 1994.

39. AUDEH, M.D., KAHN, J.M., and BARRY, J.R.: "Performance of pulse-position modulation on measured nondirected indoor infrared channels", IEEE Trans. Commun., vol.44, pp.654-659, 1996.

40. AUDEH, M.D., KAHN, J.M., and BARRY, J.R.: "Decision-feedback equalization of pulse-position modulation on measured indoor infrared channels", Proc. IEEE Int. Conf. Communictions (ICC'96), Dallas, TX, vol.2, pp.1220-1226, June 23-27, 1996.

41. REED, I.S., and SOLOMAN, G.: “Polynomial codes over certain finite fields”, J. Soc. Ind. Appl. Math., vol.8, pp.300-304, 1960.

42. VITERBI, A.: “Classification and evaluation of coherent synchronous sampled-late telemetry systems”, IRE Trans. on SET, No.1, page 13, 1962.

43. WOLF, J., and UNDERBOECK, G.: “Trellis Coding for Partial-Response Channels”, Univ. of California at San Diego, La Jolla, CA, 1986.

44. HAGENAUER, J.: “Rate-compatible punctured convolutional codes (RCPC codes) and their applications”, IEEE Trans. on Commun., vol.36, Iss.4, 1988.

45. GARRETT, I.: “Receivers for optical fibre communications”, Radio and Elec. Eng., vol.51, pp.349-361, 1981.

46. McELIECE, R.J.: "Coding for photon channels", IEEE Proc. National Telecomm. Conf., pp.23.3.1-3, 1979.

47. McELIECE, R.J.: "Practical codes for photon communications", IEEE Trans. Inform. Theory., vol.IT-27, pp393-398, 1981.

48. DIVSALAR, D., and GAGLIARDI, R.M.: “Demodulation of PPM for Reed-Solomon coded optical space channel”, in Proc. IEEE Global Telecommun. Conf., Miami, FL, 1982.

49. DIVSALAR, D., and GAGLIARDI, R.M.: “PPM Performance for Reed-Solomon Over an Optical-RF Relay Link”, IEEE Trans. Commun., vol.32, no.3, pp.302-304, 1984.

50. O'REILLY, J.J., and YICHAO, W.: “Line code design for digital pulse-position modulation”, IEE Proc., pt.F, vol.132, no.6, pp.441-446, 1985.

51. YICHAO, W., and O'REILLY, J.J.: “Repeatered optical fibre communication based on line-coded digital PPM transmission”, IEEE Conf., 1985.

52. YICHAO, W., O'REILLY, J.J.:"Syncronisation of line-coded digital PPM in repeatered transmission systems", IEE Proceedings F (Communications, Radar and Signal Processing), vol. 134, no.4, pp.377-382, 1987.

53. ATKIN, G.E., LU, M.H., and FUNG, K.S.: “Performance analysis of Reed-Solomon codes in a slightly dispersive optical PPM system”, MILCOM, vol.L, pp.7-12. 1988.

54. ATKIN, G.E., and CORRALES, H.P.: “A concatenated coding alternative for the optical PPM channel”, 8th Annual International Phoenix Conference on Computers and Communications, pp.155-159, 1989.

55. FORESTIERI, E., GANGOPADHYAY, R., and PRATI, G.: "Performance of convolutional codes in a direct detection optical PPM channel", IEEE Trans. Commun., vol.37, pp.1303-1317, 1989.

56. ATKIN, G.E., and FUNG, K.S.: "Performance analysis of coded optical PPM system using direct and coherent detection", IEEE Proc. J., pp.226-232, 1990.

57. CRYAN, R.A., and UNWIN, R.T.: "Heterodyne n-ary PPM employing Reed-Soloman codes", IEEE GLOBECOM '90, San Diego, California, pp.789-793, 1990.

58. CRYAN, R.A., and UNWIN, R.T.: "Reed-Soloman coded optical fibre digital PPM: Approaching Fundamental Limits", IEEE International Conference on Communication Systems, Singapore, vol.1, 1990.

59. CANNONE, G., MAJUMDER, S.P., GANGOPADHYAY, R., and PRATI, G.: "Performance of Convolutionally Coded Optical *M*-PPM Systems with Imperfect Slot Synchronization", IEEE Trans. Commun., vol.39, no.10, pp.1433-1437, 1991.

60. CRYAN, R.A., and UNWIN, R.T.: "Reed-Solomon coded homodyne digital pulse position modulation", IEE PROCEEDINGS -I, vol.139, no.2, pp.140-146, 1992.

61. MASSARELLA, A.J., and SIBLEY, M.J.N.: "Optical digital pulse-position modulation: Experimental results for heterodyne detection using sub-optimal filtering", Electron. Lett., pp.574-575, 1992.

62. LEE, D.C., AUDEH, M.D., and KAHN, J.M.: "Performance of pulse-position modulation with trellis-coded modulation on nondirected indoor infrared channel", in Proc. IEEE Global Telecommunications Conf., Singapore, pp.1830-1834, 1995.

63. LEE, D.C., KAHN, J.M., and AUDEH, M.D.: "Trellis-coded pulse-position modulation for wireless indoor infrared communications", IEEE Trans. Commun., vol.45, pp.1080-1087, 1997.

64. LEE, D.C.M., and KAHN, J.M.: "Coding and Equalization for PPM on Wireless Infrared Channels", IEEE Trans. Commun., vol.47, no.2, pp.255-260, 1999.

65. CRYAN, R.A.: "High sensitivity digital pulse position modulation systems", PhD Thesis, University of Huddersfield, Sept.1992.

66. ELMIRGHANI, J.M.H.M.O.: "Frame and Slot Synchronization for Optical Fibre Digital Pulse Position modulation", PhD Thesis, University of Huddersfield, Dec.1993.

67. ZWILLINGER, D.: "Differential PPM has a higher throughput than PPM for the bandlimited and average power limited channel", IEEE Trans. of Inform. Theory, vol. IT-34, Issue 5, Pt.2, pp.1269-1273, 1988.

68. SHIROKOV, G.A., and BUKHINNIK, A.: "Evaluation of the reliability of signal transmission along digital optical fibre channels with differential pulse position keying", Telecommunications and Radio Engineering,Pt 2, vol. 39, no. 7, pp.109-111, 1984.

69. PEILE, R.E.: "Error correction, interleaving and differential pulse position modulation", International Journal of Satellite Communications, vol.6, no.2, pp. 173-187, 1988.

70. SHIU, D., and KAHN, J.M.: "Differential Pulse-Position Modulation for Power-Efficient Optical Communication", IEEE Trans. Commun., vol.47, no.8, pp.1201-1210, 1999.

71. BAR-DAVID, I., and KAPLAN, G.: "Information rates of photon-limited overlapping pulse position modulation channels", IEEE Trans. Commun. pp.455-464, 1984.

72. SHALABY, H.M.H.: "A performance analysis of optical overlapping PPM-CDMA communication systems", IEEE Journal of Light. Tech., vol. 17, no. 3, pp.426-434, 1999.

73. DAVIDSON, F.: "Free-space direct detection optical communications with color coded PPM signalling", IEEE GLOBECOM 84, vol.2, pp.944-948, 1984.

74. BAYOUMI M., and DAVIDSON, F.: "Performance characteristics of laser diode communication systems with APD receivers and color-coded PPM signalling", Proceedings of the 23rd Conference on Communication, Control and Computing, IL, USA: Univ. Illinois, pp.388-389, 1985.

75. DAVIDSON, F.: "Direct detection optical communication with color coded pulse position modulation signalling", IEEE Trans. Commun., vol.33, pp.273-276, 1985.

76. DAVIDSON, F., and BAYOUMI M.: "Theoretical performance of direct detection optical communication with AlGaAs laser transmitters, avalanche

photodiode detectors and color-coded PPM signalling", Journal of Lightwave Technology, vol.LT-5, no.11, pp.1574-1583, 1987.

77. LEE, G.M., and SCROEDER, G.W.: "Optical Pulse Position Modulation with Multiple Positions per Pulsewidth", IEEE Trans. Commun.., vol. COM-25, pp.360-365, 1977.

78. GOL'DSTEYN, A. and FREZINSKIY, B.: "An Investigation of the Transmission of a Multi-Position PPM Optical Signal Through a Communications Line Containing Repeaters", Radio. Eng. Electron Phys., vol.24, pp.65-71, 1978.

79. MAROUGI, S.D., and SAYHOOD, K.H.: "Noise performance of pulse interval modulation systems", International Journal of Electronics, vol.55, no. 4, pp.603-614, 1983.

80. FYATH, R.S., et al: "Spectrum investigation of pulse-interval modulation", International Journal of Electronics, no. 5, pp.597-601, 1985.

81. GHASSEMLOOY, Z., HAYES, A.R., SEED, N.L., and KALUARACHCHI, E.D.: "Digital pulse interval modulation for optical communications", IEEE Communications Magazine, vol.36, Iss.12, pp.95-99, 1998.

82. ALDIBBIAT, N.M., GHASSEMLOOY, Z., and McLAUGHLIN, R.: "Performance of dual header pulse interval modulation (DH-PIM) in optical wireless communications", IEE Proceedings Optoelectronics, vol.13, Iss.5, pp.138-142, 2001.

83. ALDIBBIAT, N.M., GHASSEMLOOY, Z., and McLAUGHLIN, R.: "Indoor optical wireless systems employing dual header pulse interval modulation(DH-PIM)", International Journal of Communication Systems, 2005.

84. YEMIN, V., I., and PETRICH, A., V.: "Noise immunity of the reception of optical multiposition pulse-time modulated signals under intersymbol noise conditions", Radiotekhnika, no.7, pp.86-87, 1984.

85. SUGIYAMA, H., and NOSU, K.: "multiple PPM: A Method for Improving the Band-Utilization Efficiency in Optical PPM", Journal of Lightwave Technology, vol. 7, no.3, pp.465-472, 1989.

86. ELMIGHANI, J.M.H., CRYAN, R.A., and CLAYTON, F.M.: "Optical fibre multiple PPM frame synchronization", Microwave and opt. Tech. Lett., vol.6, no.14, pp.800-804, 1993.

87. MAJUMDER, S., P., and GANGOPADHYAY, R.: "Performance of convolutionally coded OOK and PPM signaling in a direct detection optical communication", IETE Technical Review, vol.7, Iss.3, pp.194-197, 1990.

88. BUDINGER, J., M., VANDERAAR, M., WAGNER, P., and BIBYK, S.: "Combinatorial pulse position modulation for power-efficient free-space laser communications", in Proc. SPIE, Los Angeles, CA, Jan. 20-21, pp.214-225, 1993.

89. FARES, D.A.: "Heterodyne detection of multipulse signaling in optical communication channels", Microwave Optical Technology Letter, vol.9, no.4, 1995.

90. VELIDI, R., and GEORGHIADES, C.N.: "Frame synchronization for optical multi-pulse pulse position modulation", IEEE Trans. Commun. vol.43, pp.1838-1843, 1995.

91. PARK, H., and BARRY, R.: "The performance of multiple-pulse-position modulation on multipath channels", IEE Proc. Optoelectron., vol.143, no.6, pp.360-364, 1996.

92. PARK, H., and BARRY, R.: "Performance analysis and channel capacity for multiple-pulse position modulation on multipath channels", IEEE International

Symposium on Personal, Indoor and Mobile Radio Communications, pp.247-251, 1996.

93. VELIDI, R., and GEORGIADES, C.N.: "On Symbol Synchronization of MPPM Sequences", IEEE Trans. Commun., vol.46, no.5, pp.587-589, 1998.

94. SIBLEY, M.J.N.: "Analysis of multiple pulse position modulation when operating over graded-index, plastic optical fibre", IEE Proc.-Optoelectron., vol.151, no.6, December 2004.

95. CRYAN, R.A., and SIBLEY, M.J.N.: "Multiple pulse position modulation employing raised cosine filtering", IEE Proceedings Optoelectronics, vol.153, Iss.4, pp.205-211, 2006.

96. ATKIN, G.E., and FARES, D.A.: "Coded multipulse position modulation in a noisy channel", Microwave and Optical Technology Letters, vol.2, no.9, pp.336-340, 1989.

97. HERRO, M.A., HU, L.: "Multi-pulse PPM and a new look at coding for direct detection optical channels using APD receivers", Proceedings of the 23rd Conference on Communication, Control and Computing, IL, USA: University of Illinois, pp 401-410, 1989.

98. TAKAHASHI, M., et al: "Capacity and effects of reed-solomon codes on multi-pulse PPM in optical communications", IEICE Trans., vol.E-72, pp.1198-1203, 1989.

99. ATKIN, G.E. and FUNG, K.S.: "Coded multipulse modulation in optical communication systems", IEEE Trans. in Communications, vol. com-42, Number 3, pp 574-582, March 1994.

100. PARK, H., and BARRY, R.: "Partial-response precoding scheme for multiple pulse-position modulation", IEEE Proc. Opto., vol. 150, no. 2, pp.133-137, 2003.

101. PARK, H., and BARRY, R.: "Trellis-Coded Multiple-Pulse-Position Modulation for Wireless Infrared Communications", IEEE Trans. Commun., vol. 52, no. 4, pp.643-651, April 2004.

102. GARRIDO-BALSELLS, J.M., GARCIA-ZAMBRANA, A., and PUERTA-NOTARIO., A.: "Novel block coding method for rate-adaptive optical wireless communications systems" in Proc. IEEE Global Telecommunications Conference (Globecom 05), St. Louis, Nov. 2005.

103. GARRIDO-BALSELLS, J.M., GARCIA-ZAMBRANA, A., and PUERTA-NOTARIO., A.: "Variable weight MPPM technique for rate-adaptive optical wireless communications" IEE Electronics letters, 2006.

104. SIBLEY, M.J.N.: "Dicode pulse-position modulation: a novel coding scheme for optical-fibre communications", IEE Proc. Optoelectronics, vol.150, no.2, pp.125-132, April 2003.

105. SIBLEY, M.J.N.: "Analysis of dicode pulse position modulation using a PINFET receiver and a slightly/highly dispersive optical channel", IEE Proc. Optoelectronics, vol.150, no.3, pp.205-209, June 2003.

106. MOHANTY, N.,C.: "Estimation of Delay of *M* PPM signal in Laguerre Communications", IEEE Trans. Commun., vol.22, no.5, pp.713-714, 1974.

107. MANSURIPUR, M., and GOODMAN, J.W., RAWSON, E.G., and NORTON, R.E.: "Fiber Optics Receiver Error Rate Prediction Using the Gram-Charlier Series", IEEE Trans. Commun., vol.28, no.3, pp. 401-407, 1980.

108. GAGLIARDI, R.M., and PRATI, G.: "On Gaussian Error Probabilities in Optical receivers", IEEE Trans. Commun., vol.28, no.9, pp.1742-1746, 1980.

109. HAYAT, M.M., SALEH, B.E.A., and GUBNER, J.A.: "Bit-Error Rates for Optical Receivers Using Avalanche Photodiodes with Dead Space", IEEE Trans. Commun., vol.43, no.1, pp.99-105, 1995.

110. NIKOLAIDIS, K., and SIBLEY, M.J.N.: "Investigation of an Optical Multiple PPM Link over a Highly Dispersive Optical Channel", IET Optoelectronics, Volume 1, Issue 3, p. 113-119, June 2007.

111. NIKOLAIDIS, K., and SIBLEY, M.J.N.: "Theoretical Investigation into the Effects of Data Mapping in an Optical multiple PPM Link", Electronics Letters, Volume 43, Issue 19, p. 1042-1044, September 2007.

112. NIKOLAIDIS, K., and SIBLEY, M.J.N.: "Optimum Mapping in an Optical Multiple PPM link using a Maximum Likelihood Sequence Detection Scheme", IET Optoelectronics, Volume 3, Issue 1, p. 47-53, February 2009.

113. NIKOLAIDIS, K., and SIBLEY, M.J.N.: "Investigation of Higher Order Optical Multiple PPM Links over a Highly Dispersive Optical Channels", submitted IET Optoelectronics paper.

BIBLIOGRAPHY

114. ASHEDEN, P.J.: "The Designer's Guide to VHDL", Morgan Kaufmann Publications, ISBN 1-55860-270-4, 1996.

115. PERRY, D.L.: "VHDL: Programming By Example", Morgan Kaufmann Publications, ISBN 1-58751-311-7, 2002.

116. JAWADEKAR, W.: "Software Engineering: Principles and Practice", McGraw Hill, ISBN 0070583714, 2004.

117. GHEZZI, C., JAZAYERI, M., and MANDRIOLI, D.: "Fundamentals of Software Engineering", 2nd Edition, Pearson Education (Addison Wesley) ISBN 0-13-099183-X, 2003.

118. VLIET, H.: "Software Engineering: Principles and Practice", 2nd Edition, John Wiley and Sons, ISBN 0-471-97508-7, 2002.

119. BRAUDE, E.J.: "Software Engineering: An Object-Oriented Perspective", John Wiley and Sons, ISBN 0-471-32208-3, 2001.

120. PARNAS, D.L., WEISS, D.M. and HOFFMAN, D. M.: "Software Fundamentals: Collected Papers", Addison-Wesley, ISBN 0-201-70369-6, 2001.

121. PRESSMAN, R.S.: "Software Engineering: A Practicioner's Approach", 5th Edition, McGraw Hill, ISBN 0-07-365578-3, 2001.

122. SOMMERVILLE, I.: "Software Engineering", 6th Edition, Pearson Education (Addison Wesley), ISBN 0-201-39815-X, 2001.

123. PFLEEGER, S.L.: "Software Engineering: Theory and Practice", Prentice Hall, ISBN 0-13-624842-X, 1998.

124. RANDELL, B. and BUXTON, J.N.: "Software Engineering Techniques", report of a conference sponsored by the NATO Science Committee, Rome, Italy, 164pp, Scientific Affairs Division, NATO (1970), Brussels 27-31 Oct. 1969.

125. BOOCH, G., RUMBAUGH, J. and JACOBSON, I.: "Unified Modeling Language User Guide", 2nd Edition, Prentice Hall Publications, ISBN: 0321267974, 2005.

126. MILES R.: "Learning UML 2.0", O'REILLY Publications, ISBN: 0596009828, 2006.

127. DEITEL, and DEITEL: "C++ How to Program", Chapter 9, "Test Patterns", 4th Edition, Prentice Hall Publications, ISBN 918-114-123941-1, 2007.

128. SCHACH, S.R.: "Object Oriented Software Engineering", McGraw Hill, ISBN 978-007-125941-5, 2008.

129. BRUEGGE, B. and DUTOIT, A.H.: "Object-Oriented Software Engineering: Using UML, Patterns and Java", Pearson Education, ISBN 0-13-191179-1, 2004.

130. LETHBRIDGE, T.R. and LAGANIERE, R.: "Object-Oriented Software Engineering: Practical Software Development using UML and Java", McGraw Hill, ISBN 0-07-709761-0, 2001.

131. BRUEGGE, B.: "Object-Oriented Software Engineering: Conquering Complex and Changing Systems", Prentice Hall, ISBN 0-13-489725-0, 1999.

132. PERSONICK, S.D.: 'Receiver design for digital fiber optic communication systems, I, II', Bell Syst. Tech. J., 52, pp. 843-886, 1973.

133. HARDY, G. and WRIGHT, E.: "An introduction to the theory of numbers", 5th ed., Oxford Univ. Press, 1979.

134. SWEENEY, P.: "ERROR CONTROL CODING: FROM THEORY TO PRATICE", WILEY Publications, ISBN 9-780470-844567, 2002.

APPENDIX A

Figures

---

17 FIGURES

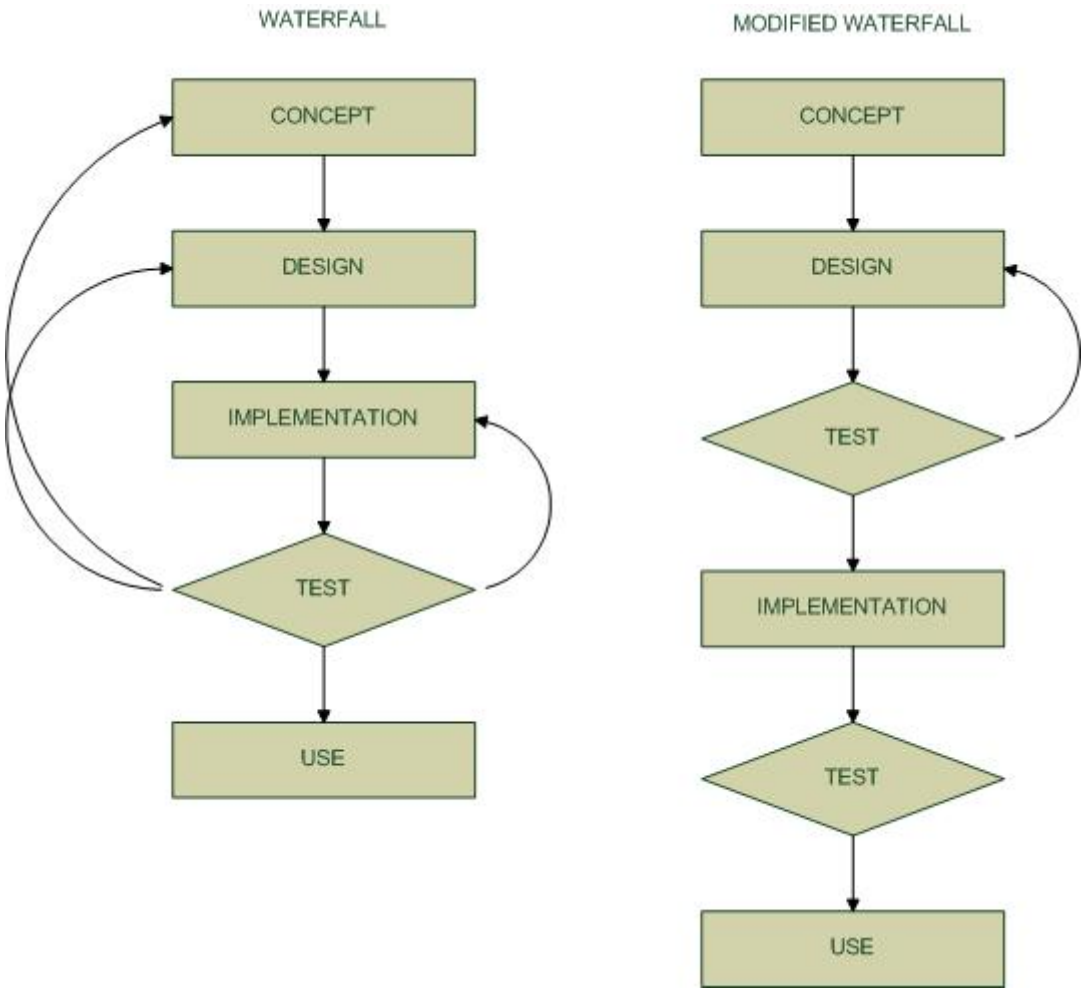


Figure 3.1: The Waterfall and Modified Waterfall Design Methodologies.

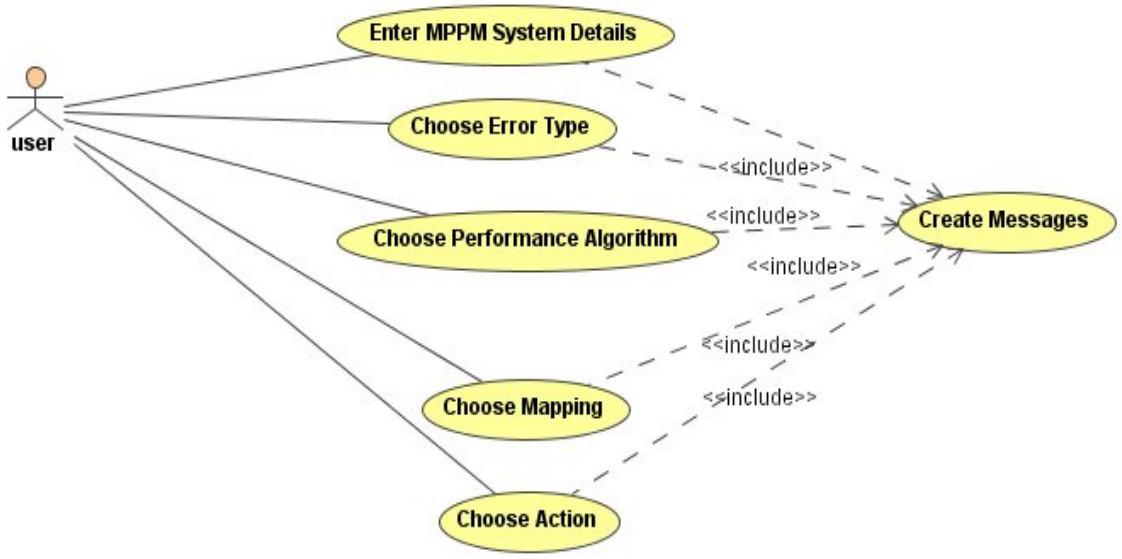


Figure 3.2: A level-0 Use Case Diagram of the software.

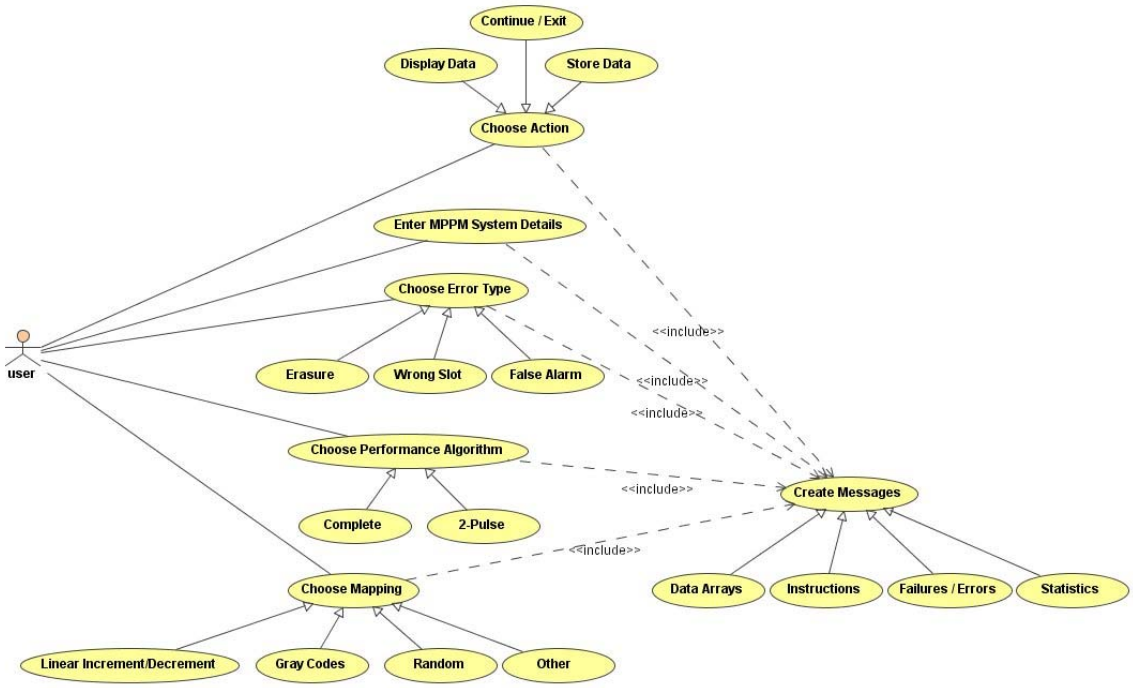


Figure 3.3: A level-1 Use Case Diagram of the software.

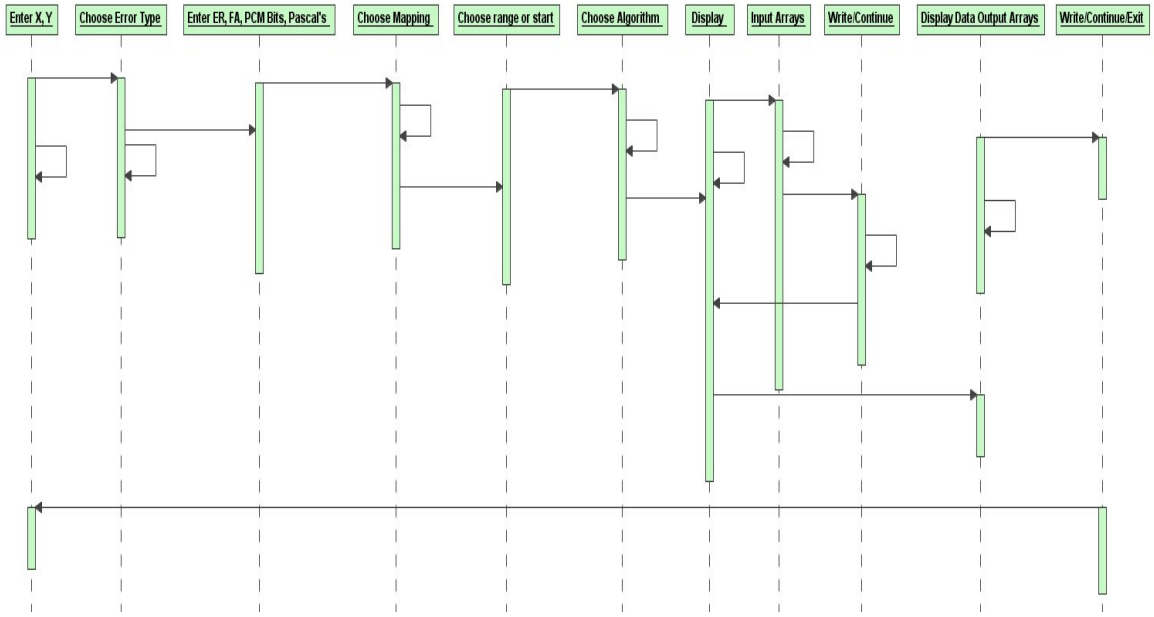


Figure 3.4: A Sequence Diagram of the software.

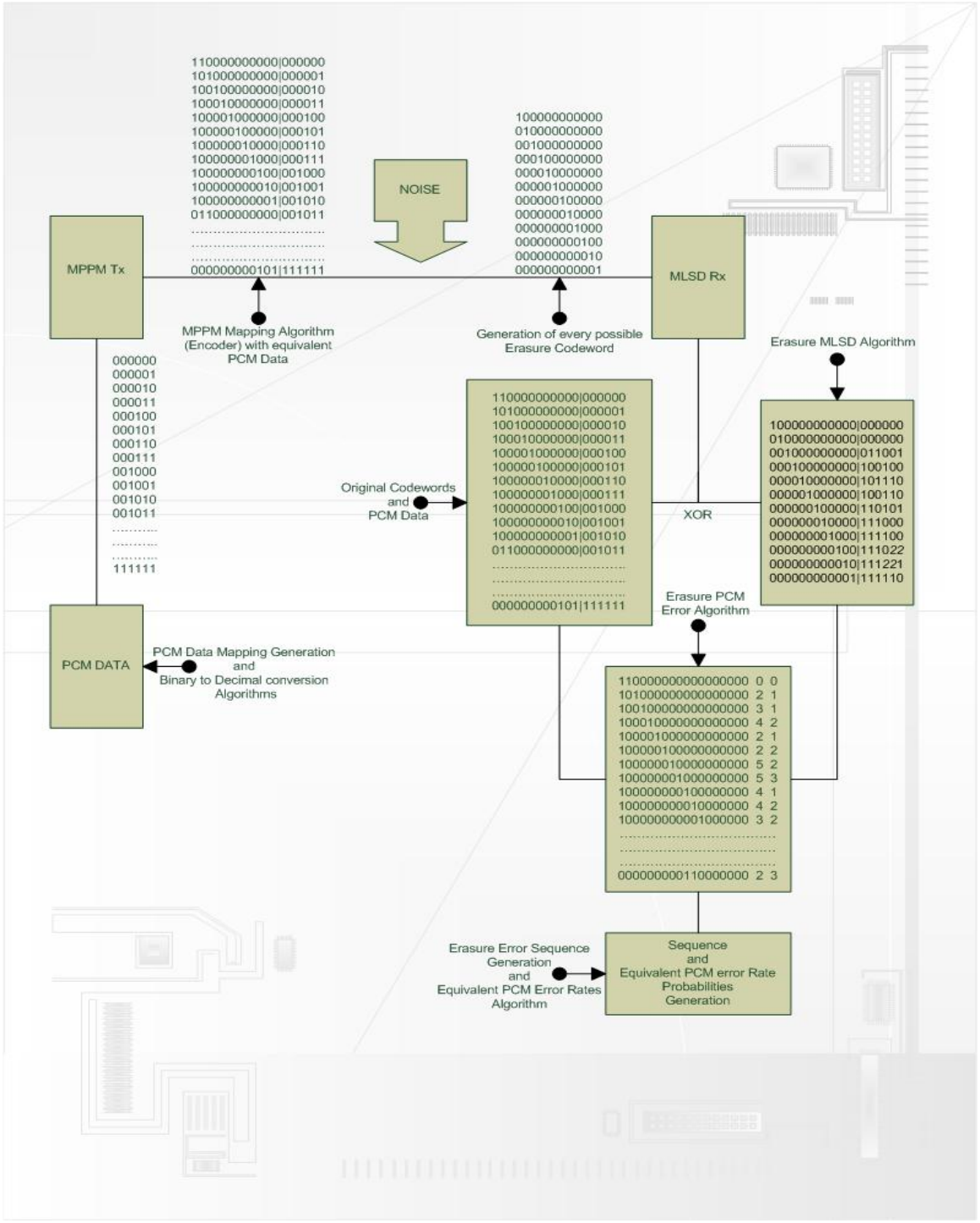


Figure 3.5: The software processing for an Erasure error in a $\binom{12}{6}$ *MPPM system.*

Figure 3.6: A Console-32 Interface.

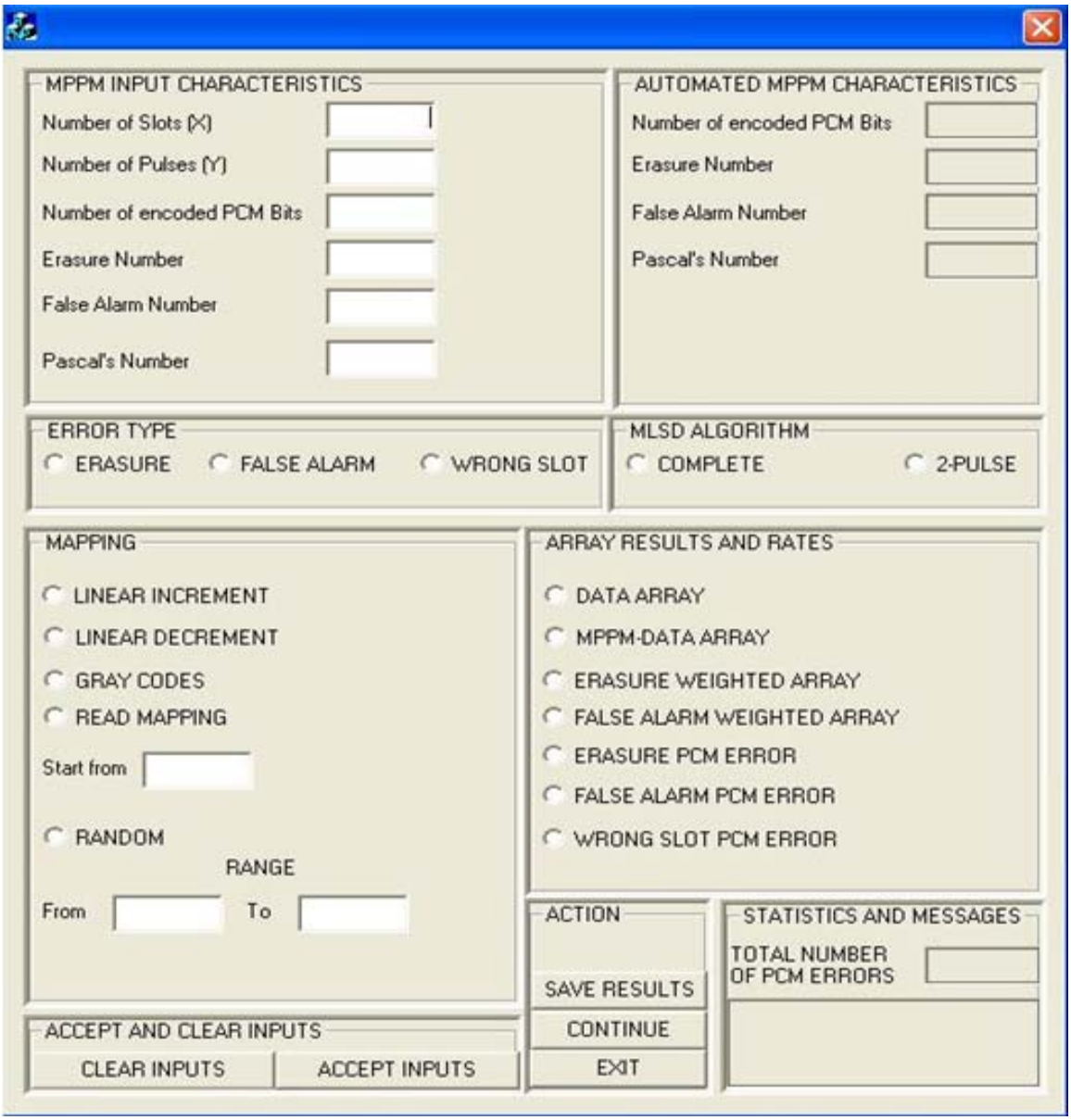


Figure 3.7: A Visual MFC Interface.

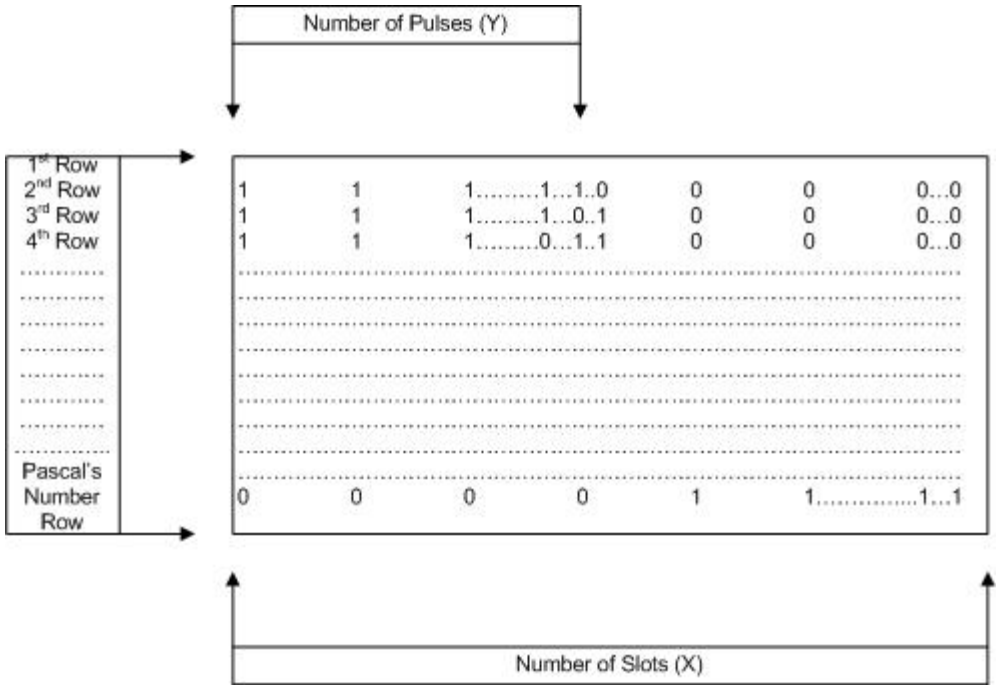


Figure 3.8: An $\binom{X}{Y}$ *MPPM system.*

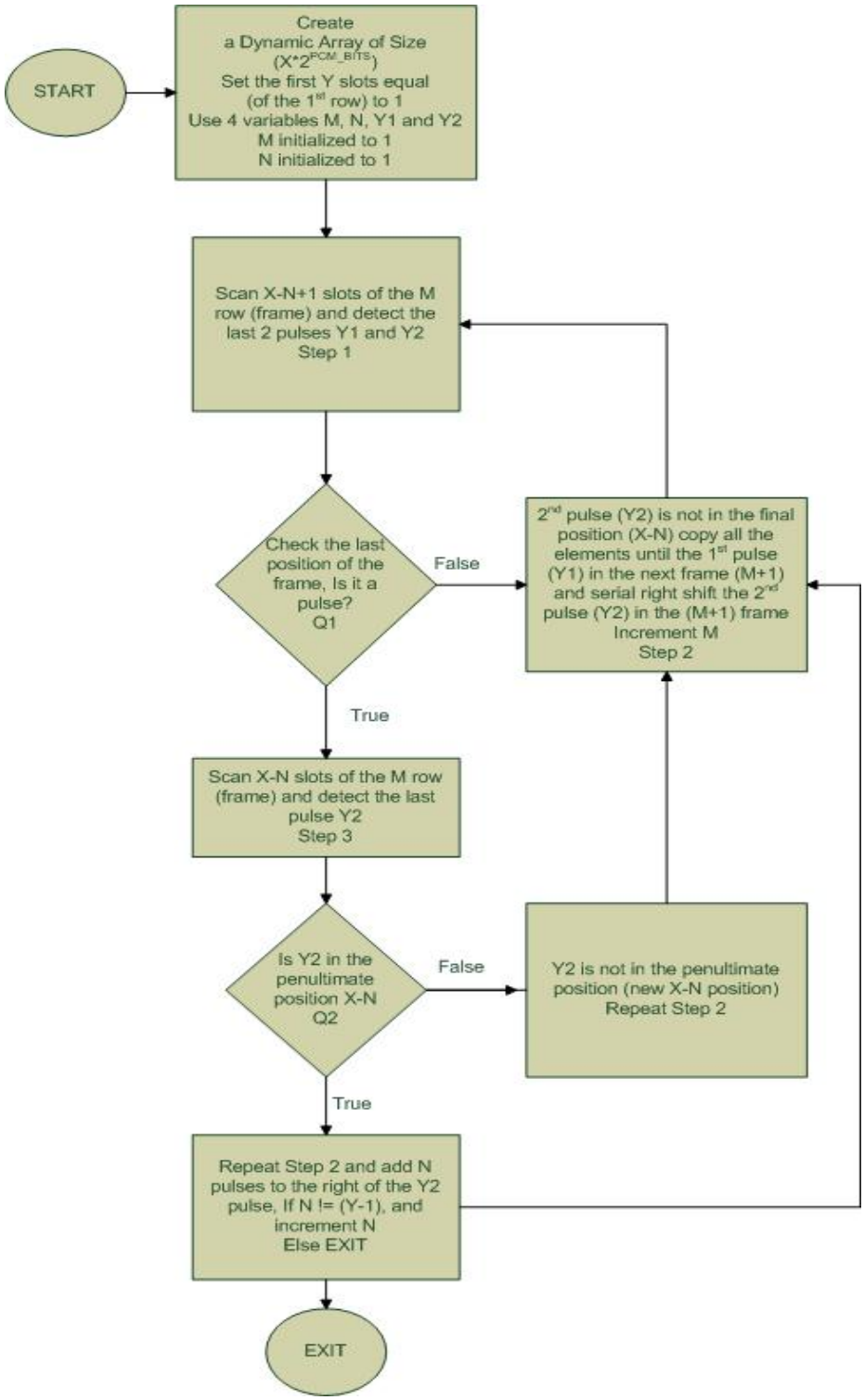


Figure 3.9: The $\binom{X}{Y}$ *MPPM mapping algorithm.*

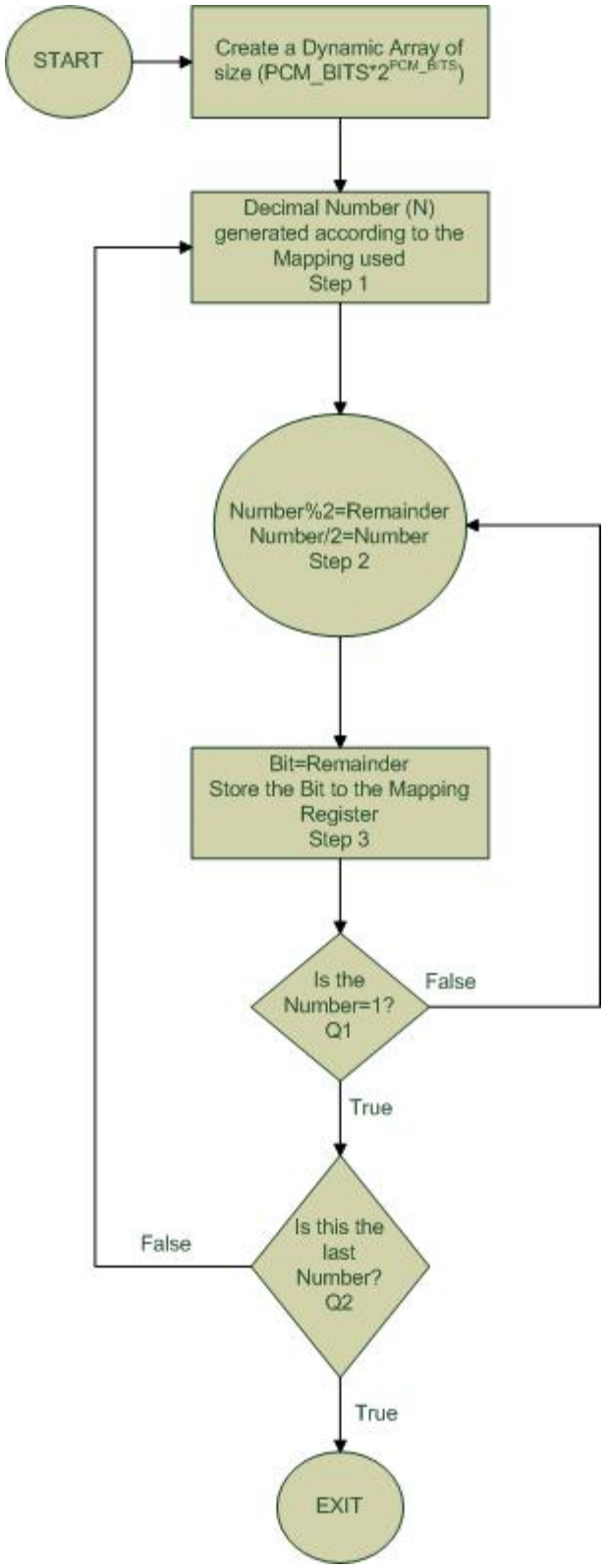


Figure 3.10: The Decimal-to-Binary Conversion.

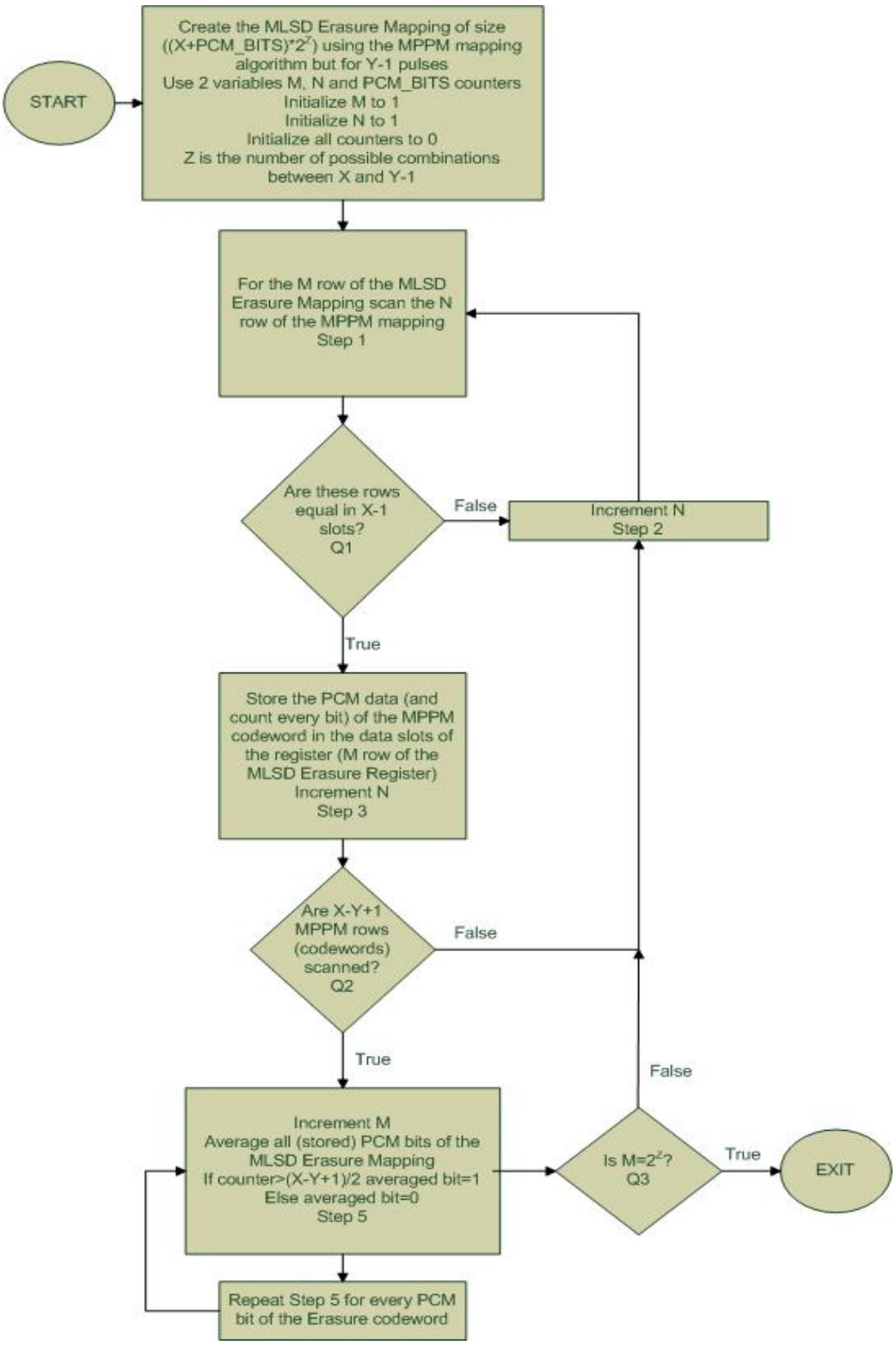


Figure 3.11: The ER MLSD algorithm.

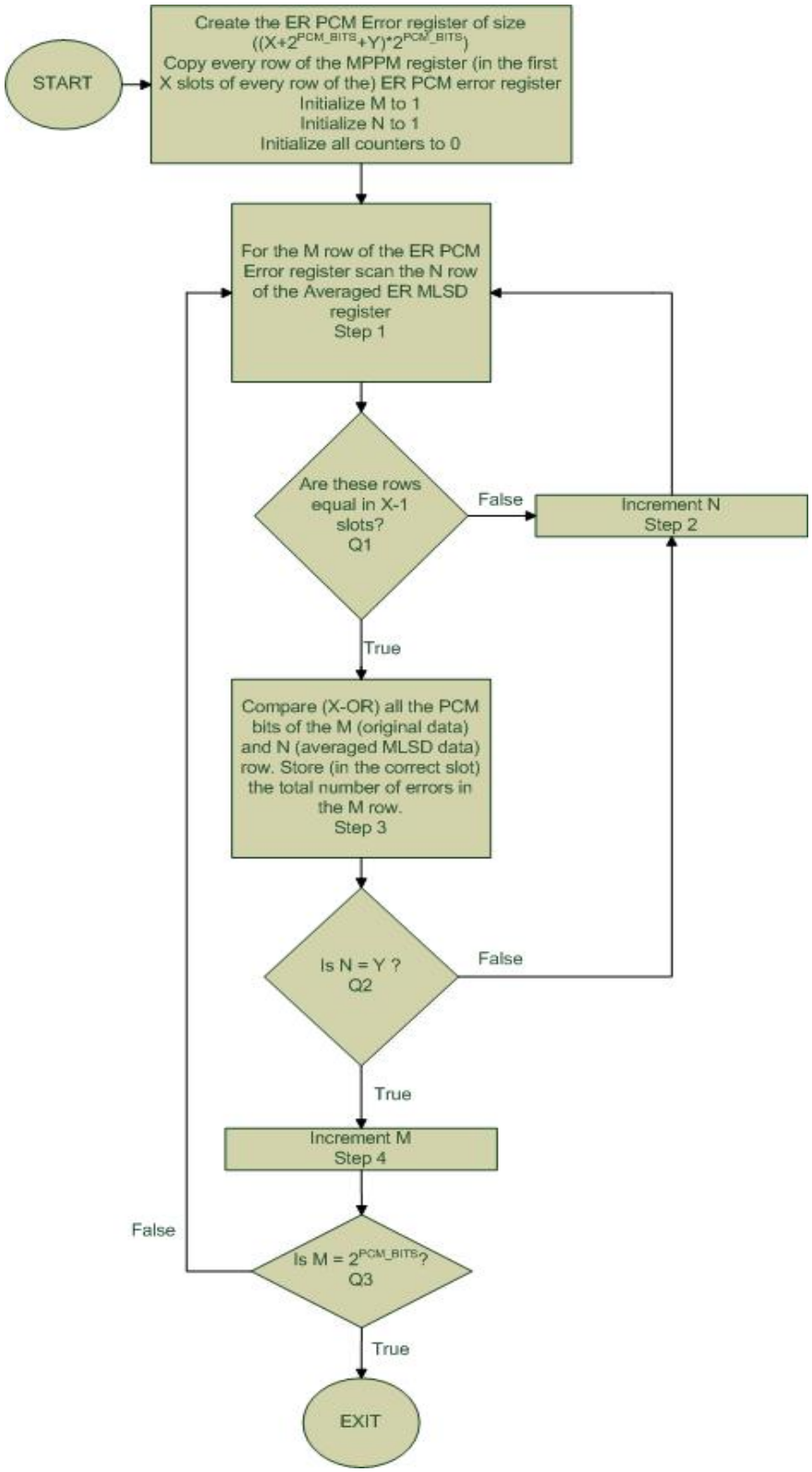


Figure 3.12: The ER PCM error algorithm.

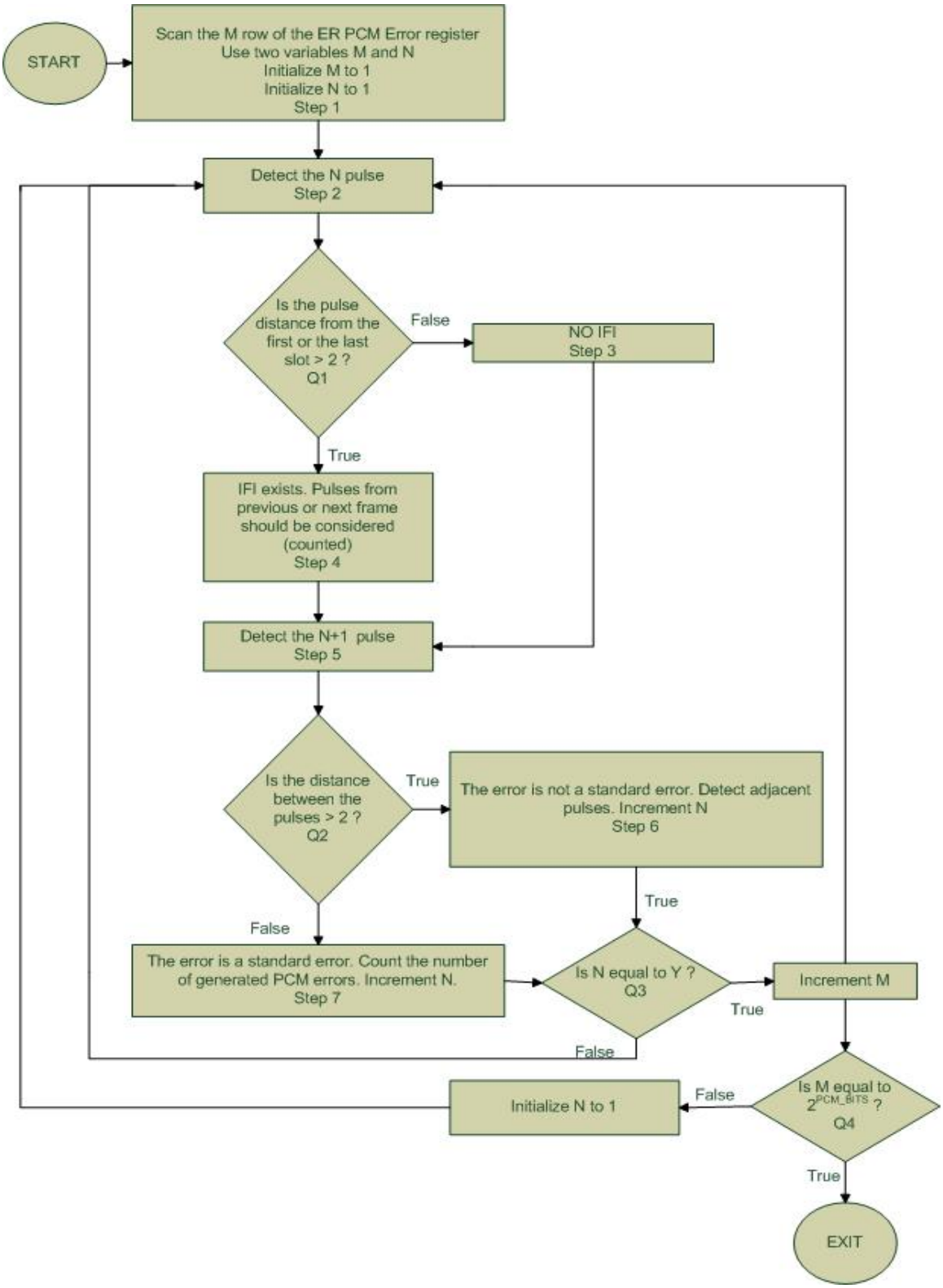


Figure 3.13: The Erasure error sequence detection algorithm.

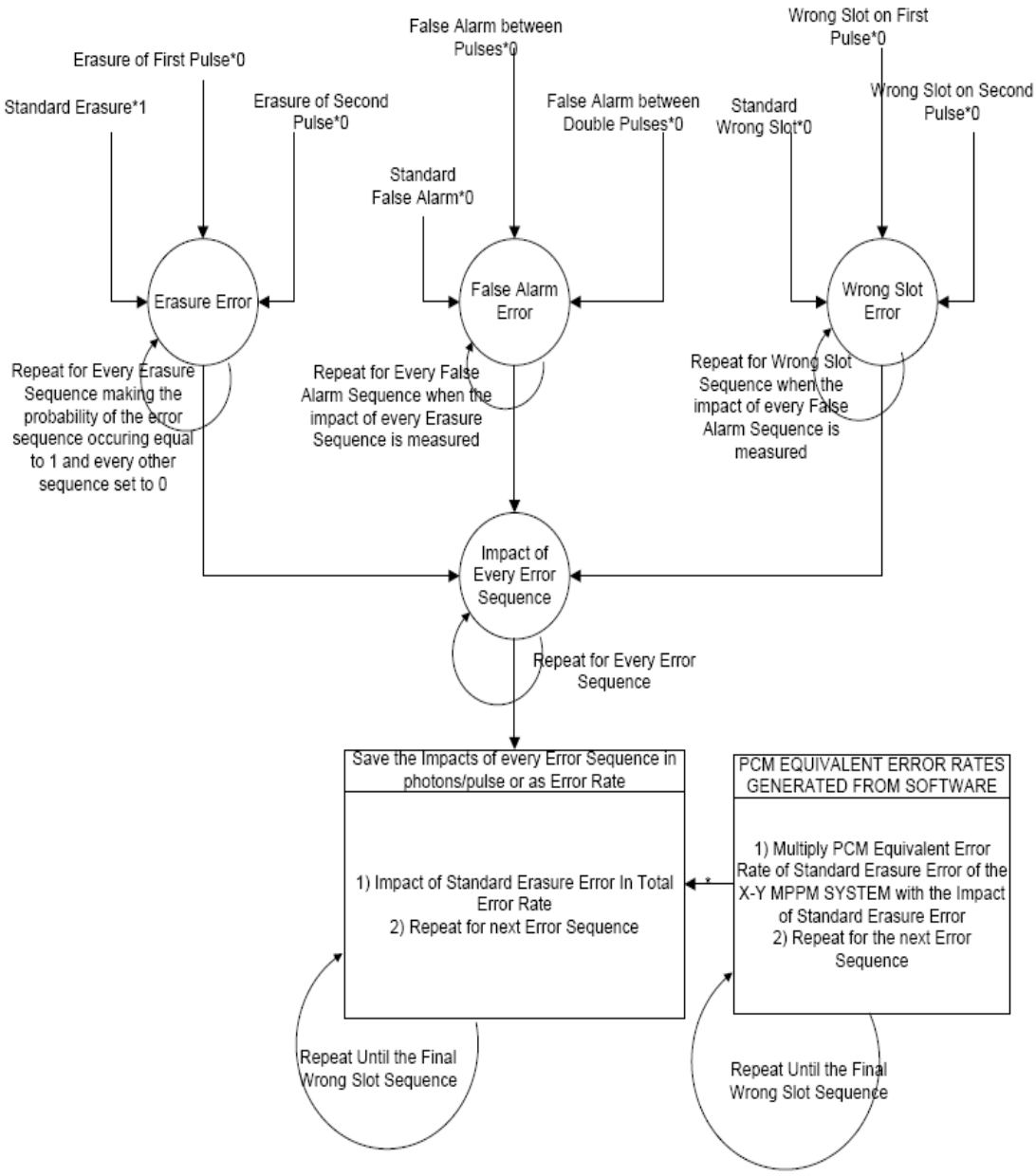
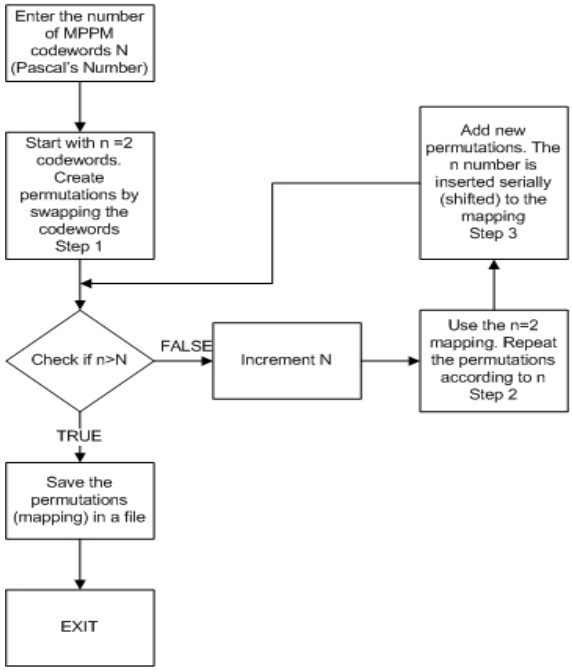


Figure 6.1: Data flow diagram of the optimization routine.



STEP 1
Mapping n=2

1	2
2	1

STEP 2
Mapping n=3
Repeat the permutations of n=2 three (3) times

1	2
1	2
1	2
2	1
2	1
2	1

STEP 3
Mapping n=3
Add three (n) as demonstrated below

1	2	3
1	3	2
3	1	2
3	2	1
2	3	1
2	1	3

STEP 2
Mapping n=4
Repeat the mapping (permutations) in Step 3 four (4) times

1	2	3
1	2	3
1	2	3
1	2	3
1	3	2
1	3	2
1	3	2
1	3	2
3	1	2
3	1	2
3	1	2
3	1	2
3	2	1
3	2	1
3	2	1
3	2	1
2	3	1
2	3	1
2	3	1
2	3	1
2	1	3
2	1	3
2	1	3
2	1	3

STEP 3
Mapping n=4
Add four (4) as demonstrated below

1	2	3	4
1	2	4	3
1	4	2	3
4	1	2	3
4	1	3	2
1	4	3	2
1	3	4	2
1	3	2	4
3	1	2	4
3	1	4	2
3	4	1	2
4	3	1	2
4	3	2	1
3	4	2	1
3	2	4	1
3	2	1	4
2	3	1	4
2	3	4	1
2	4	3	1
4	2	3	1
4	2	1	3
2	4	1	3
2	1	4	3
2	1	3	4

Figure 6.2: Data flow diagram of the maximum permutation algorithm.

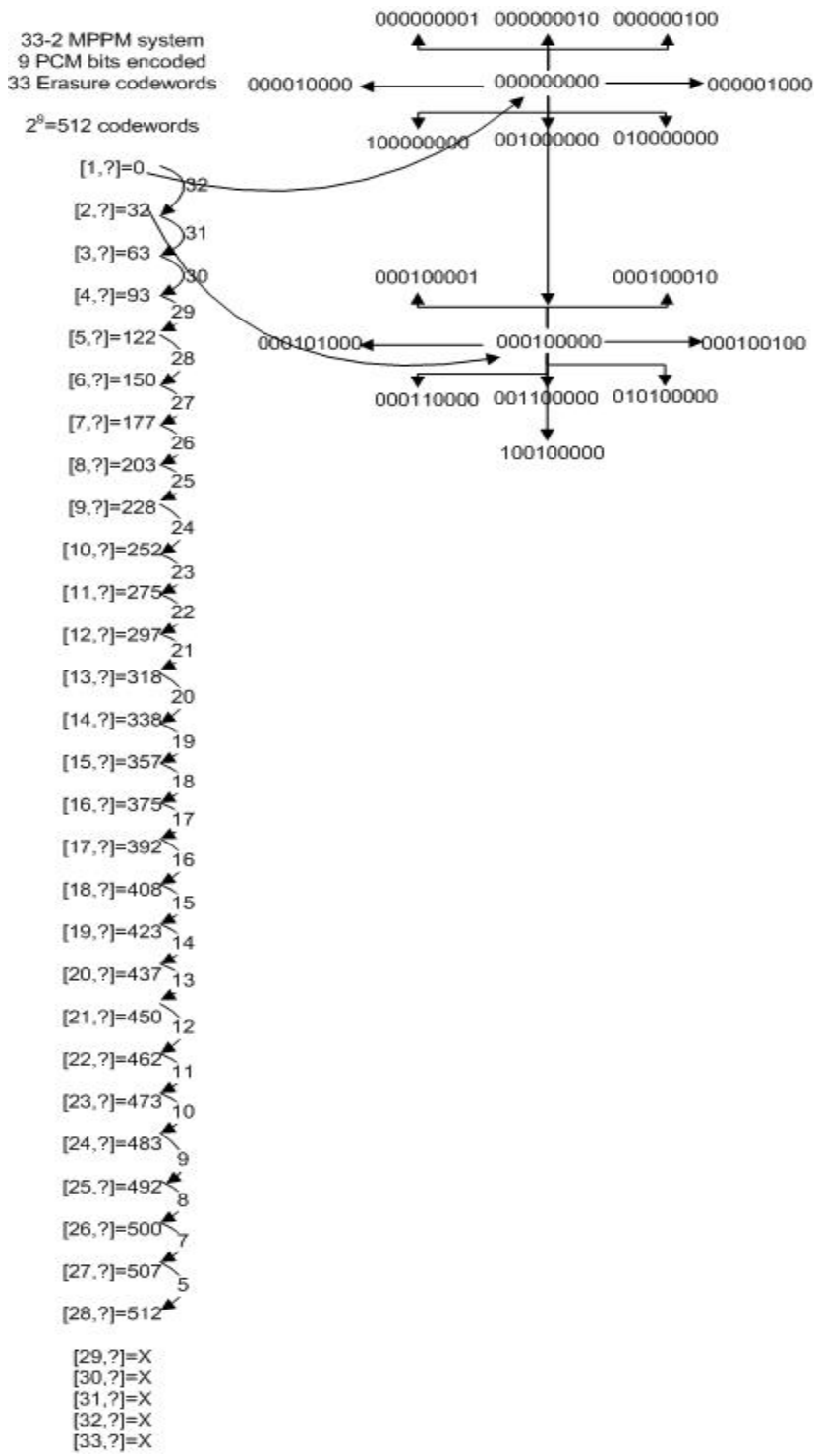


Figure 7.3: The methodology used to obtain optimum mapping in a $\binom{33}{2}$ *MPPM system*

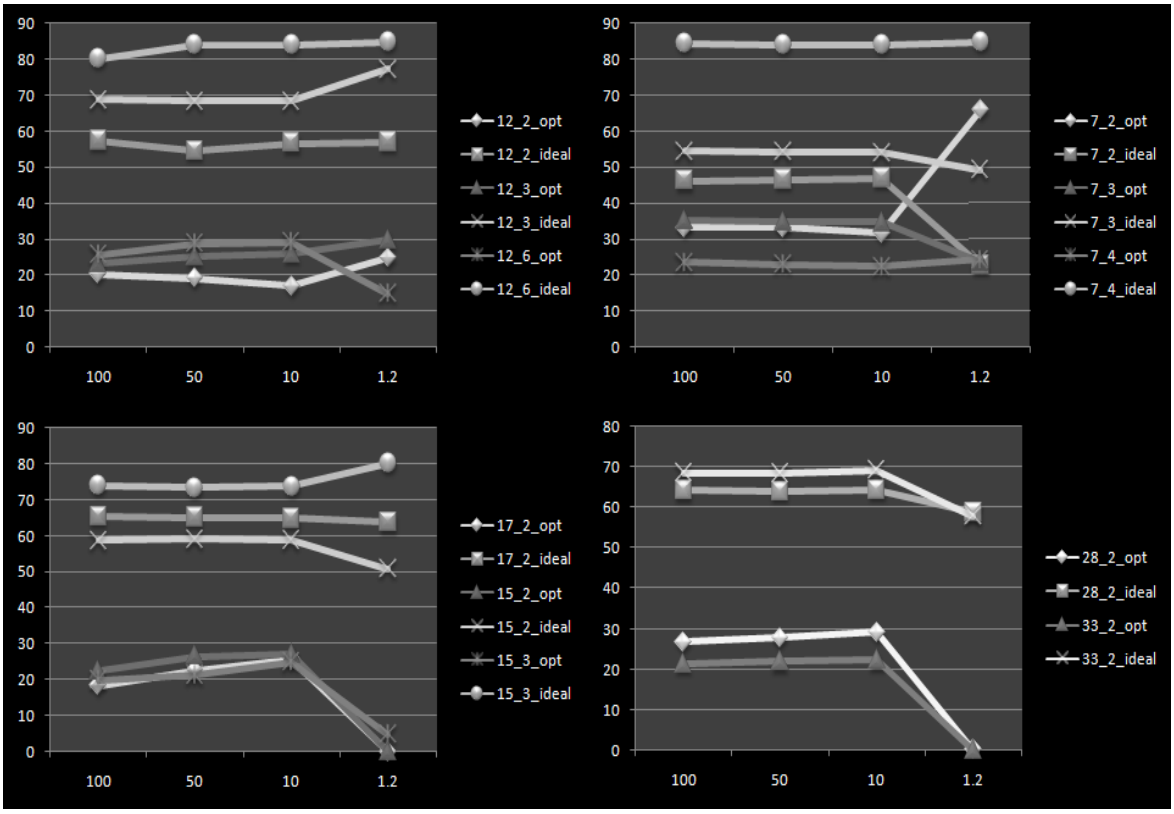


Figure 7.4: Percentage change in error rate for a $\binom{7}{2}$, $\binom{7}{3}$, $\binom{7}{4}$, $\binom{12}{2}$, $\binom{12}{3}$, $\binom{12}{6}$, $\binom{15}{2}$, $\binom{15}{3}$, $\binom{17}{2}$, $\binom{28}{2}$ *and* $\binom{33}{2}$ *multiple PPM system using linear increment mapping as the reference and a PCM error rate of 1 bit in* 10^9 *pulses. The mappings considered are Optimum (OPT) and Ideal (IDEAL).*

APPENDIX B

Tables

---

13 TABLES

Test No.	Section	Test Description	Results Expected	Results Obtained	Any Remarks
1	Interface	Y > X	X cannot be less than Y	X cannot be less than Y	Correct
2	Interface	X = 0	X cannot be zero	X cannot be zero	Correct
3	Interface	Y = 0	Y cannot be zero	Y cannot be zero	Correct
4	Interface	X < 0	X cannot be negative	X cannot be negative	Correct
5	Interface	Y < 0	Y cannot be negative	Y cannot be negative	Correct
6	Data	Data Mapping < 1	Linear, Gray, Random, Predefined	Incorrect Selection	Correct
7	Data	Data Mapping > 5	Linear, Gray, Random, Predefined	Incorrect Selection	Correct
8	Data	Range < 0	Range > 0	Incorrect Data	Correct
9	Data	Range > $2^{PCM\ BITS}$	Range < $2^{PCM\ BITS}$	Incorrect Data	Correct

10	Processing	Algorithm Selected < 1	Complete or 2-Pulse	Incorrect Selection	Correct
11	Processing	Algorithm Selected > 2	Complete or 2-Pulse	Incorrect Selection	Correct
12	Sequences	Display Sequences > 2	Yes or No	Incorrect Selection	Correct
13	Sequences	Display Sequences < 1	Yes or No	Incorrect Selection	Correct
14	Results	Result Handling < 1	Display, Write, Continue, Exit	Incorrect Selection	Correct
15	Results	Result Handling > 9	Display, Write, Continue, Exit	Incorrect Selection	Correct
16	EXIT	Press -1 at any time	Exit	Exit	Correct
17	Data	Data Mapping = 1	Linear Mapping	0 to $2^{PCM\ BITS}$	Correct
18	Data	Data Mapping = 2	Linear Mapping	0 to $2^{PCM\ BITS}$	Correct
19	Data	Data Mapping = 3	Gray Codes	0 to $2^{PCM\ BITS}$	Correct
20	Data	Data Mapping = 4	Random	0 to $2^{PCM\ BITS}$	Correct
21	Data	Data Mapping = 5	Read	0 to $2^{PCM\ BITS}$	Correct
22	Error Type	Error Type < 1	Erasure, False Alarm, Wrong	Incorrect Selection	Correct

			Slot		
23	Error Type	Error Type > 1	Erasure, False Alarm, Wrong Slot	Incorrect Selection	Correct
24	Error Type	Error Type = 1	Erasure	Correct Selection	Correct
25	Error Type	Error Type = 2	False Alarm	Correct Selection	Correct
26	Error Type	Error Type = 3	Wrong Slot	Correct Selection	Correct
27	Processing	Algorithm Selected = 1	Complete	Correct Selection	Correct
28	Processing	Algorithm Selected = 2	2-Pulse	Correct Selection	Correct
29	Data	Data Mapping = 1	Linear	Correct Selection	Correct
30	Data	Data Mapping = 2	Linear	Correct Selection	Correct
31	Data	Data Mapping = 3	Gray	Correct Selection	Correct
32	Data	Data Mapping = 4	Random	Correct Selection	Correct
33	Data	Data Mapping = 5	Read	Correct	Correct

				Selection	
34	Sequences	Display Sequences = 1	Yes	Correct Selection	Correct
35	Sequences	Display Sequences = 2	No	Correct Selection	Correct
36	Results	Result Handling = 1	Display	Correct Selection	Correct
37	Results	Result Handling = 2	Display	Correct Selection	Correct
38	Results	Result Handling = 3	Display	Correct Selection	Correct
39	Results	Result Handling = 4	Display	Correct Selection	Correct
40	Results	Result Handling = 5	Display	Correct Selection	Correct
41	Results	Result Handling = 6	Display	Correct Selection	Correct
42	Results	Result Handling = 7	Display	Correct Selection	Correct
43	Results	Result Handling = 8	Display	Correct Selection	Correct
44	Results	Result Handling = 9	Display	Correct Selection	Correct

45	Results	Result Handling = 1	Save	Correct Selection	Correct
46	Results	Result Handling = 3	Continue	Correct Selection	Correct

Table 3.13: Test Patterns.

	Detect 1st pulse (erasure error in 2nd pulse)	**Detect 2nd pulse**
[1,2]	0	0
[1,3]	$1x[10/64xP_{e1010\mathit{I}}+9/64xP_{e110\mathit{I}}+45/64xP_{e10\mathit{I}}]$	$2x[10/64xP_{e10\mathit{I}}+9/64xP_{e1\mathit{I}}+45/64xP_e]$
[1,4]	$1xP_e$	$3x[10/64xP_{e10\mathit{I}}+9/64xP_{e1\mathit{I}}+45/64xP_e]$
[1,5]	$2xP_e$	$4x[10/64xP_{e10\mathit{I}}+9/64xP_{e1\mathit{I}}+45/64xP_e]$
[1,6]	$1xP_e$	$2x[10/64xP_{e10\mathit{I}}+9/64xP_{e1\mathit{I}}+45/64xP_e]$
[1,7]	$2xP_e$	$2x[10/64xP_{e10\mathit{I}}+9/64xP_{e1\mathit{I}}+45/64xP_e]$
[1,8]	$2xP_e$	$5x[10/64xP_{e10\mathit{I}}+9/64xP_{e1\mathit{I}}+45/64xP_e]$
[1,9]	$3xP_e$	$5x[10/64xP_{e10\mathit{I}}+9/64xP_{e1\mathit{I}}+45/64xP_e]$
[1,10]	$1xP_e$	$4x[10/64xP_{e10\mathit{I}}+9/64xP_{e1\mathit{I}}+45/64xP_e]$
[1,11]	$2xP_e$	$4x[10/64xP_{e10\mathit{I}}+9/64xP_{e1\mathit{I}}+45/64xP_e]$
[1,12]	$2x[11/64xP_{e\mathit{I}1}+53/64xP_e]$	$3x[10/64xP_{e10\mathit{I}}+9/64xP_{e1\mathit{I}}+45/64xP_e]$
[2,3]	$3x[9/64xP_{e101\mathit{I}}+55/64xP_{e1\mathit{I}}]$	$2x[9/64xP_{e10\mathit{I}}+55/64xP_e]$
[2,4]	$2xP_{e10\mathit{I}}$	$2x[9/64xP_{e10\mathit{I}}+55/64xP_e]$
[2,5]	$3xP_e$	$3x[9/64xP_{e10\mathit{I}}+55/64xP_e]$
[2,6]	$3xP_e$	$2x[9/64xP_{e10\mathit{I}}+55/64xP_e]$
[2,7]	$4xP_e$	$4x[9/64xP_{e10\mathit{I}}+55/64xP_e]$
[2,8]	$1xP_e$	$2x[9/64xP_{e10\mathit{I}}+55/64xP_e]$
[2,9]	$2xP_e$	$4x[9/64xP_{e10\mathit{I}}+55/64xP_e]$
[2,10]	$2xP_e$	$4x[9/64xP_{e10\mathit{I}}+55/64xP_e]$
[2,11]	$3xP_e$	$4x[9/64xP_{e10\mathit{I}}+55/64xP_e]$
[2,12]	$2x[9/64xP_{e10\mathit{I}}+55/64xP_e]$	$3x[9/64xP_{e10\mathit{I}}+55/64xP_e]$
[3,4]	$2xP_{e1\mathit{I}}$	$3xP_e$
[3,5]	$4xP_{e10\mathit{I}}$	$3xP_e$
[3,6]	$3xP_e$	$3xP_e$
[3,7]	$1xP_e$	$4xP_e$
[3,8]	0	$2xP_e$
[3,9]	$2xP_e$	$3xP_e$
[3,10]	$1xP_e$	$3xP_e$
[3,11]	$2xP_e$	$4xP_e$
[3,12]	$1x[11/64xP_{e\mathit{I}1}+53/64xP_e]$	$3xP_e$
[4,5]	$4xP_{e1\mathit{I}}$	$2xP_e$
[4,6]	$5xP_{e10\mathit{I}}$	$4xP_e$
[4,7]	$1xP_e$	$3xP_e$
[4,8]	$2xP_e$	$3xP_e$
[4,9]	$2xP_e$	$4xP_e$

[4,10]	$3xP_e$	$2xP_e$
[4,11]	0	$5xP_e$
[4,12]	$1x[11/64xP_{e\boldsymbol{1}1}+53/64xP_e]$	$4xP_e$
[5,6]	$1xP_{e1\boldsymbol{1}}$	0
[5,7]	$2xP_{e10\boldsymbol{1}}$	$2xP_e$
[5,8]	$2xP_e$	$1xP_e$
[5,9]	$3xP_e$	$3xP_e$
[5,10]	$1xP_e$	$3xP_e$
[5,11]	$2xP_e$	$3xP_e$
[5,12]	$1x[11/64xP_{e\boldsymbol{1}1}+53/64xP_e]$	$2xP_e$
[6,7]	$3xP_{e1\boldsymbol{1}}$	$2xP_e$
[6,8]	$1xP_{e10\boldsymbol{1}}$	$3xP_e$
[6,9]	$2xP_e$	$3xP_e$
[6,10]	$3xP_e$	$3xP_e$
[6,11]	$4xP_e$	$3xP_e$
[6,12]	$2x[11/64xP_{e\boldsymbol{1}1}+53/64xP_e]$	$2xP_e$
[7,8]	$2xP_{e1\boldsymbol{1}}$	$3xP_e$
[7,9]	$1xP_{e10\boldsymbol{1}}$	$1xP_e$
[7,10]	0	$4xP_e$
[7,11]	$2xP_e$	$4xP_e$
[7,12]	$1x[11/64xP_{e\boldsymbol{1}1}+53/64xP_e]$	$2xP_e$
[8,9]	0	$1xP_e$
[8,10]	$1xP_{e10\boldsymbol{1}}$	$2xP_e$
[8,11]	$1xP_e$	$3xP_e$
[8,12]	$2x[11/64xP_{e\boldsymbol{1}1}+53/64xP_e]$	$2xP_e$
[9,10]	0	$3xP_e$
[9,11]	$1xP_{e10\boldsymbol{1}}$	$2xP_e$
[9,12]	$1x[11/64xP_{e\boldsymbol{1}1}+53/64xP_e]$	0
[10,11]	$3xP_{e1\boldsymbol{1}}$	$2xP_e$
Total	$243.456\ P_{\boldsymbol{e}} + 22.7712\ P_{\boldsymbol{e}1\boldsymbol{1}} + 27.4944\ P_{\boldsymbol{e}10\boldsymbol{1}} + 0.141\ P_{\boldsymbol{e}110\boldsymbol{1}} + 1.891\ P_{\boldsymbol{e}\boldsymbol{1}1} + 0.156\ P_{\boldsymbol{e}1010\boldsymbol{1}}$	

Table 3.15: Full analysis of Erasure Sequences in a 12-2 MPPM system.

[1,2]	0	0	0	0	0	0	
[1,2,3]	0	0	0	0	0	1	1 x [9/64 x $P_{fl11\theta}$ + 55/64 x $P_{fl1\theta}$]
[1,2,4]	0	0	0	0	0	0	0
[1,2,5]	0	0	0	0	0	1	1 x P_f
[1,2,6]	0	0	0	1	0	0	1 x P_f
[1,2,7]	0	0	0	1	0	1	2 x P_f
[1,2,8]	0	0	0	0	0	0	0
[1,2,9]	0	0	0	0	0	1	1 x P_f
[1,2,10]	0	0	0	0	0	0	0
[1,2,11]	0	0	0	0	0	1	1 x P_f
[1,2,12]	0	0	0	0	0	0	0
[1,3]	0	0	0	0	0	1	
[1,2,3]	0	0	0	0	0	1	0
[1,3,4]	0	0	0	0	0	1	0
[1,3,5]	0	0	0	0	1	1	1 x P_f
[1,3,6]	0	0	0	1	0	1	1 x P_f
[1,3,7]	0	0	0	0	0	1	0
[1,3,8]	0	0	0	0	0	1	0
[1,3,9]	0	0	0	0	1	1	1 x P_f
[1,3,10]	0	0	1	0	0	1	1 x P_f
[1,3,11]	0	0	1	0	0	1	1 x P_f
[1,3,12]	0	0	1	0	0	1	1 x P_f
[1,4]	0	0	0	0	1	0	
[1,2,4]	0	0	0	0	0	0	1 x [9/64x$P_{fl11\theta}$+55/64x$P_{fl1\theta}$]
[1,3,4]	0	0	0	0	0	1	2 x P_f
[1,4,5]	0	0	0	0	1	0	0
[1,4,6]	0	0	0	1	1	0	1 x P_f
[1,4,7]	0	0	0	0	0	0	1 x P_f
[1,4,8]	0	0	0	0	1	0	0
[1,4,9]	0	0	0	0	1	0	0
[1,4,10]	0	0	0	0	1	0	0
[1,4,11]	0	0	0	0	0	0	1 x P_f
[1,4,12]	0	0	0	0	1	0	0

[1,5]	0	0	0	0	1	1	
[1,2,5]	0	0	0	0	0	1	1 x [9/64x$P_{f11\theta}$ + 55/64x$P_{f1\theta}$]
[1,3,5]	0	0	0	0	1	1	0
[1,4,5]	0	0	0	0	1	0	1 x P_f
[1,5,6]	0	0	0	1	1	0	2 x $P_{f1\theta}$
[1,5,7]	0	0	0	1	1	1	1 x P_f
[1,5,8]	0	0	0	0	1	0	1 x P_f
[1,5,9]	0	0	0	0	1	1	0
[1,5,10]	0	0	1	0	1	0	2 x P_f
[1,5,11]	0	0	1	0	1	1	1 x P_f
[1,5,12]	0	0	1	0	1	0	2 x P_f
[1,6]	0	0	0	1	0	0	
[1,2,6]	0	0	0	1	0	0	0
[1,3,6]	0	0	0	1	0	1	1 x P_f
[1,4,6]	0	0	0	1	1	0	1 x P_f
[1,5,6]	0	0	0	1	1	0	1 x P_f
[1,6,7]	0	0	0	1	0	1	1 x $P_{f1\theta}$
[1,6,8]	0	0	0	1	1	0	1 x P_f
[1,6,9]	0	0	0	1	1	1	2 x P_f
[1,6,10]	0	0	0	0	0	0	1 x P_f
[1,6,11]	0	0	0	0	0	1	2 x P_f
[1,6,12]	0	0	0	0	1	0	2 x P_f
[1,7]	0	0	0	1	0	1	
[1,2,7]	0	0	0	1	0	1	0
[1,3,7]	0	0	0	0	0	1	1 x P_f
[1,4,7]	0	0	0	0	0	0	2 x P_f
[1,5,7]	0	0	0	1	1	1	1 x P_f
[1,6,7]	0	0	0	1	0	1	0
[1,7,8]	0	0	0	1	1	1	1 x $P_{f1\theta}$
[1,7,9]	0	0	0	1	0	1	0
[1,7,10]	0	0	0	1	0	1	0
[1,7,11]	0	0	0	1	0	1	0
[1,7,12]	0	0	0	1	1	1	1 x P_f

[1,8]	0	0	0	1	1	0	
[1,2,8]	0	0	0	0	0	0	2 x [9/64x$P_{f11\theta}$ + 55/64x$P_{f1\theta}$]
[1,3,8]	0	0	0	0	0	1	3 x P_f
[1,4,8]	0	0	0	0	1	0	1 x P_f
[1,5,8]	0	0	0	0	1	0	1 x P_f
[1,6,8]	0	0	0	1	1	0	0
[1,7,8]	0	0	0	1	1	1	1 x P_f
[1,8,9]	0	0	0	1	1	0	0
[1,8,10]	0	0	1	0	0	0	3 x P_f
[1,8,11]	0	0	1	0	1	0	2 x P_f
[1,8,12]	0	0	1	0	1	0	2 x P_f
[1,9]	0	0	0	1	1	1	
[1,2,9]	0	0	0	0	0	1	2 x [9/64x$P_{f11\theta}$ + 55/64x$P_{f1\theta}$]
[1,3,9]	0	0	0	0	1	1	1 x P_f
[1,4,9]	0	0	0	0	1	0	2 x P_f
[1,5,9]	0	0	0	0	1	1	1 x P_f
[1,6,9]	0	0	0	1	1	1	0
[1,7,9]	0	0	0	1	0	1	1 x P_f
[1,8,9]	0	0	0	1	1	0	1 x P_f
[1,9,10]	0	0	1	1	0	0	3 x $P_{f1\theta}$
[1,9,11]	0	0	1	1	0	1	2 x P_f
[1,9,12]	0	0	1	1	1	0	2 x Pf
[1,10]	0	0	1	0	0	0	
[1,2,10]	0	0	0	0	0	0	1 x [9/64x$P_{f11\theta}$ + 55/64x$P_{f1\theta}$]
[1,3,10]	0	0	1	0	0	1	1 x P_f
[1,4,10]	0	0	0	0	1	0	2 x P_f
[1,5,10]	0	0	1	0	1	0	1 x P_f
[1,6,10]	0	0	0	0	0	0	1 x P_f
[1,7,10]	0	0	0	1	0	1	3 x P_f
[1,8,10]	0	0	1	0	0	0	0
[1,9,10]	0	0	1	1	0	0	1 x P_f
[1,10,11]	0	0	1	0	0	1	1 x $P_{f1\theta}$
[1,10,12]	0	0	1	0	1	0	1 x P_f

[1,11]	0	0	1	0	0	1	
[1,2,11]	0	0	0	0	0	1	1 x [9/64x$P_{fl1\theta}$ + 55/64x$P_{fl\theta}$]
[1,3,11]	0	0	1	0	0	1	0
[1,4,11]	0	0	0	0	0	0	2 x P_f
[1,5,11]	0	0	1	0	1	1	1 x P_f
[1,6,11]	0	0	0	0	0	1	1 x P_f
[1,7,11]	0	0	0	1	0	1	2 x P_f
[1,8,11]	0	0	1	0	1	0	2 x P_f
[1,9,11]	0	0	1	1	0	1	1 x P_f
[1,10,11]	0	0	1	0	0	1	0
[1,11,12]	0	0	1	0	1	1	2 x [11/64x$P_{fl\theta l}$ + 53/64x$P_{fl\theta}$]
[1,12]	0	0	1	0	1	0	
[1,2,12]	0	0	0	0	0	0	2 x [9/64x$P_{fl1\theta}$ + 55/64x$P_{fl\theta}$]
[1,3,12]	0	0	1	0	0	1	2 x P_f
[1,4,12]	0	0	0	0	1	0	1 x P_f
[1,5,12]	0	0	1	0	1	0	0
[1,6,12]	0	0	0	0	1	0	1 x P_f
[1,7,12]	0	0	0	1	1	1	3 x P_f
[1,8,12]	0	0	1	0	1	0	0
[1,9,12]	0	0	1	1	1	0	1 x P_f
[1,10,12]	0	0	1	0	1	0	1 x Pf
[1,11,12]	0	0	1	0	1	1	2 x P_f
[2,3]	0	0	1	0	1	1	
[1,2,3]	0	0	0	0	0	1	2 x [9/64x$P_{fl\theta l}$ + 55/64xP_f]
[2,3,4]	0	0	1	1	0	1	2 x $P_{fl1\theta}$
[2,3,5]	0	0	1	1	1	1	1 x P_f
[2,3,6]	0	0	1	1	1	1	1 x P_f
[2,3,7]	0	0	0	1	0	1	0
[2,3,8]	0	1	1	0	0	1	2 x P_f
[2,3,9]	0	1	1	0	1	1	1 x P_f
[2,3,10]	0	1	1	0	1	1	1 x P_f
[2,3,11]	0	1	1	0	1	1	1 x P_f
[2,3,12]	0	1	1	1	0	1	3 x P_f

[2,4]	0	0	1	1	0	0	
[1,2,4]	0	0	0	0	0	0	2 x [9/64x$P_{f1\theta1}$ + 55/64xP_f]
[2,3,4]	0	0	1	1	0	1	1 x $P_{f1\theta1}$
[2,4,5]	0	0	1	1	0	0	0
[2,4,6]	0	0	1	1	1	0	1 x P_f
[2,4,7]	0	0	1	1	0	0	0
[2,4,8]	0	0	0	0	0	0	2 x P_f
[2,4,9]	0	0	0	0	0	0	2 x P_f
[2,4,10]	0	0	0	0	1	0	3 x P_f
[2,4,11]	0	0	0	1	0	0	1 x P_f
[2,4,12]	0	0	0	1	0	0	1 x P_f
[2,5]	0	0	1	1	0	1	
[1,2,5]	0	0	0	0	0	1	2 x [9/64x$P_{f1\theta1}$ + 55/64xP_f]
[2,3,5]	0	0	1	1	1	1	1 x $P_{f1\theta}$
[2,4,5]	0	0	1	1	0	0	1 x P_f
[2,5,6]	0	0	1	1	1	0	2 x $P_{f1\theta}$
[2,5,7]	0	0	1	1	1	1	1 x P_f
[2,5,8]	0	0	1	0	0	0	2 x P_f
[2,5,9]	0	0	1	0	0	1	1 x P_f
[2,5,10]	0	0	1	0	1	0	3 x P_f
[2,5,11]	0	0	1	0	1	1	2 x P_f
[2,5,12]	0	0	1	1	0	0	1 x P_f
[2,6]	0	0	1	1	1	0	
[1,2,6]	0	0	0	1	0	0	2 x [9/64x$P_{f1\theta1}$ + 55/64xP_f]
[2,3,6]	0	0	1	1	1	1	1 x $P_{f1\theta}$
[2,4,6]	0	0	1	1	1	0	0
[2,5,6]	0	0	1	1	1	0	0
[2,6,7]	0	0	1	1	1	1	1 x $P_{f1\theta}$
[2,6,8]	0	0	1	1	1	0	0
[2,6,9]	0	0	1	1	1	1	1 x P_f
[2,6,10]	0	1	0	0	1	0	3 x P_f
[2,6,11]	0	1	0	0	1	1	4 x P_f
[2,6,12]	0	1	0	1	1	0	2 x P_f

[2,7]	0	0	1	1	1	1	
[1,2,7]	0	0	0	1	0	1	2 x [9/64x$P_{f1\theta1}$ + 55/64xP_f]
[2,3,7]	0	0	0	1	0	1	1 x $P_{f1\theta}$
[2,4,7]	0	0	1	1	0	0	2 x P_f
[2,5,7]	0	0	1	1	1	1	0
[2,6,7]	0	0	1	1	1	1	0
[2,7,8]	0	1	0	0	1	1	3 x $P_{f1\theta}$
[2,7,9]	0	1	0	1	0	1	3 x P_f
[2,7,10]	0	1	0	1	1	1	2 x P_f
[2,7,11]	0	1	0	1	1	1	2 x P_f
[2,7,12]	0	1	0	1	1	1	2 x [11/64xP_f + 53/64xP_f]
[2,8]	0	1	0	0	0	0	
[1,2,8]	0	0	0	0	0	0	1 x [9/64x$P_{f1\theta1}$ + 55/64xP_f]
[2,3,8]	0	1	1	0	0	1	2 x $P_{f1\theta}$
[2,4,8]	0	0	0	0	0	0	1 x P_f
[2,5,8]	0	0	1	0	0	0	2 x P_f
[2,6,8]	0	0	1	1	1	0	4 x P_f
[2,7,8]	0	1	0	0	1	1	2 x P_f
[2,8,9]	0	1	0	0	0	0	0
[2,8,10]	0	1	0	0	0	0	0
[2,8,11]	0	1	0	0	1	0	1 x P_f
[2,8,12]	0	1	0	0	0	0	0
[2,9]	0	1	0	0	0	1	
[1,2,9]	0	0	0	0	0	1	1 x [9/64x$P_{f1\theta1}$ + 55/64xP_f]
[2,3,9]	0	1	1	0	1	1	2 x $P_{f1\theta}$
[2,4,9]	0	0	0	0	0	0	2 x P_f
[2,5,9]	0	0	1	0	0	1	2 x P_f
[2,6,9]	0	0	1	1	1	1	4 x P_f
[2,7,9]	0	1	0	1	0	1	1 x P_f
[2,8,9]	0	1	0	0	0	0	1 x P_f
[2,9,10]	0	1	0	0	0	0	1 x $P_{f1\theta}$
[2,9,11]	0	1	0	0	0	1	0
[2,9,12]	0	1	0	1	0	0	2 x P_f

[2,10]	0	1	0	0	1	0	
[1,2,10]	0	0	0	0	0	0	2 x [9/64x$P_{f1\theta1}$ + 55/64xP_f]
[2,3,10]	0	1	1	0	1	1	2 x $P_{f1\theta}$
[2,4,10]	0	0	0	0	1	0	1 x P_f
[2,5,10]	0	0	1	0	1	0	2 x P_f
[2,6,10]	0	1	0	0	1	0	0
[2,7,10]	0	1	0	1	1	1	2 x P_f
[2,8,10]	0	1	0	0	0	0	1 x P_f
[2,9,10]	0	1	0	0	0	0	1 x P_f
[2,10,11]	0	1	0	0	1	1	1 x $P_{f1\theta}$
[2,10,12]	0	1	0	1	1	0	2 x P_f
[2,11]	0	1	0	0	1	1	
[1,2,11]	0	0	0	0	0	1	2 x [9/64x$P_{f1\theta1}$ + 55/64xP_f]
[2,3,11]	0	1	1	0	1	1	1 x $P_{f1\theta}$
[2,4,11]	0	0	0	1	0	0	4 x P_f
[2,5,11]	0	0	1	0	1	1	2 x P_f
[2,6,11]	0	1	0	0	1	1	0
[2,7,11]	0	1	0	1	1	1	1 x P_f
[2,8,11]	0	1	0	0	1	0	1 x P_f
[2,9,11]	0	1	0	0	0	1	1 x P_f
[2,10,11]	0	1	0	0	1	1	0
[2,11,12]	0	1	0	1	1	1	3 x [11/64x$P_{f1\theta1}$ + 53/64x$P_{f1\theta}$]
[2,12]	0	1	0	1	0	0	
[1,2,12]	0	0	0	0	0	0	2 x [9/64x$P_{f1\theta1}$ + 55/64xP_f]
[2,3,12]	0	1	1	1	0	1	2 x $P_{f1\theta}$
[2,4,12]	0	0	0	1	0	0	1 x P_f
[2,5,12]	0	0	1	1	0	0	2 x P_f
[2,6,12]	0	1	0	1	1	0	1 x P_f
[2,7,12]	0	1	0	1	1	1	2 x P_f
[2,8,12]	0	1	0	0	0	0	1 x P_f
[2,9,12]	0	1	0	1	0	0	0
[2,10,12]	0	1	0	1	1	0	2 x P_f
[2,11,12]	0	1	0	1	1	1	3 x P_f

[3,4]	0	1	0	1	0	1	
[1,3,4]	0	0	0	0	0	1	2 x [9/64xP_{fl0} + 55/64xP_f]
[2,3,4]	0	0	1	1	0	1	2 x P_f
[3,4,5]	0	1	0	1	1	0	2 x P_{fl1}
[3,4,6]	0	1	0	1	1	1	1 x P_f
[3,4,7]	0	1	0	0	0	0	2 x P_f
[3,4,8]	0	1	0	0	0	1	1 x P_f
[3,4,9]	0	1	0	0	1	0	3 x P_f
[3,4,10]	0	1	0	0	1	1	2 x P_f
[3,4,11]	0	1	0	1	0	0	1 x P_f
[3,4,12]	0	1	0	1	0	1	0
[3,5]	0	1	0	1	1	0	
[1,3,5]	0	0	0	0	1	1	3 x [9/64xP_{fl0} + 55/64xP_f]
[2,3,5]	0	0	1	1	1	1	3 x P_f
[3,4,5]	0	1	0	1	1	0	0
[3,5,6]	0	1	0	1	1	0	0
[3,5,7]	0	1	0	1	1	0	0
[3,5,8]	0	1	1	0	0	0	3 x P_f
[3,5,9]	0	1	1	0	1	0	2 x P_f
[3,5,10]	0	1	1	0	1	0	2 x P_f
[3,5,11]	0	1	1	1	1	0	1 x P_f
[3,5,12]	0	1	1	1	0	0	2 x [11/64xP_f + 53/64xP_f]
[3,6]	0	1	0	1	1	1	
[1,3,6]	0	0	0	1	0	1	2 x [9/64xP_{fl0} + 55/64xP_f]
[2,3,6]	0	0	1	1	1	1	2 x P_f
[3,4,6]	0	1	0	1	1	1	0
[3,5,6]	0	1	0	1	1	0	1 x P_f
[3,6,7]	0	1	1	1	0	1	2 x P_{fl0}
[3,6,8]	0	1	1	1	1	1	1 x P_f
[3,6,9]	0	1	1	1	1	1	1 x P_f
[3,6,10]	0	1	0	0	1	1	1 x P_f
[3,6,11]	0	1	0	1	0	1	1 x P_f
[3,6,12]	0	1	0	1	1	1	0

[3,7]	0	1	1	0	0	0	
[1,3,7]	0	0	0	0	0	1	3 x [9/64x$P_{f1\boldsymbol{0}}$ + 55/64xP_f]
[2,3,7]	0	0	0	1	0	1	3 x $P_{f\boldsymbol{0}1}$
[3,4,7]	0	1	0	0	0	0	1 x $P_{f1\boldsymbol{0}}$
[3,5,7]	0	1	0	1	1	0	3 x P_f
[3,6,7]	0	1	1	1	0	1	2 x P_f
[3,7,8]	0	1	1	0	0	1	1 x $P_{f1\boldsymbol{0}}$
[3,7,9]	0	1	1	0	0	0	0
[3,7,10]	0	1	1	0	0	1	1 x P_f
[3,7,11]	0	1	1	1	0	0	1 x P_f
[3,7,12]	0	1	1	1	0	1	2 x [11/64xP_f + 53/64xP_f]
[3,8]	0	1	1	0	0	1	
[1,3,8]	0	0	0	0	0	1	2 x [9/64x$P_{f1\boldsymbol{0}}$ + 55/64xP_f]
[2,3,8]	0	1	1	0	0	1	0
[3,4,8]	0	1	0	0	0	1	1 x $P_{f1\boldsymbol{0}}$
[3,5,8]	0	1	1	0	0	0	1 x P_f
[3,6,8]	0	1	1	1	1	1	2 x P_f
[3,7,8]	0	1	1	0	0	1	0
[3,8,9]	0	1	1	0	0	0	1 x $P_{f1\boldsymbol{0}}$
[3,8,10]	0	1	1	0	0	1	0
[3,8,11]	0	1	1	0	0	0	1 x P_f
[3,8,12]	0	1	1	0	0	1	0
[3,9]	0	1	1	0	1	0	
[1,3,9]	0	0	0	0	1	1	3 x [9/64x$P_{f1\boldsymbol{0}}$ + 55/64xP_f]
[2,3,9]	0	1	1	0	1	1	1 x P_f
[3,4,9]	0	1	0	0	1	0	1 x $P_{f1\boldsymbol{0}}$
[3,5,9]	0	1	1	0	1	0	0
[3,6,9]	0	1	1	1	1	1	2 x P_f
[3,7,9]	0	1	1	0	0	0	1 x P_f
[3,8,9]	0	1	1	0	0	0	1 x P_f
[3,9,10]	0	1	1	0	1	0	0
[3,9,11]	0	1	1	1	0	0	2 x P_f
[3,9,12]	0	1	1	1	1	0	1 x [11/64xP_f + 53/64xP_f]

[3,10]	0	1	1	0	1	1	
[1,3,10]	0	0	1	0	0	1	2 x [9/64x$P_{f1\emptyset}$ + 55/64xP_f]
[2,3,10]	0	1	1	0	1	1	0
[3,4,10]	0	1	0	0	1	1	1 x $P_{f1\emptyset}$
[3,5,10]	0	1	1	0	1	0	1 x P_f
[3,6,10]	0	1	0	0	1	1	1 x P_f
[3,7,10]	0	1	1	0	0	1	1 x P_f
[3,8,10]	0	1	1	0	0	1	1 x P_f
[3,9,10]	0	1	1	0	1	0	1 x P_f
[3,10,11]	0	1	1	1	1	1	1 x $P_{f1\emptyset}$
[3,10,12]	0	1	1	1	1	1	2 x P_f
[3,11]	0	1	1	1	0	0	
[1,3,11]	0	0	1	0	0	1	3 x [9/64x$P_{f1\emptyset}$ + 55/64xP_f]
[2,3,11]	0	1	1	0	1	1	3 x P_f
[3,4,11]	0	1	0	1	0	0	1 x $P_{f1\emptyset}$
[3,5,11]	0	1	1	1	1	0	1 x P_f
[3,6,11]	0	1	0	1	0	1	2 x P_f
[3,7,11]	0	1	1	1	0	0	0
[3,8,11]	0	1	1	0	0	0	1 x P_f
[3,9,11]	0	1	1	1	0	0	0
[3,10,11]	0	1	1	1	1	1	2 x P_f
[3,11,12]	0	1	1	1	0	1	1 x [11/64x$P_{f1\emptyset1}$ + 53/64x$P_{f1\emptyset}$]
[3,12]	0	1	1	1	0	1	
[1,3,12]	0	0	1	0	0	1	2 x [9/64x$P_{f1\emptyset}$ + 55/64xP_f]
[2,3,12]	0	1	1	1	0	1	0
[3,4,12]	0	1	0	1	0	1	1 x $P_{f1\emptyset}$
[3,5,12]	0	1	1	1	0	0	1 x P_f
[3,6,12]	0	1	0	1	1	1	2 x P_f
[3,7,12]	0	1	1	1	0	1	0
[3,8,12]	0	1	1	0	0	1	1 x P_f
[3,9,12]	0	1	1	1	1	0	2 x P_f
[3,10,12]	0	1	1	1	1	1	2 x P_f
[3,11,12]	0	1	1	1	0	1	1 x P_f

[4,5]	0	1	1	1	1	0	
[1,4,5]	0	0	0	0	1	0	3 x [9/64x$P_{f1\theta}$ + 55/64xP_f]
[2,4,5]	0	0	1	1	0	0	2 x P_f
[3,4,5]	0	1	0	1	1	0	1 x P_f
[4,5,6]	0	1	1	1	1	0	0
[4,5,7]	1	0	0	1	1	0	3 x P_f
[4,5,8]	1	0	1	0	0	0	4 x P_f
[4,5,9]	1	0	1	0	1	0	3 x P_f
[4,5,10]	1	0	1	0	1	0	3 x P_f
[4,5,11]	1	0	1	1	1	0	2 x P_f
[4,5,12]	1	0	1	1	0	0	3 x P_f
[4,6]	0	1	1	1	1	1	
[1,4,6]	0	0	0	1	1	0	3 x [9/64x$P_{f1\theta}$ + 55/64xP_f]
[2,4,6]	0	0	1	1	1	0	2 x P_f
[3,4,6]	0	1	0	1	1	1	1 x P_f
[4,5,6]	0	1	1	1	1	0	1 x $P_{f1\theta1}$
[4,6,7]	1	0	1	1	0	1	3 x $P_{f1\theta}$
[4,6,8]	1	0	1	1	1	1	2 x P_f
[4,6,9]	1	0	1	1	1	1	2 x P_f
[4,6,10]	1	1	0	0	1	1	3 x P_f
[4,6,11]	1	1	0	1	0	1	3 x P_f
[4,6,12]	1	1	0	1	1	1	2 x P_f
[4,7]	1	0	0	0	0	0	
[1,4,7]	0	0	0	0	0	0	1 x [9/64x$P_{f1\theta}$ + 55/64xP_f]
[2,4,7]	0	0	1	1	0	0	3 x P_f
[3,4,7]	0	1	0	0	0	0	2 x P_f
[4,5,7]	1	0	0	1	1	0	2 x $P_{f1\theta}$
[4,6,7]	1	0	1	1	0	1	3 x P_f
[4,7,8]	1	0	0	0	0	1	1 x $P_{f1\theta}$
[4,7,9]	1	0	0	0	0	0	0
[4,7,10]	1	0	0	0	0	1	1 x P_f
[4,7,11]	1	0	0	1	0	0	1 x P_f
[4,7,12]	1	0	0	1	0	1	2 x P_f

[4,8]	1	0	0	0	0	1	
[1,4,8]	0	0	0	0	1	0	3 x [9/64x$P_{f1\theta}$ + 55/64xP_f]
[2,4,8]	0	0	0	0	0	0	2 x P_f
[3,4,8]	0	1	0	0	0	1	2 x P_f
[4,5,8]	1	0	1	0	0	0	2 x $P_{f1\theta}$
[4,6,8]	1	0	1	1	1	1	3 x P_f
[4,7,8]	1	0	0	0	0	1	0
[4,8,9]	1	0	0	0	0	0	1 x $P_{f1\theta}$
[4,8,10]	1	0	0	0	0	1	0
[4,8,11]	1	0	0	0	0	0	1 x P_f
[4,8,12]	1	0	0	0	0	1	0
[4,9]	1	0	0	0	1	0	
[1,4,9]	0	0	0	0	1	0	1 x [9/64x$P_{f1\theta}$ + 55/64xP_f]
[2,4,9]	0	0	0	0	0	0	2 x P_f
[3,4,9]	0	1	0	0	1	0	2 x P_f
[4,5,9]	1	0	1	0	1	0	1 x $P_{f1\theta}$
[4,6,9]	1	0	1	1	1	1	3 x P_f
[4,7,9]	1	0	0	0	0	0	1 x P_f
[4,8,9]	1	0	0	0	0	0	1 x P_f
[4,9,10]	1	0	0	0	1	0	0
[4,9,11]	1	0	0	1	0	0	2 x P_f
[4,9,12]	1	0	0	1	1	0	1 x P_f
[4,10]	1	0	0	0	1	1	
[1,4,10]	0	0	0	0	1	0	2 x [9/64x$P_{f1\theta}$ + 55/64xP_f]
[2,4,10]	0	0	0	0	1	0	2 x P_f
[3,4,10]	0	1	0	0	1	1	2 x P_f
[4,5,10]	1	0	1	0	1	0	2 x $P_{f1\theta}$
[4,6,10]	1	1	0	0	1	1	1 x P_f
[4,7,10]	1	0	0	0	0	1	1 x P_f
[4,8,10]	1	0	0	0	0	1	1 x P_f
[4,9,10]	1	0	0	0	1	0	1 x P_f
[4,10,11]	1	0	0	1	1	1	1 x $P_{f1\theta}$
[4,10,12]	1	0	0	1	1	1	2 x P_f

[4,11]	1	0	0	1	0	0	
[1,4,11]	0	0	0	0	0	0	2 x [9/64x$P_{f1\boldsymbol{0}}$ + 55/64xP_f]
[2,4,11]	0	0	0	1	0	0	1 x P_f
[3,4,11]	0	1	0	1	0	0	2 x P_f
[4,5,11]	1	0	1	1	1	0	2 x $P_{f1\boldsymbol{0}}$
[4,6,11]	1	1	0	1	0	1	2 x P_f
[4,7,11]	1	0	0	1	0	0	0
[4,8,11]	1	0	0	0	0	0	1 x P_f
[4,9,11]	1	0	0	1	0	0	0
[4,10,11]	1	0	0	1	1	1	2 x P_{f01}
[4,11,12]	1	0	0	1	0	1	1 x [11/64x$P_{f1\boldsymbol{0}1}$ + 53/64x$P_{f1\boldsymbol{0}}$]
[4,12]	1	0	0	1	0	1	
[1,4,12]	0	0	0	0	1	0	4 x [9/64x$P_{f1\boldsymbol{0}}$ + 55/64xP_f]
[2,4,12]	0	0	0	1	0	0	2 x P_f
[3,4,12]	0	1	0	1	0	1	2 x P_f
[4,5,12]	1	0	1	1	0	0	2 x $P_{f1\boldsymbol{0}}$
[4,6,12]	1	1	0	1	1	1	2 x P_f
[4,7,12]	1	0	0	1	0	1	0
[4,8,12]	1	0	0	0	0	1	1 x P_f
[4,9,12]	1	0	0	1	1	0	2 x P_f
[4,10,12]	1	0	0	1	1	1	2 x P_f
[4,11,12]	1	0	0	1	0	1	1 x P_f
[5,6]	1	0	0	1	1	0	
[1,5,6]	0	0	0	1	1	0	1 x [9/64x$P_{f1\boldsymbol{0}}$ + 55/64xP_f]
[2,5,6]	0	0	1	1	1	0	2 x P_f
[3,5,6]	0	1	0	1	1	0	2 x P_f
[4,5,6]	0	1	1	1	1	0	3 x P_f
[5,6,7]	1	0	0	1	1	1	1 x $P_{f11\boldsymbol{0}}$
[5,6,8]	1	0	0	1	1	0	1 x P_f
[5,6,9]	1	0	0	1	1	1	***2*** x P_f
[5,6,10]	1	0	0	0	1	0	1 x P_f
[5,6,11]	1	0	0	0	1	1	2 x P_f
[5,6,12]	1	0	0	1	1	0	0

[5,7]	1	0	0	1	1	1	
[1,5,7]	0	0	0	1	1	1	1 x [9/64xP_{fl0} + 55/64xP_f]
[2,5,7]	0	0	1	1	1	1	2 x P_f
[3,5,7]	0	1	0	1	1	0	3 x P_f
[4,5,7]	1	0	0	1	1	0	1 x P_f
[5,6,7]	1	0	0	1	1	1	0
[5,7,8]	1	0	0	0	1	1	1 x P_{fl0}
[5,7,9]	1	0	0	1	0	1	1 x P_f
[5,7,10]	1	0	0	1	1	1	0
[5,7,11]	1	0	0	1	1	1	0
[5,7,12]	1	0	0	1	1	1	0
[5,8]	1	0	1	0	0	0	
[1,5,8]	0	0	0	0	1	0	3 x [9/64xP_{fl0} + 55/64xP_f]
[2,5,8]	0	0	1	0	0	0	1 x P_f
[3,5,8]	0	1	1	0	0	0	2 x P_f
[4,5,8]	1	0	1	0	0	0	0
[5,6,8]	1	0	0	1	1	0	2 x P_{fl0}
[5,7,8]	1	0	0	0	1	1	3 x P_f
[5,8,9]	1	0	1	0	0	0	0
[5,8,10]	1	0	1	0	0	0	0
[5,8,11]	1	0	1	0	1	0	1 x P_f
[5,8,12]	1	0	1	0	0	0	0
[5,9]	1	0	1	0	0	1	
[1,5,9]	0	0	0	0	1	1	3 x [9/64xP_{fl0} + 55/64xP_f]
[2,5,9]	0	0	1	0	0	1	1 x P_f
[3,5,9]	0	1	1	0	1	0	4 x P_f
[4,5,9]	1	0	1	0	1	0	2 x P_f
[5,6,9]	1	0	0	1	1	1	2 x P_{fl0}
[5,7,9]	1	0	0	1	0	1	2 x P_f
[5,8,9]	1	0	1	0	0	0	1 x P_f
[5,9,10]	1	0	1	0	0	0	1 x P_{fl0}
[5,9,11]	1	0	1	0	0	1	0
[5,9,12]	1	0	1	1	0	0	2 x P_f

[5,10]	1	0	1	0	1	0	
[1,5,10]	0	0	1	0	1	0	1 x [9/64x$P_{f1\theta}$ + 55/64xP_f]
[2,5,10]	0	0	1	0	1	0	1 x P_f
[3,5,10]	0	1	1	0	1	0	2 x P_f
[4,5,10]	1	0	1	0	1	0	0
[5,6,10]	1	0	0	0	1	0	1 x $P_{f1\theta}$
[5,7,10]	1	0	0	1	1	1	3 x P_f
[5,8,10]	1	0	1	0	0	0	1 x P_f
[5,9,10]	1	0	1	0	0	0	1 x P_f
[5,10,11]	1	0	1	0	1	1	1 x $P_{f1\theta}$
[5,10,12]	1	0	1	1	1	0	2 x P_f
[5,11]	1	0	1	0	1	1	
[1,5,11]	0	0	1	0	1	1	1 x [9/64x$P_{f1\theta}$ + 55/64xP_f]
[2,5,11]	0	0	1	0	1	1	1 x P_f
[3,5,11]	0	1	1	1	1	0	4 x P_f
[4,5,11]	1	0	1	1	1	0	2 x P_f
[5,6,11]	1	0	0	0	1	1	1 x $P_{f1\theta}$
[5,7,11]	1	0	0	1	1	1	2 x P_f
[5,8,11]	1	0	1	0	1	0	1 x P_f
[5,9,11]	1	0	1	0	0	1	1 x P_f
[5,10,11]	1	0	1	0	1	1	0
[5,11,12]	1	0	1	1	1	1	3 x [11/64x$P_{f1\theta1}$ + 53/64x$P_{f1\theta}$]
[5,12]	1	0	1	1	0	0	
[1,5,12]	0	0	1	0	1	0	3 x [9/64x$P_{f1\theta}$ + 55/64xP_f]
[2,5,12]	0	0	1	1	0	0	1 x P_f
[3,5,12]	0	1	1	1	0	0	2 x P_f
[4,5,12]	1	0	1	1	0	0	0
[5,6,12]	1	0	0	1	1	0	2 x $P_{f1\theta}$
[5,7,12]	1	0	0	1	1	1	3 x P_f
[5,8,12]	1	0	1	0	0	0	1 x P_f
[5,9,12]	1	0	1	1	0	0	0
[5,10,12]	1	0	1	1	1	0	2 x P_f
[5,11,12]	1	0	1	1	1	1	3 x P_f

[6,7]	1	0	1	1	0	1	
[1,6,7]	0	0	0	1	0	1	2 x [9/64xP_{fl0} + 55/64xP_f]
[2,6,7]	0	0	1	1	1	1	2 x P_f
[3,6,7]	0	1	1	1	0	1	2 x P_f
[4,6,7]	1	0	1	1	0	1	0
[5,6,7]	1	0	0	1	1	1	2 x P_f
[6,7,8]	1	0	1	1	1	1	1 x P_{fl1}
[6,7,9]	1	0	1	1	0	1	0
[6,7,10]	1	1	0	1	0	1	2 x P_f
[6,7,11]	1	1	0	1	0	1	2 x P_f
[6,7,12]	1	1	0	1	1	1	3 x [11/64xP_f + 53/64xP_f]
[6,8]	1	0	1	1	1	0	
[1,6,8]	0	0	0	1	1	0	2 x [9/64xP_{fl0} + 55/64xP_f]
[2,6,8]	0	0	1	1	1	0	1 x P_f
[3,6,8]	0	1	1	1	1	1	3 x P_f
[4,6,8]	1	0	1	1	1	1	1 x P_f
[5,6,8]	1	0	0	1	1	0	0
[6,7,8]	1	0	1	1	1	1	1 x P_{fl0l}
[6,8,9]	1	0	1	1	1	0	0
[6,8,10]	1	1	1	0	0	0	3 x P_f
[6,8,11]	1	1	1	0	1	0	2 x P_f
[6,8,12]	1	1	1	0	1	0	2 x [11/64xP_f + 53/64xP_f]
[6,9]	1	0	1	1	1	1	
[1,6,9]	0	0	0	1	1	1	2 x [9/64xP_{fl0} + 55/64xP_f]
[2,6,9]	0	0	1	1	1	1	1 x P_f
[3,6,9]	0	1	1	1	1	1	2 x P_f
[4,6,9]	1	0	1	1	1	1	0
[5,6,9]	1	0	0	1	1	1	0
[6,7,9]	1	0	1	1	0	1	1 x P_{fl0}
[6,8,9]	1	0	1	1	1	0	1 x P_f
[6,9,10]	1	1	1	1	0	0	3 x P_{fl0}
[6,9,11]	1	1	1	1	0	1	2 x P_f
[6,9,12]	1	1	1	1	1	0	2 x [11/64xP_f + 53/64xP_f]

[6,10]	1	1	0	0	0	0	
[1,6,10]	0	0	0	0	0	0	2 x [9/64x$P_{f1\theta}$ + 55/64xP_f]
[2,6,10]	0	1	0	0	1	0	2 x P_f
[3,6,10]	0	1	0	0	1	1	3 x P_f
[4,6,10]	1	1	0	0	1	1	2 x P_f
[5,6,10]	1	0	0	0	1	0	2 x P_f
[6,7,10]	1	1	0	1	0	1	2 x $P_{f1\theta}$
[6,8,10]	1	1	1	0	0	0	1 x P_f
[6,9,10]	1	1	1	1	0	0	2 x P_f
[6,10,11]	1	1	0	0	0	1	1 x $P_{f1\theta}$
[6,10,12]	1	1	0	0	1	0	1 x [11/64xP_f + 53/64xP_f]
[6,11]	1	1	0	0	0	1	
[1,6,11]	0	0	0	0	0	1	2 x [9/64x$P_{f1\theta}$ + 55/64xP_f]
[2,6,11]	0	1	0	0	1	1	2 x P_f
[3,6,11]	0	1	0	1	0	1	2 x P_f
[4,6,11]	1	1	0	1	0	1	1 x P_f
[5,6,11]	1	0	0	0	1	1	2 x P_f
[6,7,11]	1	1	0	1	0	1	1 x $P_{f1\theta}$
[6,8,11]	1	1	1	0	1	0	3 x P_f
[6,9,11]	1	1	1	1	0	1	2 x P_f
[6,10,11]	1	1	0	0	0	1	0
[6,11,12]	1	1	0	0	1	1	2 x [11/64x$P_{f1\theta1}$ + 53/64x$P_{f1\theta}$]
[6,12]	1	1	0	0	1	0	
[1,6,12]	0	0	0	0	1	0	2 x [9/64x$P_{f1\theta}$ + 55/64xP_f]
[2,6,12]	0	1	0	1	1	0	2 x P_f
[3,6,12]	0	1	0	1	1	1	3 x P_f
[4,6,12]	1	1	0	1	1	1	2 x P_f
[5,6,12]	1	0	0	1	1	0	2 x P_f
[6,7,12]	1	1	0	1	1	1	2 x $P_{f1\theta}$
[6,8,12]	1	1	1	0	1	0	1 x P_f
[6,9,12]	1	1	1	1	1	0	2 x P_f
[6,10,12]	1	1	0	0	1	0	1 x P_f
[6,11,12]	1	1	0	0	1	1	2 x P_f

[7,8]	1	1	0	0	1	1	
[1,7,8]	0	0	0	1	1	1	3 x [9/64x$P_{f1\emptyset}$ + 55/64xP_f]
[2,7,8]	0	1	0	0	1	1	1 x P_f
[3,7,8]	0	1	1	0	0	1	3 x P_f
[4,7,8]	1	0	0	0	0	1	2 x P_f
[5,7,8]	1	0	0	0	1	1	1 x P_f
[6,7,8]	1	0	1	1	1	1	3 x P_f
[7,8,9]	1	1	0	0	0	0	2 x $P_{f11\emptyset}$
[7,8,10]	1	1	0	0	0	1	1 x P_f
[7,8,11]	1	1	0	0	1	0	1 x P_f
[7,8,12]	1	1	0	0	1	1	0
[7,9]	1	1	0	1	0	0	
[1,7,9]	0	0	0	1	0	1	3 x [9/64x$P_{f1\emptyset}$ + 55/64xP_f]
[2,7,9]	0	1	0	1	0	1	2 x P_f
[3,7,9]	0	1	1	0	0	0	3 x P_f
[4,7,9]	1	0	0	0	0	0	2 x P_f
[5,7,9]	1	0	0	1	0	1	2 x P_f
[6,7,9]	1	0	1	1	0	1	3 x P_f
[7,8,9]	1	1	0	0	0	0	1 x $P_{f1\emptyset 1}$
[7,9,10]	1	1	0	1	0	0	0
[7,9,11]	1	1	0	1	0	0	0
[7,9,12]	1	1	0	1	1	0	1 x P_f
[7,10]	1	1	0	1	0	1	
[1,7,10]	0	0	0	1	0	1	2 x [9/64x$P_{f1\emptyset}$ + 55/64xP_f]
[2,7,10]	0	1	0	1	1	1	2 x P_f
[3,7,10]	0	1	1	0	0	1	3 x P_f
[4,7,10]	1	0	0	0	0	1	2 x P_f
[5,7,10]	1	0	0	1	1	1	2 x P_f
[6,7,10]	1	1	0	1	0	1	0
[7,8,10]	1	1	0	0	0	1	1 x $P_{f1\emptyset}$
[7,9,10]	1	1	0	1	0	0	1 x P_f
[7,10,11]	1	1	0	1	1	1	1 x $P_{f1\emptyset}$
[7,10,12]	1	1	0	1	1	1	1 x P_f

[7,11]	1	1	0	1	1	0	
[1,7,11]	0	0	0	1	0	1	4 x [9/64x$P_{f1\emptyset}$ + 55/64xP_f]
[2,7,11]	0	1	0	1	1	1	2 x P_f
[3,7,11]	0	1	1	1	0	0	3 x P_f
[4,7,11]	1	0	0	1	0	0	2 x P_f
[5,7,11]	1	0	0	1	1	1	2 x P_f
[6,7,11]	1	1	0	1	0	1	2 x P_f
[7,8,11]	1	1	0	0	1	0	1 x $P_{f1\emptyset}$
[7,9,11]	1	1	0	1	0	0	1 x P_f
[7,10,11]	1	1	0	1	1	1	1 x P_f
[7,11,12]	1	1	0	1	1	1	1 x [11/64x$P_{f1\emptyset1}$ + 53/64x$P_{f1\emptyset}$]
[7,12]	1	1	0	1	1	1	
[1,7,12]	0	0	0	1	1	1	2 x [9/64x$P_{f1\emptyset}$ + 55/64xP_f]
[2,7,12]	0	1	0	1	1	1	1 x P_f
[3,7,12]	0	1	1	1	0	1	3 x P_f
[4,7,12]	1	0	0	1	0	1	2 x P_f
[5,7,12]	1	0	0	1	1	1	1 x P_f
[6,7,12]	1	1	0	1	1	1	0
[7,8,12]	1	1	0	0	1	1	1 x $P_{f1\emptyset}$
[7,9,12]	1	1	0	1	1	0	1 x P_f
[7,10,12]	1	1	0	1	1	1	1 x P_f
[7,11,12]	1	1	0	1	1	1	1 x P_f
[8,9]	1	1	1	0	0	0	
[1,8,9]	0	0	0	1	1	0	5 x [9/64x$P_{f1\emptyset}$ + 55/64xP_f]
[2,8,9]	0	1	0	0	0	0	2 x P_f
[3,8,9]	0	1	1	0	0	0	1 x P_f
[4,8,9]	1	0	0	0	0	0	2 x P_f
[5,8,9]	1	0	1	0	0	0	1 x P_f
[6,8,9]	1	0	1	1	1	0	3 x P_f
[7,8,9]	1	1	0	0	0	0	1 x P_f
[8,9,10]	1	1	1	0	0	0	0
[8,9,11]	1	1	1	0	0	0	0
[8,9,12]	1	1	1	0	1	0	1 x P_f

[8,10]	1	1	1	0	0	1	
[1,8,10]	0	0	1	0	0	0	3 x [9/64x$P_{f1\theta}$ + 55/64xP_f]
[2,8,10]	0	1	0	0	0	0	3 x P_f
[3,8,10]	0	1	1	0	0	1	1 x P_f
[4,8,10]	1	0	0	0	0	1	2 x P_f
[5,8,10]	1	0	1	0	0	0	2 x P_f
[6,8,10]	1	1	1	0	0	0	1 x P_f
[7,8,10]	1	1	0	0	0	1	1 x P_f
[8,9,10]	1	1	1	0	0	0	1 x $P_{f1\theta1}$
[8,10,11]	1	1	1	0	1	1	1 x $P_{f1\theta}$
[8,10,12]	1	1	1	0	1	1	1 x P_f
[8,11]	1	1	1	0	1	0	
[1,8,11]	0	0	1	0	1	0	2 x [9/64x$P_{f1\theta}$ + 55/64xP_f]
[2,8,11]	0	1	0	0	1	0	2 x P_f
[3,8,11]	0	1	1	0	0	0	2 x P_f
[4,8,11]	1	0	0	0	0	0	3 x P_f
[5,8,11]	1	0	1	0	1	0	1 x P_f
[6,8,11]	1	1	1	0	1	0	0
[7,8,11]	1	1	0	0	1	0	1 x P_f
[8,9,11]	1	1	1	0	0	0	1 x $P_{f1\theta}$
[8,10,11]	1	1	1	0	1	1	1 x P_f
[8,11,12]	1	1	1	0	1	1	1 x [11/64x$P_{f1\theta1}$ + 53/64x$P_{f1\theta}$]
[8,12]	1	1	1	0	1	1	
[1,8,12]	0	0	1	0	1	0	3 x [9/64x$P_{f1\theta}$ + 55/64xP_f]
[2,8,12]	0	1	0	0	0	0	4 x P_f
[3,8,12]	0	1	1	0	0	1	2 x P_f
[4,8,12]	1	0	0	0	0	1	3 x P_f
[5,8,12]	1	0	1	0	0	0	3 x P_f
[6,8,12]	1	1	1	0	1	0	1 x P_f
[7,8,12]	1	1	0	0	1	1	1 x P_f
[8,9,12]	1	1	1	0	1	0	1 x $P_{f1\theta}$
[8,10,12]	1	1	1	0	1	1	1 x P_f
[8,11,12]	1	1	1	0	1	1	1 x P_f

[9,10]	1	1	1	1	0	0	
[1,9,10]	0	0	1	1	0	0	2 x [9/64xP_{f10} + 55/64xP_f]
[2,9,10]	0	1	0	0	0	0	3 x P_f
[3,9,10]	0	1	1	0	1	0	3 x P_f
[4,9,10]	1	0	0	0	1	0	4 x P_f
[5,9,10]	1	0	1	0	0	0	2 x P_f
[6,9,10]	1	1	1	1	0	0	0
[7,9,10]	1	1	0	1	0	0	1 x P_f
[8,9,10]	1	1	1	0	0	0	1 x P_f
[9,10,11]	1	1	1	1	0	1	1 x P_{f110}
[9,10,12]	1	1	1	1	1	0	1 x P_f
[9,11]	1	1	1	1	0	1	
[1,9,11]	0	0	1	1	0	1	2 x [9/64xP_{f10} + 55/64xP_f]
[2,9,11]	0	1	0	0	0	1	3 x P_f
[3,9,11]	0	1	1	1	0	0	2 x P_f
[4,9,11]	1	0	0	1	0	0	3 x P_f
[5,9,11]	1	0	1	0	0	1	2 x P_f
[6,9,11]	1	1	1	1	0	1	0
[7,9,11]	1	1	0	1	0	0	2 x P_f
[8,9,11]	1	1	1	0	0	0	2 x P_f
[9,10,11]	1	1	1	1	0	1	0
[9,11,12]	1	1	1	1	1	1	2 x [11/64xP_{f101} + 53/64xP_{f10}]
[9,12]	1	1	1	1	1	0	
[1,9,12]	0	0	1	1	1	0	2 x [9/64xP_{f10} + 55/64xP_f]
[2,9,12]	0	1	0	1	0	0	3 x P_f
[3,9,12]	0	1	1	1	1	0	1 x P_f
[4,9,12]	1	0	0	1	1	0	2 x P_f
[5,9,12]	1	0	1	1	0	0	2 x P_f
[6,9,12]	1	1	1	1	1	0	0
[7,9,12]	1	1	0	1	1	0	1 x P_f
[8,9,12]	1	1	1	0	1	0	1 x P_f
[9,10,12]	1	1	1	1	1	0	1 x P_f
[9,11,12]	1	1	1	1	1	1	2 x P_f

[10,11]	1	1	1	1	1	1	
[1,10,11]	0	0	1	0	0	1	4 x [9/64x$P_{f1\mathbf{0}}$ + 55/64xP_f]
[2,10,11]	0	1	0	0	1	1	3 x P_f
[3,10,11]	0	1	1	1	1	1	1 x P_f
[4,10,11]	1	0	0	1	1	1	2 x P_f
[5,10,11]	1	0	1	0	1	1	2 x P_f
[6,10,11]	1	1	0	0	0	1	3 x P_f
[7,10,11]	1	1	0	1	1	1	1 x P_f
[8,10,11]	1	1	1	0	1	1	1 x P_f
[9,10,11]	1	1	1	1	0	1	1 x P_f
[10,11,12]	1	1	1	1	1	1	0
Total	787.992 P_f + 0.141 $P_{f111\mathbf{0}}$ + 11.266 $P_{f11\mathbf{0}}$ + 119.5 $P_{f1\mathbf{0}}$ + 10.282 $P_{f1\mathbf{0}1}$						

Table 3.16: Full analysis of False Alarm Sequences in a 12-2 MPPM system.

Coded	1st Pulse		2nd Pulse	
i,j	**i-1**	**i+1**	**j-1**	**j+1**
[1,2]	0	0	0	$1x[9/64xP_{s11I}+55/64xP_{s1I}]$
[1,3]	0	$2x[9/64xP_{s1I}+55/64P_s]$	0	$2x[9/64xP_{s110I}+55/64P_{s10I}]$
[1,4]	0	$3x[9/64xP_{s1I}+55/64P_s]$	$2xP_s$	$1xP_s$
[1,5]	0	$3x[9/64xP_{s1I}+55/64P_s]$	$1xP_s$	$3xP_s$
[1,6]	0	$2x[9/64xP_{s1I}+55/64P_s]$	$1xP_s$	$1xP_s$
[1,7]	0	$2x[9/64xP_{s1I}+55/64P_s]$	0	$2xP_s$
[1,8]	0	$3x[9/64xP_{s1I}+55/64P_s]$	$1xP_s$	$1xP_s$
[1,9]	0	$3x[9/64xP_{s1I}+55/64P_s]$	$1xP_s$	$4xP_s$
[1,10]	0	$3x[9/64xP_{s1I}+55/64P_s]$	$1xP_s$	$1xP_s$
[1,11]	0	$3x[9/64xP_{s1I}+55/64P_s]$	0	$2xP_s$
[1,12]	0	$4x[9/64xP_{s1I}+55/64P_s]$	$2x[11/64P_{sI1}+53/64P_s]$	$2x[11/64P_{sI1}+53/64P_s]$
[2,3]	$2x[9/64xP_{s10I1}+55/64P_{sI1}]$	$2x[9/64xP_{s10I1}+55/64P_{sI1}]$	0	$3xP_{s1I}$
[2,4]	$2x[9/64xP_{s10I}+55/64P_s]$	$3x[9/64xP_{s10I}+55/64P_s]$	$1xP_{s10I}$	$1xP_{s10I}$
[2,5]	$2x[9/64xP_{s10I}+55/64P_s]$	$4x[9/64xP_{s10I}+55/64P_s]$	$1xP_s$	$2xP_s$
[2,6]	$2x[9/64xP_{s10I}+55/64P_s]$	$3x[9/64xP_{s10I}+55/64P_s]$	0	$1xP_s$
[2,7]	$2x[9/64xP_{s10I}+55/64P_s]$	$4x[9/64xP_{s10I}+55/64P_s]$	0	$5xP_s$
[2,8]	$1x[9/64xP_{s10I}+55/64P_s]$	$2x[9/64xP_{s10I}+55/64P_s]$	$2xP_s$	$1xP_s$
[2,9]	$1x[9/64xP_{s10I}+55/64P_s]$	$3x[9/64xP_{s10I}+55/64P_s]$	$1xP_s$	$2xP_s$
[2,10]	$2x[9/64xP_{s10I}+55/64P_s]$	$2x[9/64xP_{s10I}+55/64P_s]$	$1xP_s$	$1xP_s$
[2,11]	$2x[9/64xP_{s10I}+55/64P_s]$	$4x[9/64xP_{s10I}+55/64P_s]$	0	$3xP_s$
[2,12]	$2x[9/64xP_{s10I}+55/64P_s]$	$2x[9/64xP_{s10I}+55/64P_s]$	$3x[11/64P_{sI1}+53/64P_s]$	$2x[11/64P_{sI1}+53/64P_s]$
[3,4]	$2xP_{sI1}$	$3xP_{sI1}$	0	$2xP_{s1I}$
[3,5]	$3xP_s$	$1xP_s$	0	$1xP_{s10I}$
[3,6]	$2xP_s$	$1xP_s$	$1xP_s$	$4xP_s$
[3,7]	$3xP_s$	$3xP_s$	$2xP_s$	$1xP_s$
[3,8]	0	$3xP_s$	0	$2xP_s$
[3,9]	$1xP_s$	$3xP_s$	$1xP_s$	$1xP_s$
[3,10]	0	$3xP_s$	$1xP_s$	$3xP_s$
[3,11]	$3xP_s$	$3xP_s$	$2xP_s$	$1xP_s$
[3,12]	0	$3xP_s$	$1xP_s$	$1x[11/64P_{sI1}+53/64P_s]$
[4,5]	$1xP_{sI1}$	$2xP_{sI1}$	0	$1xP_{s1I}$
[4,6]	$1xP_s$	$4xP_s$	$1xP_{s10I}$	$6xP_{s10I}$
[4,7]	$2xP_s$	$3xP_s$	$3xP_s$	$1xP_s$
[4,8]	$2xP_s$	$2xP_s$	0	$2xP_s$
[4,9]	$2xP_s$	$3xP_s$	$1xP_s$	$1xP_s$
[4,10]	$2xP_s$	$2xP_s$	$1xP_s$	$3xP_s$

[4,11]	$2xP_s$	$4xP_s$	$2xP_s$	$1xP_s$
[4,12]	$2xP_s$	$2xP_s$	$1x[11/64P_{sI1}+ 53/64P_s]$	$1x[11/64P_{sI1}+ 53/64P_s]$
[5,6]	$3xP_{sI1}$	0	0	$1xP_{s1I}$
[5,7]	$1xP_s$	$2xP_s$	0	$4xP_{s10I}$
[5,8]	0	$2xP_s$	$3xP_s$	$1xP_s$
[5,9]	$2xP_s$	$2xP_s$	$1xP_s$	$2xP_s$
[5,10]	0	$3xP_s$	$1xP_s$	$1xP_s$
[5,11]	$2xP_s$	$3xP_s$	0	$3xP_s$
[5,12]	0	$4xP_s$	$3xP_s$	$1x[11/64P_{sI1}+ 53/64P_s]$
[6,7]	$2xP_{sI1}$	$2xP_{sI1}$	0	$2xP_{s1I}$
[6,8]	0	$4xP_s$	$1xP_{s10I}$	$1xP_{s10I}$
[6,9]	0	$4xP_s$	$1xP_s$	$5xP_s$
[6,10]	$2xP_s$	$2xP_s$	$2xP_s$	$1xP_s$
[6,11]	$2xP_s$	$3xP_s$	0	$2xP_s$
[6,12]	$2xP_s$	$2xP_s$	$2x[11/64P_{sI1}+ 53/64P_s]$	$2x[11/64P_{sI1}+ 53/64P_s]$
[7,8]	$3xP_{sI1}$	$3xP_{sI1}$	0	$3xP_{s1I}$
[7,9]	$3xP_s$	$2xP_s$	$1xP_{s10I}$	$1xP_{s10I}$
[7,10]	0	$2xP_s$	$1xP_s$	$2xP_s$
[7,11]	$2xP_s$	$2xP_s$	$1xP_s$	$1xP_s$
[7,12]	0	$2xP_s$	$1x[11/64P_{sI1}+ 53/64P_s]$	$1x[11/64P_{sI1}+ 53/64P_s]$
[8,9]	$1xP_{sI1}$	$1xP_{sI1}$	0	$1xP_{s1I}$
[8,10]	$1xP_s$	$2xP_s$	$1xP_{s10I}$	$2xP_{s10I}$
[8,11]	$1xP_s$	$3xP_s$	$1xP_s$	$1xP_s$
[8,12]	$1xP_s$	$2xP_s$	$1xP_s$	$2x[11/64P_{sI1}+ 53/64P_s]$
[9,10]	$1xP_{sI1}$	$3xP_{sI1}$	0	$1xP_{s1I}$
[9,11]	$2xP_s$	$1xP_s$	0	$2xP_{s10I}$
[9,12]	$1xP_s$	0	$2x[11/64P_{sI1}+ 53/64P_s]$	$1x[11/64P_{sI1}+ 53/64P_s]$
[10,11]	$1xP_{sI1}$	$2\ xP_{sI1}$	0	0
Total	$324.203\ P_s + 18.375\ P_{s1I} + 0.562\ P_{s11I} + 30.766\ P_{s10I} + 0.281\ P_{s110I} + 38.25\ P_{sI1} + 0.562\ P_{s10I1}$			

Table 3.17: Full analysis of Wrong Slot Sequences in a 12-2 MPPM system.

Optimum mapping for the 7-2 MULTIPLE PPM System			
c/w	**PCM**	**c/w**	**PCM**
[1,2]	0010	**[3,4]**	1100
[1,3]	1001	**[3,5]**	0100
[1,4]	0101	**[3,6]**	1011
[1,5]	0000	**[3,7]**	1010
[1,6]	0011	**[4,5]**	1101
[1,7]	0001	**[4,6]**	Not used
[2,3]	0100	**[4,7]**	Not used
[2,4]	1111	**[5,6]**	Not used
[2,5]	1110	**[5,7]**	Not used
[2,6]	0111	**[6,7]**	Not used
[2,7]	0110		

Table 7.1: The estimated "optimum" mapping for the $\binom{7}{2}$ *multiple PPM system encoding 4 PCM bits.*

Optimum mapping for the 7-3 MULTIPLE PPM System				**Optimum mapping for the 7-4 MULTIPLE PPM System**			
c/w	**PCM**	**c/w**	**PCM**	**c/w**	**PCM**	**c/w**	**PCM**
[1,2,3]	00001	**[2,3,7]**	10000	**[1,2,3,4]**	00001	**[1,4,6,7]**	11101
[1,2,4]	00100	**[2,4,5]**	00110	**[1,2,3,5]**	00010	**[1,5,6,7]**	11110
[1,2,5]	00010	**[2,4,6]**	01100	**[1,2,3,6]**	00100	**[2,3,4,5]**	00100
[1,2,6]	01000	**[2,4,7]**	01110	**[1,2,3,7]**	00000	**[2,3,4,6]**	11001
[1,2,7]	00000	**[2,5,6]**	11010	**[1,2,4,5]**	00011	**[2,3,4,7]**	01011
[1,3,4]	00101	**[2,5,7]**	10110	**[1,2,4,6]**	00101	**[2,3,5,6]**	11010
[1,3,5]	00011	**[2,6,7]**	11100	**[1,2,4,7]**	01001	**[2,3,5,7]**	11011
[1,3,6]	01001	**[3,4,5]**	10111	**[1,2,5,6]**	00110	**[2,3,6,7]**	11000
[1,3,7]	10001	**[3,4,6]**	11111	**[1,2,5,7]**	01010	**[2,4,5,6]**	10000
[1,4,5]	00111	**[3,4,7]**	11101	**[1,2,6,7]**	01100	**[2,4,5,7]**	10100
[1,4,6]	01101	**[3,5,6]**	11011	**[1,3,4,5]**	00111	**[2,4,6,7]**	11001
[1,4,7]	10101	**[3,5,7]**	11110	**[1,3,4,6]**	01101	**[2,5,6,7]**	10010
[1,5,6]	01011	**[3,6,7]**	01010	**[1,3,4,7]**	10101	**[3,4,5,6]**	10011
[1,5,7]	10011	**[4,5,6]**	01111	**[1,3,5,6]**	01110	**[3,4,5,7]**	11111
[1,6,7]	11001	**[4,5,7]**	Not used	**[1,3,5,7]**	10100	**[3,4,6,7]**	Not used
[2,3,4]	10100	**[4,6,7]**	Not used	**[1,3,6,7]**	11100	**[3,5,6,7]**	Not used
[2,3,5]	10010	**[5,6,7]**	Not used	**[1,4,5,6]**	01111	**[4,5,6,7]**	Not used
[2,3,6]	11000			**[1,4,5,7]**	10111		

Table 7.2: The estimated "optimum" mappings for the $\binom{7}{3}$ *and* $\binom{7}{4}$ *multiple PPM systems encoding 5 PCM bits.*

Optimum mapping for the 15-3 MULTIPLE PPM System							
c/w	**PCM**	**c/w**	**PCM**	**c/w**	**PCM**	**c/w**	**PCM**
[1,2,3]	00000001	**[1,8,10]**	00110010	**[2,6,11]**	10011110	**[3,6,9]**	11101100
[1,2,4]	00000100	**[1,8,11]**	00010011	**[2,6,12]**	10010110	**[3,6,10]**	11111000
[1,2,5]	00011000	**[1,8,12]**	00110011	**[2,6,13]**	10011111	**[3,6,11]**	10101000
[1,2,6]	00000110	**[1,8,13]**	00110111	**[2,6,14]**	10011100	**[3,6,12]**	11001000
[1,2,7]	00001100	**[1,8,14]**	01100011	**[2,6,15]**	10101010	**[3,6,13]**	11100000
[1,2,8]	00110000	**[1,8,15]**	01100010	**[2,7,8]**	10111111	**[3,6,14]**	01101000
[1,2,9]	00100000	**[1,9,10]**	00111010	**[2,7,9]**	10100010	**[3,6,15]**	01001100
[1,2,10]	00010000	**[1,9,11]**	00101000	**[2,7,10]**	10101110	**[3,7,8]**	01001101
[1,2,11]	00000000	**[1,9,12]**	00111000	**[2,7,11]**	10110110	**[3,7,9]**	01001110
[1,2,12]	00001000	**[1,9,13]**	00111001	**[2,7,12]**	10100100	**[3,7,10]**	01001111
[1,2,13]	00000010	**[1,9,14]**	01111000	**[2,7,13]**	10100110	**[3,7,11]**	01010000
[1,2,14]	01000000	**[1,9,15]**	01110000	**[2,7,14]**	10100111	**[3,7,12]**	01010001
[1,2,15]	01000010	**[1,10,11]**	00010010	**[2,7,15]**	11101101	**[3,7,13]**	01010011
[1,3,4]	00011001	**[1,10,12]**	00101010	**[2,8,9]**	10101111	**[3,7,14]**	01010100
[1,3,5]	00001111	**[1,10,13]**	00110100	**[2,8,10]**	10101001	**[3,7,15]**	01110100
[1,3,6]	00100011	**[1,10,14]**	01110110	**[2,8,11]**	10100101	**[3,8,9]**	01110111
[1,3,7]	00101011	**[1,10,15]**	01010110	**[2,8,12]**	10101100	**[3,8,10]**	01111001
[1,3,8]	00111011	**[1,11,12]**	00100001	**[2,8,13]**	10101101	**[3,8,11]**	01111011
[1,3,9]	00011011	**[1,11,13]**	01000001	**[2,8,14]**	10001101	**[3,8,12]**	01111100
[1,3,10]	00001010	**[1,11,14]**	01000100	**[2,8,15]**	11111011	**[3,8,13]**	01111101
[1,3,11]	00000011	**[1,11,15]**	01001000	**[2,9,10]**	10110001	**[3,8,14]**	01110001
[1,3,12]	00001001	**[1,12,13]**	01000101	**[2,9,11]**	10110010	**[3,8,15]**	11110100
[1,3,13]	00001011	**[1,12,14]**	01100001	**[2,9,12]**	10010011	**[3,9,10]**	11110101
[1,3,14]	01001011	**[1,12,15]**	01110001	**[2,9,13]**	10110011	**[3,9,11]**	11110110
[1,3,15]	01010111	**[1,13,14]**	01000111	**[2,9,14]**	10110111	**[3,9,12]**	11111001
[1,4,5]	00010100	**[1,13,15]**	01000011	**[2,9,15]**	11110010	**[3,9,13]**	11111010
[1,4,6]	00000111	**[1,14,15]**	01100000	**[2,10,11]**	11110011	**[3,9,14]**	11111100
[1,4,7]	00111101	**[2,3,4]**	10000001	**[2,10,12]**	11100010	**[3,9,15]**	01010111
[1,4,8]	00110101	**[2,3,5]**	10000100	**[2,10,13]**	10111010	**[3,10,11]**	01011001
[1,4,9]	00010001	**[2,3,6]**	10011000	**[2,10,14]**	10100001	**[3,10,12]**	01011100
[1,4,10]	00010111	**[2,3,7]**	10000110	**[2,10,15]**	11000011	**[3,10,13]**	01011101
[1,4,11]	00000101	**[2,3,8]**	10001100	**[2,11,12]**	11000010	**[3,10,14]**	01011111
[1,4,12]	00010101	**[2,3,9]**	10110000	**[2,11,13]**	11001010	**[3,10,15]**	11100000
[1,4,13]	00011101	**[2,3,10]**	10100000	**[2,11,14]**	11010010	**[3,11,12]**	11011101
[1,4,14]	01010101	**[2,3,11]**	10010000	**[2,11,15]**	11001100	**[3,11,13]**	11011110

[1,4,15]	01110101	**[2,3,12]**	10000000	**[2,12,13]**	11100110	**[3,11,14]**	11100001
[1,5,6]	00001110	**[2,3,13]**	10001000	**[2,12,14]**	11010110	**[3,11,15]**	11100011
[1,5,7]	00111100	**[2,3,14]**	10000010	**[2,12,15]**	11000111	**[3,12,13]**	11100100
[1,5,8]	00111110	**[2,3,15]**	11000000	**[2,13,14]**	11001110	**[3,12,14]**	11100101
[1,5,9]	00011010	**[2,4,5]**	10011001	**[2,13,15]**	11010111	**[3,12,15]**	01100011
[1,5,10]	00011110	**[2,4,6]**	10001111	**[2,14,15]**	01111010	**[3,13,14]**	01100100
[1,5,11]	00010110	**[2,4,7]**	10100011	**[3,4,5]**	01001010	**[3,13,15]**	01100111
[1,5,12]	00011111	**[2,4,8]**	10101011	**[3,4,6]**	01010010	**[3,14,15]**	01101100
[1,5,13]	00011100	**[2,4,9]**	10111011	**[3,4,7]**	01011000	**[4,5,6]**	11000001
[1,5,14]	00011110	**[2,4,10]**	10011011	**[3,4,8]**	11011010	**[4,5,7]**	01101111
[1,5,15]	01111110	**[2,4,11]**	10001010	**[3,4,9]**	01011010	**[4,5,8]**	11110000
[1,6,7]	00101111	**[2,4,12]**	10000011	**[3,4,10]**	01011011	**[4,5,9]**	11110001
[1,6,8]	00100010	**[2,4,13]**	10001001	**[3,4,11]**	01101010	**[4,5,10]**	10110100
[1,6,9]	00101110	**[2,4,14]**	10001011	**[3,4,12]**	01101110	**[4,5,11]**	10111000
[1,6,10]	00110110	**[2,4,15]**	11011011	**[3,4,13]**	01001001	**[4,5,12]**	10111001
[1,6,11]	00100100	**[2,5,6]**	10010100	**[3,4,14]**	01100010	**[4,5,13]**	11001011
[1,6,12]	00100110	**[2,5,7]**	10111101	**[3,4,15]**	01111111	**[4,5,14]**	11001101
[1,6,13]	00100111	**[2,5,8]**	10000111	**[3,5,6]**	11011111	**[4,5,15]**	11001001
[1,6,14]	01100110	**[2,5,9]**	10110101	**[3,5,7]**	11101111	**[4,6,7]**	11011000
[1,6,15]	01000110	**[2,5,10]**	10010001	**[3,5,8]**	11110111	**[4,6,8]**	11011001
[1,7,8]	00111111	**[2,5,11]**	10010111	**[3,5,9]**	11111101	**[4,6,9]**	11001111
[1,7,9]	00101001	**[2,5,12]**	10000101	**[3,5,10]**	11111110	**[4,6,10]**	11010000
[1,7,10]	00100101	**[2,5,13]**	10010101	**[3,5,11]**	11111111	**[4,6,11]**	11010001
[1,7,11]	00101100	**[2,5,14]**	10011101	**[3,5,12]**	11101011	**[4,6,12]**	11010011
[1,7,12]	00101101	**[2,5,15]**	11010100	**[3,5,13]**	11101110	**[4,6,13]**	11010100
[1,7,13]	00001101	**[2,6,7]**	10010010	**[3,5,14]**	11100111	**[4,6,14]**	01100101
[1,7,14]	01101101	**[2,6,8]**	10111100	**[3,5,15]**	11101000	**[4,6,15]**	11000100
[1,7,15]	00100001	**[2,6,9]**	10111110	**[3,6,7]**	11101001	**[4,7,8]**	11000101
[1,8,9]	00110001	**[2,6,10]**	10011010	**[3,6,8]**	11101010	**[4,7,9]**	00000000

Table 7.3: The estimated "optimum" mapping for the $\binom{15}{3}$ *multiple PPM system encoding 8 PCM bits (codewords below [4,7,9] are not used).*

Optimum mapping for the 33-2 MULTIPLE PPM System							
c/w	**PCM**	**c/w**	**PCM**	**c/w**	**PCM**	**c/w**	**PCM**
[1,2]	000100000	**[5,12]**	101101010	**[10,15]**	001100111	**[16,26]**	001100011
[1,3]	000000011	**[5,13]**	100111010	**[10,16]**	001000111	**[16,27]**	100110111
[1,4]	000000101	**[5,14]**	001010010	**[10,17]**	101101100	**[16,28]**	111110011
[1,5]	001000010	**[5,15]**	001101011	**[10,18]**	011111100	**[16,29]**	101100011
[1,6]	000010100	**[5,16]**	101111110	**[10,19]**	000000111	**[16,30]**	101110110
[1,7]	010000001	**[5,17]**	101111000	**[10,20]**	010111100	**[16,31]**	101110111
[1,8]	011000000	**[5,18]**	111111010	**[10,21]**	001000110	**[16,32]**	111110110
[1,9]	000000100	**[5,19]**	001101010	**[10,22]**	111111100	**[16,33]**	101111111
[1,10]	000011000	**[5,20]**	001110110	**[10,23]**	111110100	**[17,18]**	110011000
[1,11]	100000001	**[5,21]**	001001010	**[10,24]**	001100110	**[17,19]**	110001110
[1,12]	000001001	**[5,22]**	011101010	**[10,25]**	011011100	**[17,20]**	110000100
[1,13]	000001010	**[5,23]**	001111011	**[10,26]**	001110100	**[17,21]**	110000010
[1,14]	000010010	**[5,24]**	001110011	**[10,27]**	010101110	**[17,22]**	110001100
[1,15]	101000000	**[5,25]**	101111010	**[10,28]**	011111111	**[17,23]**	110001001
[1,16]	000000110	**[5,26]**	001111110	**[10,29]**	000011010	**[17,24]**	110000001
[1,17]	100001000	**[5,27]**	011111010	**[10,30]**	001001110	**[17,25]**	110101000
[1,18]	010001000	**[5,28]**	001111010	**[10,31]**	001111100	**[17,26]**	111001000
[1,19]	100000010	**[5,29]**	000111010	**[10,32]**	011111101	**[17,27]**	110001011
[1,20]	000010001	**[5,30]**	001011010	**[10,33]**	010011101	**[17,28]**	110001101
[1,21]	000000010	**[5,31]**	001110010	**[11,12]**	100100011	**[17,29]**	110000000
[1,22]	000001100	**[5,32]**	001111000	**[11,13]**	100011111	**[17,30]**	110001000
[1,23]	001001000	**[5,33]**	011111110	**[11,14]**	100010010	**[17,31]**	110010100
[1,24]	001000001	**[6,7]**	010010000	**[11,15]**	101000011	**[17,32]**	110001010
[1,25]	000001000	**[6,8]**	010011010	**[11,16]**	101010111	**[17,33]**	101001000
[1,26]	001010000	**[6,9]**	011000110	**[11,17]**	100001011	**[18,19]**	110111010
[1,27]	010000000	**[6,10]**	010111110	**[11,18]**	100010000	**[18,20]**	110111100
[1,28]	100000100	**[6,11]**	100010110	**[11,19]**	100000111	**[18,21]**	111011010
[1,29]	000000000	**[6,12]**	110110110	**[11,20]**	110010001	**[18,22]**	111011000
[1,30]	000000001	**[6,13]**	110011110	**[11,21]**	111010011	**[18,23]**	110011001
[1,31]	100000000	**[6,14]**	111010110	**[11,22]**	100000011	**[18,24]**	110011011
[1,32]	001000000	**[6,15]**	010010101	**[11,23]**	110010011	**[18,25]**	110111000
[1,33]	000010000	**[6,16]**	011110110	**[11,24]**	101010011	**[18,26]**	110011100
[2,3]	000100011	**[6,17]**	010001110	**[11,25]**	100111011	**[18,27]**	110011010
[2,4]	000101100	**[6,18]**	010011100	**[11,26]**	100010111	**[18,28]**	110011101
[2,5]	000111000	**[6,19]**	110000110	**[11,27]**	000010011	**[18,29]**	110010000

[2,6]	000100110	**[6,20]**	110010111	**[11,28]**	100011011	**[18,30]**	010011000
[2,7]	000110001	**[6,21]**	110010110	**[11,29]**	100010001	**[18,31]**	011011000
[2,8]	000101010	**[6,22]**	010000110	**[1130]**	001010011	**[18,32]**	100011000
[2,9]	000100100	**[6,23]**	011011110	**[11,31]**	101011011	**[18,33]**	100011010
[2,10]	000110100	**[6,24]**	011010111	**[11,32]**	100110011	**[19,20]**	110100101
[2,11]	000110010	**[6,25]**	010011110	**[11,33]**	100010011	**[19,21]**	110100110
[2,12]	000101001	**[6,26]**	011010110	**[12,13]**	100101010	**[19,22]**	110100100
[2,13]	100100100	**[6,27]**	010010111	**[12,14]**	101001001	**[19,23]**	110101101
[2,14]	101100000	**[6,28]**	010110110	**[12,15]**	100100101	**[19,24]**	110100011
[2,15]	000100101	**[6,29]**	010010010	**[12,16]**	101111001	**[19,25]**	110100010
[2,16]	001110000	**[6,30]**	000010110	**[12,17]**	110101001	**[19,26]**	111100111
[2,17]	110100000	**[6,31]**	010010100	**[12,18]**	100001001	**[19,27]**	010100111
[2,18]	000101000	**[6,32]**	010010110	**[12,19]**	110101011	**[19,28]**	010000101
[2,19]	100100000	**[6,33]**	010010011	**[12,20]**	100111101	**[19,29]**	110000111
[2,20]	100110000	**[7,8]**	011110011	**[12,21]**	101101011	**[19,30]**	110100111
[2,21]	100100010	**[7,9]**	010110100	**[12,22]**	100101100	**[19,31]**	110101110
[2,22]	100101000	**[7,10]**	011110000	**[12,23]**	100111001	**[19,32]**	110110111
[2,23]	010101000	**[7,11]**	110110011	**[12,24]**	101101001	**[19,33]**	110111111
[2,24]	001100000	**[7,12]**	100110001	**[12,25]**	111101001	**[20,21]**	111110000
[2,25]	001101000	**[7,13]**	110110000	**[12,26]**	100101101	**[20,22]**	111010101
[2,26]	010100000	**[7,14]**	010110010	**[12,27]**	100101111	**[20,23]**	110111101
[2,27]	010100010	**[7,15]**	110100001	**[12,28]**	100101011	**[20,24]**	111101101
[2,28]	001100001	**[7,16]**	111110001	**[12,29]**	100100001	**[20,25]**	111100001
[2,29]	000100001	**[7,17]**	010111000	**[12,30]**	100001101	**[20,26]**	110110100
[2,30]	000100010	**[7,18]**	110111001	**[12,31]**	100101001	**[20,27]**	100001111
[2,31]	010100101	**[7,19]**	010100011	**[12,32]**	100011001	**[20,28]**	110010101
[2,32]	001100010	**[7,20]**	010110101	**[12,33]**	110101111	**[20,29]**	100010100
[2,33]	000110000	**[7,21]**	110110001	**[13,14]**	111111110	**[20,30]**	100010101
[3,4]	001111111	**[7,22]**	011111001	**[13,15]**	101111100	**[20,31]**	110110101
[3,5]	000111011	**[7,23]**	010011001	**[13,16]**	101101110	**[20,32]**	101110100
[3,6]	010011111	**[7,24]**	010110011	**[13,17]**	110101010	**[20,33]**	110000101
[3,7]	010110111	**[7,25]**	010101001	**[13,18]**	100111000	**[21,22]**	111000000
[3,8]	000001111	**[7,26]**	011110101	**[13,19]**	100100110	**[21,23]**	111000011
[3,9]	010101111	**[7,27]**	010010001	**[13,20]**	100110100	**[21,24]**	111000010
[3,10]	000111100	**[7,28]**	010111001	**[13,21]**	111101110	**[21,25]**	111100010
[3,11]	000110011	**[7,29]**	010100001	**[13,22]**	110101100	**[21,26]**	111000110
[3,12]	000101101	**[7,30]**	010110000	**[13,23]**	110111011	**[21,27]**	010000010
[3,13]	000111110	**[7,31]**	010110001	**[13,24]**	110110010	**[21,28]**	111001010
[3,14]	000011011	**[7,32]**	001110001	**[13,25]**	100111100	**[21,29]**	101001010

[3,15]	000100111	**[7,33]**	011110001	**[13,26]**	110111110	**[21,30]**	101000110
[3,16]	001110111	**[8,9]**	011101111	**[13,27]**	100001110	**[21,31]**	011010010
[3,17]	000111001	**[8,10]**	011101001	**[13,28]**	100111110	**[21,32]**	101100010
[3,18]	010111011	**[8,11]**	010011011	**[13,29]**	100011100	**[21,33]**	110000011
[3,19]	100111111	**[8,12]**	111101011	**[13,30]**	100011110	**[22,23]**	111011100
[3,20]	010111101	**[8,13]**	101001011	**[13,31]**	100101110	**[22,24]**	111101100
[3,21]	000110110	**[8,14]**	011010011	**[13,32]**	100110110	**[22,25]**	111001100
[3,22]	000111101	**[8,15]**	001000011	**[13,33]**	110011111	**[22,26]**	111000100
[3,23]	000011111	**[8,16]**	111001111	**[14,15]**	101110000	**[22,27]**	111001101
[3,24]	001101111	**[8,17]**	011001000	**[14,16]**	101010010	**[22,28]**	111001110
[3,25]	000101011	**[8,18]**	011011010	**[14,17]**	100001010	**[22,29]**	111010000
[3,26]	000110101	**[8,19]**	111001011	**[14,18]**	110010010	**[22,30]**	011001100
[3,27]	010111111	**[8,20]**	010101011	**[14,19]**	111110010	**[22,31]**	101001100
[3,28]	000111111	**[8,21]**	011000010	**[14,20]**	100110010	**[22,32]**	111011110
[3,29]	000011110	**[8,22]**	011001110	**[14,21]**	101000010	**[22,33]**	111010100
[3,30]	000010111	**[8,23]**	011001001	**[14,22]**	101001110	**[23,24]**	111001001
[3,31]	000101111	**[8,24]**	011001011	**[14,23]**	101010001	**[23,25]**	111111001
[3,32]	000101110	**[8,25]**	011001010	**[14,24]**	111010010	**[23,26]**	111111000
[3,33]	000110111	**[8,26]**	011000001	**[14,25]**	101011000	**[23,27]**	111011011
[4,5]	001011110	**[8,27]**	010001011	**[14,26]**	101010100	**[23,28]**	111011111
[4,6]	001010111	**[8,28]**	011011011	**[14,27]**	100000110	**[23,29]**	101011001
[4,7]	001110101	**[8,29]**	011000011	**[14,28]**	101011110	**[23,30]**	111011001
[4,8]	011001101	**[8,30]**	001001011	**[14,29]**	101010000	**[23,31]**	111010001
[4,9]	011010101	**[8,31]**	011000111	**[14,30]**	001010110	**[23,32]**	111011101
[4,10]	001011100	**[8,32]**	011001111	**[14,31]**	101011010	**[23,33]**	111111101
[4,11]	001011011	**[8,33]**	011101011	**[14,32]**	101110010	**[24,25]**	111101111
[4,12]	001101101	**[9,10]**	011110100	**[14,33]**	101010110	**[24,26]**	111000101
[4,13]	000011100	**[9,11]**	011000101	**[15,16]**	101100111	**[24,27]**	010000011
[4,14]	001011000	**[9,12]**	011101101	**[15,17]**	101001101	**[24,28]**	111111011
[4,15]	001111101	**[9,13]**	111100110	**[15,18]**	101000100	**[24,29]**	111000001
[4,16]	101011101	**[9,14]**	111100000	**[15,19]**	100100111	**[24,30]**	111000111
[4,17]	001001001	**[9,15]**	111100100	**[15,20]**	100110101	**[24,31]**	111100011
[4,18]	000011001	**[9,16]**	011100111	**[15,21]**	101000101	**[24,32]**	111101010
[4,19]	000001101	**[9,17]**	010101100	**[15,22]**	111100101	**[24,33]**	011100011
[4,20]	000010101	**[9,18]**	001101100	**[15,23]**	101010101	**[25,26]**	111101000
[4,21]	011011111	**[9,19]**	010100110	**[15,24]**	101100001	**[25,27]**	010101101
[4,22]	001001100	**[9,20]**	011100101	**[15,25]**	101101101	**[25,28]**	001001111
[4,23]	011011001	**[9,21]**	011100010	**[15,26]**	101110101	**[25,29]**	100001100
[4,24]	001000101	**[9,22]**	011000100	**[15,27]**	100000101	**[25,30]**	101001111

[4,25]	001111001	**[9,23]**	011101000	**[15,28]**	101100101	**[25,31]**	101101000
[4,26]	001010001	**[9,24]**	011100001	**[15,29]**	101000001	**[25,32]**	101011100
[4,27]	011011101	**[9,25]**	011101100	**[15,30]**	001100101	**[25,33]**	010101010
[4,28]	001011101	**[9,26]**	011100100	**[15,31]**	101000111	**[26,27]**	110001111
[4,29]	001001101	**[9,27]**	010000100	**[15,32]**	101100100	**[26,28]**	011110111
[4,30]	000011101	**[9,28]**	011101110	**[15,33]**	101101111	**[26,29]**	011010000
[4,31]	001010101	**[9,29]**	001100100	**[16,17]**	101011111	**[26,30]**	011010001
[4,32]	001011111	**[9,30]**	010100100	**[16,18]**	101110001	**[26,31]**	011010100
[4,33]	001011001	**[9,31]**	011100000	**[16,19]**	111110111	**[26,32]**	100011101
[5,6]	010111010	**[9,32]**	011100110	**[16,20]**	111110101	**[26,33]**	010001100
[5,7]	011110010	**[9,33]**	001000100	**[16,21]**	111010111	**[27,28]**	010001111
[5,8]	011111011	**[10,11]**	000001011	**[16,22]**	101100110	**[27,29]**	010001010
[5,9]	001101110	**[10,12]**	001101001	**[16,23]**	101111101	**[27,30]**	010000111
[5,10]	011111000	**[10,13]**	000001110	**[16,24]**	101110011	**[27,31]**	010001001
[5,11]	101111011	**[10,14]**	001010100	**[16,25]**	111111111	**[27,32]**	010001101

Table 7.4: The estimated "optimum" mapping for the $\binom{33}{2}$ *multiple PPM system encoding 9 PCM bits (codewords below [27,32] are not used).*

f_n		100	50	10	1.2	f_n	100	50	10	1.2
GC	$\binom{7}{2}$	-13.0	-12.8	-13.2	-25.6	$\binom{15}{2}$	-10.8	-10.9	-10.9	0.0
	$\binom{7}{3}$	14.2	10.5	13.9	24.8	$\binom{15}{3}$	-1.8	-1.8	-1.0	-13.6
	$\binom{7}{4}$	1.5	1.5	1.2	0.8	$\binom{15}{7}$	-4.3	-5.3	-5.6	-16.7
	$\binom{12}{2}$	-2.2	0.2	0.8	-50.1	$\binom{17}{2}$	0.0	0.3	0.0	0.0
	$\binom{12}{3}$	-2.4	-5.1	-5.6	-21.8	$\binom{17}{3}$	-4.8	-4.9	-5.4	-16.9
	$\binom{12}{6}$	-10.2	-9.2	-9.2	-8.9	$\binom{17}{8}$	-11.2	-10.2	-9.8	-19.3
OPT	$\binom{7}{2}$	-33.1	-33.1	-31.7	-65.8	$\binom{15}{2}$	-22.5	-26.7	-27.3	0.0
	$\binom{7}{3}$	-35.2	-35	-34.7	-23.7	$\binom{15}{3}$	-19.9	-21.3	-24.8	-4.7
	$\binom{7}{4}$	-23.4	-22.9	-22.3	-24.3	$\binom{15}{7}$	-16.6	-16.2	-17.1	-0.1
	$\binom{12}{2}$	-20.3	-19.1	-17.0	-24.8	$\binom{17}{2}$	-18.3	-22.6	-25.7	0.0
	$\binom{12}{3}$	-23.3	-25.3	-26.0	-30.0	$\binom{17}{3}$	-25.6	-27.6	-28.2	-7.2
	$\binom{12}{6}$	-25.6	-28.9	-29.2	-14.9	$\binom{17}{8}$	-29.1	-29.9	-20.3	0.2

Table 7.5: Percentage change in error rate for a $\binom{7}{2}$, $\binom{7}{3}$, $\binom{7}{4}$, $\binom{12}{2}$, $\binom{12}{3}$, $\binom{12}{6}$, $\binom{15}{2}$, $\binom{15}{3}$, $\binom{15}{7}$, $\binom{17}{2}$, $\binom{17}{3}$ *and* $\binom{17}{8}$ *multiple PPM system using linear increment mapping as the reference and a PCM error rate of 1 bit in* 10^9 *pulses. The mappings considered are Gray Code (GC) and Optimum (OPT).*

f_n		100	50	10	1.2	f_n	100	50	10	1.2
GC	$\binom{22}{2}$	-10.8	-10.6	-8.8	0.0	$\binom{28}{14}$	-7.0	-9.4	-10.1	-13.7
	$\binom{22}{3}$	-3.2	-5.1	-6.7	-7.7	$\binom{33}{2}$	-10.3	-10.6	-9.8	0.0
	$\binom{22}{8}$	-8.7	-9.1	-10.3	-11.2	$\binom{33}{3}$	-7.1	-6.7	-8.8	-12.3
	$\binom{22}{11}$	-5.8	-9.3	-12.0	-11.9	$\binom{33}{9}$	-8.9	-9.4	-10.1	-11.4
	$\binom{28}{2}$	-5.2	-5.2	-4.2	0.0	$\binom{33}{12}$	-10.4	-13.7	-16.7	-17.1
	$\binom{28}{9}$	-6.1	-7.9	-8.1	-10.4	$\binom{33}{16}$	-11.1	-13.3	-15.7	-19.2
OPT	$\binom{22}{2}$	-21.7	-23.5	-26.6	0.0	$\binom{28}{14}$	-28.7	-25.7	-20.6	-4.7
	$\binom{22}{3}$	-26.7	-27.1	-28.3	0.4	$\binom{33}{2}$	-21.4	-22	-22.3	0.0
	$\binom{22}{8}$	-27.8	-29.2	-20.1	-5.6	$\binom{33}{3}$	-22.1	-24.8	-25.7	0.8
	$\binom{22}{11}$	-22.0	-21.8	-20.4	-8.9	$\binom{33}{9}$	-28.6	-29.9	-31.8	-0.5
	$\binom{28}{2}$	-26.7	-27.8	-29.2	0.2	$\binom{33}{12}$	-31.2	-33.7	-37.7	-5.1
	$\binom{28}{9}$	-20.1	-22.2	-25.7	-0.8	$\binom{33}{16}$	-33.3	-37.8	-39.2	-7.8

Table 7.6: Percentage change in error rate for a $\binom{22}{2}$, $\binom{22}{3}$, $\binom{22}{8}$, $\binom{22}{11}$, $\binom{28}{2}$, $\binom{28}{9}$, $\binom{28}{14}$, $\binom{33}{2}$, $\binom{33}{3}$, $\binom{33}{9}$, $\binom{33}{12}$ *and* $\binom{33}{16}$ *multiple PPM system using linear increment mapping as the reference and a PCM error rate of 1 bit in* 10^9 *pulses. The mappings considered are Gray Code (GC) and Optimum (OPT).*

Erasure Error [1,2,?]	**Mapping**	
	Optimum	**Ideal**
[1,2]	000001	000000
[1,3]	000100	000001
[1,4]	011000	000010
[1,5]	000110	000100
[1,6]	001100	001000
[1,7]	000010	010000
[1,8]	100000	100000
[1,9]	010000	000001
[1,10]	000000	000010
[1,11]	001000	000100
[1,12]	010000	001000
Averaged [1,?] codeword	000000	010000
Averaged Hamming Distance	13	11

Table 7.7: Total Hamming distance for "Optimum" and "Ideal" mapping of the [1,?] ER averaged codeword in a $\binom{12}{2}$ *multiple PPM system.*

Ideal mapping for the 7-4 MULTIPLE PPM System			
c/w	**PCM**	**c/w**	**PCM**
[1,2,3,4]	00000	**[1,4,6,7]**	01000
[1,2,3,5]	00001	**[1,5,6,7]**	00001
[1,2,3,6]	00010	**[2,3,4,5]**	00010
[1,2,3,7]	00100	**[2,3,4,6]**	00100
[1,2,4,5]	01000	**[2,3,4,7]**	00100
[1,2,4,6]	00001	**[2,3,5,6]**	00010
[1,2,4,7]	00010	**[2,3,5,7]**	01000
[1,2,5,6]	00100	**[2,3,6,7]**	00001
[1,2,5,7]	01000	**[2,4,5,6]**	00010
[1,2,6,7]	00001	**[2,4,5,7]**	00100
[1,3,4,5]	00010	**[2,4,6,7]**	00010
[1,3,4,6]	00100	**[2,5,6,7]**	00100
[1,3,4,7]	00100	**[3,4,5,6]**	00100
[1,3,5,6]	01000	**[3,4,5,7]**	10000
[1,3,5,7]	00001	**[3,4,6,7]**	Not used
[1,3,6,7]	00001	**[3,5,6,7]**	Not used
[1,4,5,6]	00010	**[4,5,6,7]**	Not used
[1,4,5,7]	00100		

Table 7.8: Ideal mapping for the $\binom{7}{4}$ *multiple PPM system encoding 5 PCM bits.*

	12_2		12_3		12_6		15_2		15_3	
f_n	OPT	IDEAL	OPT	IDEAL	OPT	IDEAL	OPT	IDEAL	OPT	IDEAL
100	20.3	57.4	23.3	68.6	25.6	80.1	22.5	58.6	19.9	73.9
50	19.1	54.6	25.3	68.4	28.9	84.2	26.7	59.1	21.3	73.4
10	17	56.8	26	68.3	29.2	84	27.3	58.8	24.8	73.7
1.2	24.8	57.1	30	77.3	14.9	84.8	0	50.8	4.7	80.1
	7_2		7_3		7_4		17_2		28_2	
f_n	OPT	IDEAL	OPT	IDEAL	OPT	IDEAL	OPT	IDEAL	OPT	IDEAL
100	33.1	46.4	35.2	54.4	23.4	84.4	18.3	65.4	26.7	64.3
50	33.1	46.7	35	54.2	22.9	84.2	22.6	65.2	27.8	64
10	31.7	47	34.7	54.1	22.3	84	25.7	64.8	29.2	64.4
1.2	65.8	43	23.7	49.3	24.3	84.8	0	63.8	0.2	58.9
	33_2									
f_n	OPT	IDEAL								
100	21.4	68.6								
50	22	68.4								
10	22.3	69.2								
1.2	0	57.8								

Table 7.9: Percentage change in error rate for a $\binom{7}{2}$, $\binom{7}{3}$, $\binom{7}{4}$, $\binom{12}{2}$, $\binom{12}{3}$, $\binom{12}{6}$, $\binom{15}{2}$, $\binom{15}{3}$, $\binom{17}{2}$, $\binom{28}{2}$ *and* $\binom{33}{2}$ *multiple PPM system using linear increment mapping as the reference and a PCM error rate of 1 bit in* 10^9 *pulses. The mappings considered are Optimum (OPT) and Ideal (IDEAL).*

APPENDIX C

Software Printout

```
//-------------------------------------------------------------------------------------//
//                                    COMMENTS                                         //
//-------------------------------------------------------------------------------------//
//Program:     MPPM PCM EQUIVALENT Error Rate Calculation                              //

//Cut:         Final                                                                   //

//Date:        31/08/06                                                                //

//Author:      Konstantinos Nikolaidis                                                 //

//Inputs:      Keyboard (X-Y numbers and other i/p's)                                  //

//Description: The program calculates erasure, false alarm, wrong slot error rates and //
//             display the error sequences of the MPPM system using 2 algorithms       //
//             (2-pulse and Original)                                                  //
//Further:     Create error reduction algorithms                                       //
//                                                                                     //
//Time Analysis:  BEST CASE:              WORST CASE:                                  //
//Space Analysis: BEST CASE:              WORST CASE:                                  //
//                                                                                     //
//USE:            ERASURE:     i.e. for 12-2  er put 64 then 6 then 12 and 0 (pasc 2) //
//                                  for 12-1  er put 8  then 3 then 0  and 0           //
//                                  for 12-11 er put 8  then 3 then 66 and 0           //
//                                                                                     //
//                FALSE ALARM: i.e. for 12-2  fa put 64 then 6 then 0 and 220 (pasc 3)//
//                                  for 12-1  fa put 8  then 3 then 0 and 28           //
//                                  for 12-11 fa put 8  then 3 then 0 and 1            //
//                                                                                     //
//                WRONG SLOT:  i.e. for 12-2  ws put 64 then 6 then 12 and 220       //
//                                  (pasc 2)-(pasc 3)                                  //
//                                  for 12-1  ws put 8  then 3 then 0  and 28          //
//                                  for 12-11 ws put 8  then 3 then 66 and 1           //
//                                                                                     //
//                for the RANDOM mapping range i.e. for 12-2 put from 0 to 64          //

//                for the LINEAR DECREMENT mapping start from pasc-1 i.e. 12-2 put     //
//                start 63                                                             //
//                                                                                     //
//                                                                                     //
//             System   PCM Bits      Data Range   Pasc-RD-LI-LD-GC ER(pasc2)-FA(pasc3) //
//                                                                                     //
//              12-1       3            8-(0-7)           1-28 (pow(2,pcm bits) 2)     //
//              12-2       6           64-(0-63)          12-220                       //
//              12-3       7          128-(0-127)         66-495                       //
//              12-4       8          256-(0-255)         220-792                      //
//              12-5       9          512-(0-511)         495-924                      //
//              12-6       9          512-(0-511)         792-792                      //
//              12-7       9          512-(0-511)         924-495                      //
//              12-8       8          256-(0-255)         792-220                      //
//              12-9       7          128-(0-127)         495-66                       //
//              12-10      6           64-(0-63)          220-12                       //
//              12-11      3            8-(0-7)           66-1                         //
//                                                                                     //
//             The commented areas are code used for automated calculation only        //
//                          Uncomment the areas with the registers                     //
//       This version is only for printing (A4)!!!!There is no indentation in the code //
//                         The indented version will be used during the viva           //
//                                                                                     //
//-------------------------------------------------------------------------------------//
//                                    CLASSES                                          //
//-------------------------------------------------------------------------------------//


#include<iostream.h>
#include<iomanip.h>
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include<math.h>
#include<ctype.h>
#include<assert.h>
#include<errno.h>
#include<float.h>
#include<limits.h>
#include<locale.h>
#include<setjmp.h>
#include<signal.h>
#include<stdarg.h>
#include<stddef.h>
#include<string.h>
```

```
#include<conio.h>

#include<windows.h>
#include <wincon.h>

#include <fstream.h>

//21T

//---------------------------------------------------------------------------------------//
//                                  FUNCTION PROTOTYPES                                  //
//---------------------------------------------------------------------------------------//

void gotoxy(int x, int y)
{
  HANDLE hConsoleOutput; COORD dwCursorPosition;
  dwCursorPosition.X = x; dwCursorPosition.Y = y;
  hConsoleOutput = GetStdHandle(STD OUTPUT HANDLE);
  SetConsoleCursorPosition(hConsoleOutput,dwCursorPosition);
} //------4T

int clrscr()//all the screen
{
  HANDLE hndl = GetStdHandle(STD OUTPUT_HANDLE);
  CONSOLE SCREEN BUFFER INFO csbi;
  GetConsoleScreenBufferInfo(hndl, &csbi);
  DWORD written;
  DWORD N = csbi.dwSize.X * csbi.dwCursorPosition.Y + csbi.dwCursorPosition.X + 1;
  COORD curhome = {0,0};
  FillConsoleOutputCharacter(hndl, ' ', N, curhome, &written);
  csbi.srWindow.Bottom -= csbi.srWindow.Top;
  csbi.srWindow.Top = 0;
  SetConsoleWindowInfo(hndl, TRUE, &csbi.srWindow);
  SetConsoleCursorPosition(hndl, curhome);
  return 0;
} //------12T

int clrscr2()//arrays
{
  HANDLE hndl = GetStdHandle(STD OUTPUT_HANDLE);
  CONSOLE SCREEN BUFFER INFO csbi;
  GetConsoleScreenBufferInfo(hndl, &csbi);
  DWORD written;
  DWORD N = csbi.dwSize.X * csbi.dwCursorPosition.Y + csbi.dwCursorPosition.X + 1;
  COORD curhome = {10,55};
  FillConsoleOutputCharacter(hndl, ' ', N, curhome, &written);
  csbi.srWindow.Bottom -= csbi.srWindow.Top;
  csbi.srWindow.Top = 0;
  SetConsoleWindowInfo(hndl, TRUE, &csbi.srWindow);
  SetConsoleCursorPosition(hndl, curhome);
  return 0;
} //-----12T


//---------------------------------------------------------------------------------------//


void control_panel ();

void print input arrays (int *, unsigned long int, int, int, int, unsigned long int,
unsigned long int, unsigned long int, int, int, unsigned long int, unsigned long int,
unsigned long int, int, int, int, int, int, int, int);

int print error rate arrays (int *, unsigned long int, int, int, int, int, int, int,
unsigned long int, unsigned long int, unsigned long int, int, int, unsigned long int,
unsigned long int, unsigned long int, int, int, int, int, int, int);

void print random array (int *, unsigned long int, int, int, int, unsigned long int,
unsigned long int, unsigned long int, int, int, unsigned long int, unsigned long int,
unsigned long int, int, int, int, int, int, int, int);

void print random array2 (int *, unsigned long int, int, int, int, unsigned long int,
unsigned long int, unsigned long int, int, int, unsigned long int, unsigned long int,
unsigned long int, int, int, int, int, int, int, int);

unsigned long double pascal_calc (int, int);

void fill data (int, int, unsigned long int, unsigned long int, unsigned long int, int,
int, unsigned long int, unsigned long int, unsigned long int, int, int, int, int, int,
int, int);//---7T
```

```
//-------------------------------------------------------------------------------------------//
//                                    MAIN PROGRAM                                           //
//-------------------------------------------------------------------------------------------//

void main (void)
{

 int x,y;

 int press1,press2,press3,press4,press5,press6,erasure bits,false alarm bits,
     wrong slot bits,pcm bits,aut pcm bits,start number,end number,extra pulses,
     start,finish,remainder,number,flag,counter,bit counter,row counter,row counter2,
     flag_er,flag_fa,flag_ws,limit,pulse_counter,pulse_counter2,zero_counter;

 int i,j,k,l,m,w,p,n,n1,o,z,l1,c,question; //------47 integers

 float a;

 float photon energy,mark space correction,pulse energy,energy_in_frame,
       energy_per_pcm_bit,B,dBm,minimum; //--------9 floats

 unsigned long float erasure weight,erasure weight2,false_alarm_weight,
       false_alarm_weight2; //------4 long floats

 unsigned long int pasc,pasc2,pasc3;
 unsigned long int aut_pasc,aut_pasc2,aut_pasc3; //---------6 long integers

 int *mppm,*data,*random,*mppm_data,*weighted_er_mppm_data,*weighted_fa_mppm_data;

 int *mppm_er,*mppm_fa,*mppm_ws; //-----------9 int registers

 float *sequence, *sequence2; //------------2 float registers

 fstream file op("error rates.txt",ios::out);
 fstream file op1("input arrays.txt",ios::out);
 fstream file op2("random.txt",ios::out|ios::in);
 fstream file op3("random.txt", ios::in);
 fstream file_op4("sequences.txt",ios::out); //-----------5 files

 //16T

//-------------------------------------------------------------------------------------------//

 //registers used for automated calculations--save the impcat of every sequence
/*float standard [24]    = {2298,2303,2312,2324,2341,2366,2405,2466,2565,2602,2676,
                           2783,2887,2956,3041,3150,3296,3502,3812,4348,5685,6280,
                           8939,72550};
 float standard fa [24] = {2495,2502,2512,2526,2546,2574,2617,2685,2792,2832,2911,
                           3024,3133,3203,3290,3400,3545,3743,4027,4473,5685,6280,
                           8939,72550};
 float pf 10 [24]       = {2557,2565,2575,2590,2611,2641,2687,2759,2875,2918,3002,
                           3126,3245,3323,3421,3545,3711,3944,4294,4891,6336,6967,
                           8957,72550};
 float pf 110 [24]      = {2559,2566,2577,2592,2613,2642,2689,2761,2877,2920,3005,
                           3128,3248,3326,3424,3549,3715,3949,4301,4901,6363,7006,
                           8972,72550};
 float pf 101 [24]      = {2557,2565,2575,2590,2611,2641,2687,2759,2875,2918,3002,
                           3126,3245,3323,3421,3545,3711,3944,4294,4891,6336,6967,
                           8957,72550};
 float pf 1101 [24]     = {2559,2566,2577,2592,2613,2642,2689,2761,2877,2920,3005,
                           3128,3248,3326,3424,3549,3715,3949,4301,4901,6363,7007,
                           8977,72550};

 float standard er [24] = {2601,2606,2617,2632,2653,2682,2728,2791,2913,2956,3041,
3165,3286,3365,3464,3591,3761,4002,4367,5006,6640,7376,9650,72550};
 float pe1 1 [24]       = {2298,2303,2312,2324,2341,23662405,2466,2565,2602,2676,
2783,2887,2956,3041,3150,3296,3502,3812,4348,5685,6280,8939,72550};
 float pe 11 [24]       = {2539,2546,2556,2570,2590,2618,2662,2730,2842,2883,2965,
3084,3200,3275,3369,3490,3650,3875,4211,4780,6094,6606,8941,72550};
 float pe 111 [24]      = {2537,2545,2555,2568,2588,2615,2659,2734,2840,2882,2963,
3083,3198,3273,3367,3487,3647,3871,4206,4774,6081,6589,8941,72550};
 float pe 101 [24]      = {2597,2605,2616,2629,2650,2678,2724,2798,2911,2954,3039,
3163,3284,3363,3462,3588,3758,3997,4362,4999,6625,7356,9623,72550};
 float pe 1101 [24]     = {2597,2605,2616,2629,2649,2677,2723,2804,2911,2954,3039,
3163,3284,3363,3462,3588,3758,3997,4360,4997,6624,7354,9622,72550};
 float pe 10101 [24]    = {2601,2606,2617,2632,2653,2682,2728,2791,2913,2956,3041,
3165,3286,3365,3464,3591,3761,4002,4367,5006,6640,7376,9650,72550};

 float standard ws [24] = {2298,2303,2312,2324,2341,2366,2405,2466,2565,2602,2676,
2783,2887,2956,3041,3150,3296,3502,3812,4348,5685,6280,8939,72550};
 float ps1_1 [24]       = {2298,2303,2312,2324,2341,2366,2405,2466,2565,2602,2676,
```

```
2783,2887,2956,3041,3150,3296,3502,3812,4348,5685,6280,8939,72550};
 float ps 11 [24]       = {2298,2303,2312,2324,2341,2366,2405,2466,2565,2602,2676,
2783,2887,2956,3041,3150,3296,3502,3812,4348,5685,6283,11130,72550};
 float ps 111 [24]      = {2298,2303,2312,2324,2341,2366,2405,2466,2565,2602,2676,
2783,2887,2956,3041,3150,3296,3502,3812,4348,5685,6288,11690,102000};
 float ps 101 [24]      = {2298,2303,2312,2324,2341,2366,2405,2466,2565,2602,2676,
2783,2887,2956,3041,3150,3296,3502,3812,4348,5685,6280,8939,72550};
 float ps 1101 [24]     = {2298,2303,2312,2324,2341,2366,2405,2466,2565,2602,2676,
2783,2887,2956,3041,3150,3296,3502,3812,4348,5685,6280,8939,72550};
 float ps 1011 [24]     = {2298,2303,2312,2324,2341,2366,2405,2466,2565,2602,2676,
2783,2887,2956,3041,3150,3296,3502,3812,4348,5685,6280,8939,72550};
 float ps 10111 [24]    = {2298,2303,2312,2324,2341,2366,2405,2466,2565,2602,2676,
2783,2887,2956,3041,3150,3296,3502,3812,4348,5685,6280,8939,72550};
 float ps 11011 [24]    = {2298,2303,2312,2324,2341,2366,2405,2466,2565,2602,2676,
2783,2887,2956,3041,3150,3296,3502,3812,4348,5685,6280,8939,72550};
 float ps 110111 [24]   = {2298,2303,2312,2324,2341,2366,2405,2466,2565,2602,2676,
2783,2887,2956,3041,3150,3296,3502,3812,4348,5685,6280,8939,72550};
 float ps 1 1 1 [24]    = {2298,2303,2312,2324,2341,2366,2405,2466,2565,2602,2676,
2783,2887,2956,3041,3150,3296,3502,3812,4348,5685,6280,8939,72550};
 float ps 1 1 11 [24]   = {2298,2303,2312,2324,2341,2366,2405,2466,2565,2602,2676,
2783,2887,2956,3041,3150,3296,3502,3812,4348,5685,6283,10680,72550};
 float ps 11 1 1 [24]   = {2298,2303,2312,2324,2341,2366,2405,2466,2565,2602,2676,
2783,2887,2956,3041,3150,3296,3502,3812,4348,5685,6287,11170,72550};
 float ps 11 1 11 [24]  = {2298,2303,2312,2324,2341,2366,2405,2466,2565,2602,2676,
2783,2887,2956,3041,3150,3296,3502,3812,4348,5685,6287,11170,72550};*/

/*
 float standard fa [24] = {0.754789272,0.756653992,0.754716981,0.753731343,
                           0.753676471,0.753623188,0.746478873,0.742372881,
                           0.727564103,0.72327044,0.714285714,0.698550725,
                           0.681440443,0.667567568,0.650130548,0.626566416,
                           0.594272076,0.539149888,0.439672802,0.226039783,
                           0,0,0,0};
 float pf 10 [24]       = {0.992337165,0.996197719,0.99245283,0.992537313,
                           0.992647059,0.996376812,0.992957746,0.993220339,
                           0.993589744,0.993710692,0.990881459,0.994202899,
                           0.991689751,0.991891892,0.992167102,0.989974937,
                           0.990453461,0.988814318,0.985685072,0.981916817,
                           0.960176991,0.944979367,0.473684211,0,};
 float pf 110 [24]      = {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0.998624484,
                           0.868421053,0};
 float pf 101 [24]      = {0.992337165,0.996197719,0.99245283,0.992537313,
                           0.992647059,0.996376812,0.992957746,0.993220339,
                           0.993589744,0.993710692,0.990881459,0.994202899,
                           0.991689751,0.991891892,0.992167102,0.989974937,
                           0.990453461,0.988814318,0.985685072,0.981916817,
                           0.960176991,0.944979367,0.473684211,0};
 float pf_1101 [24]     = {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0};

//-------------------------------------------------------------------------------------//

 float standard er [24] = {1,1,1,1,1,1,1,0.961538462,1,1,1,1,1,1,1,1,1,1,1,1,1,0};
 float pe1 1 [24]       = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
 float pe 11 [24]       = {0.795379538,0.801980198,0.8,0.798701299,0.798076923,
                           0.797468354,0.795665635,0.781065089,0.795977011,0.793785311,
                           0.791780822,0.787958115,0.784461153,0.7799511,0.775413712,
                           0.770975057,0.761290323,0.746,0.718918919,0.656534954,
                           0.428272251,0.297445255,0.00281294,0};
 float pe 111 [24]      = {0.788778878,0.798679868,0.796721311,0.792207792,0.791666667,
                           0.787974684,0.786377709,0.792899408,0.790229885,0.790960452,
                           0.78630137,0.785340314,0.779448622,0.775061125,0.770685579,
                           0.764172336,0.75483871,0.738,0.70990991,0.647416413,
                           0.414659686,0.281934307,0.00281294,0};
 float pe 101 [24]      = {0.98679868,0.99669967,0.996721311,0.99025974,0.987179487,
                           0.984177215,0.984520124,1,0.994252874,0.994350282,
                           0.994520548,0.994764398,0.994987469,0.995110024,0.995271868,
                           0.993197279,0.993548387,0.99,0.987387387,0.986322188,
                           0.983246073,0.979927007,0.960618847,0};
 float pe 1101 [24]     = {0.98679868,0.99669967,0.996721311,0.99025974,0.987179487,
                           0.984177215,0.984520124,1,0.994252874,0.994350282,
                           0.994520548,0.994764398,0.994987469,0.995110024,0.995271868,
                           0.993197279,0.993548387,0.99,0.987387387,0.986322188,
                           0.983246073,0.979927007,0.960618847,0};
 float pe_10101 [24]    = {1,1,1,1,1,1,1,0.961538462,1,1,1,1,1,1,1,1,1,1,1,1,1,0};

 float standard ws [24] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
 float ps1 1 [24]       = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
 float ps 11 [24]       = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.375,
                           0.796437659,0};
 float ps 111 [24]      = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1};
 float ps 101 [24]      = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
 float ps_1101 [24]     = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
```

```
 float ps 1011 [24]     = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
 float ps 10111 [24]    = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
 float ps 11011 [24]    = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
 float ps 110111 [24]   = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
 float ps 1 1 1 [24]    = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
 float ps 1 1 11 [24]   = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.375,
                           0.632860778,0};
 float ps 11 1 1 [24]   = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.875,
                           0.810977826,0};
 float ps 11 1 11 [24]  = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.875,
                           0.810977826,0};*/

//-------------------------------------------------------------------------------------//

/*float standard [24]    = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

 float standard fa [24] = {0.650165017,0.656765677,0.655737705,0.655844156,0.657051282,
                           0.658227848,0.656346749,0.647928994,0.652298851,0.649717514,
                           0.643835616,0.630890052,0.616541353,0.60391198,0.588652482,
                           0.566893424,0.535483871,0.482,0.387387387,0.189969605,0,0,
                           0,0};
 float pf 10 [24]       = {0.854785479,0.864686469,0.862295082,0.863636364,
                           0.865384615,0.870253165,0.873065015,0.866863905,0.890804598,
                           0.892655367,0.893150685,0.897905759,0.897243108,0.897310513,
                           0.898345154,0.89569161,0.892473118,0.884,0.868468468,
                           0.825227964,0.681675393,0.626824818,0.006543075,0};
 float pf 110 [24]      = {0.861386139,0.867986799,0.868852459,0.87012987,0.871794872,
                           0.873417722,0.879256966,0.872781065,0.896551724,0.898305085,
                           0.901369863,0.903141361,0.904761905,0.904645477,0.905437352,
                           0.904761905,0.901075269,0.894,0.881081081,0.840425532,
                           0.709947644,0.662408759,0.011995638,0};
 float pf 101 [24]      = {0.854785479,0.864686469,0.862295082,0.863636364,0.865384615,
                           0.870253165,0.873065015,0.866863905,0.890804598,0.892655367,
                           0.893150685,0.897905759,0.897243108,0.897310513,0.898345154,
                           0.89569161,0.892473118,0.884,0.868468468,0.825227964,
                           0.681675393,0.626824818,0.006543075,0};
 float pf 1101 [24]     = {0.861386139,0.867986799,0.868852459,0.87012987,0.871794872,
                           0.873417722,0.879256966,0.872781065,0.896551724,0.898305085,
                           0.901369863,0.903141361,0.904761905,0.904645477,0.905437352,
                           0.904761905,0.901075269,0.894,0.881081081,0.840425532,
                           0.709947644,0.663321168,0.013813159,0};

//-------------------------------------------------------------------------------------//

 float standard er [24] = {1,1,1,1,1,1,1,0.961538462,1,1,1,1,1,1,1,1,1,1,1,1,1,
                           0.258451472,0};
 float pe1 1 [24]       = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
 float pe 11 [24]       = {0.795379538,0.801980198,0.8,0.798701299,0.798076923,
                           0.797468354,0.795665635,0.781065089,0.795977011,
                           0.793785311,
                           0.791780822,0.787958115,0.784461153,0.7799511,
                           0.775413712,
                           0.770975057,0.761290323,0.746,0.718918919,0.656534954,
                           0.428272251,0.297445255,0.000727008,0};
 float pe 111 [24]      = {0.788778878,0.798679868,0.796721311,0.792207792,
                           0.791666667,
                           0.787974684,0.786377709,0.792899408,0.790229885,
                           0.790960452,
                           0.78630137,0.785340314,0.779448622,0.775061125,
                           0.770685579,
                           0.764172336,0.75483871,0.738,0.70990991,0.647416413,
                           0.414659686,0.281934307,0.000727008,0};
 float pe 101 [24]      = {0.98679868,0.99669967,0.996721311,0.99025974,
                           0.990384615,
                           0.987341772,0.987616099,0.982248521,0.994252874,
                           0.994350282,
                           0.994520548,0.994764398,0.994987469,0.995110024,
                           0.995271868,
                           0.993197279,0.993548387,0.99,0.990990991,0.989361702,
                           0.984293194,0.981751825,0.248636859,0};
 float pe 1101 [24]     = {0.98679868,0.99669967,0.996721311,0.99025974,
                           0.987179487,
                           0.984177215,0.984520124,1,0.994252874,0.994350282,
                           0.994520548,
                           0.994764398,0.994987469,0.995110024,0.995271868,
                           0.993197279,
                           0.993548387,0.99,0.987387387,0.986322188,0.983246073,
                           0.979927007,0.248273355,0};
 float pe 10101 [24]    = {1,1,1,1,1,1,1,0.961538462,1,1,1,1,1,1,1,1,1,1,1,1,1,
                           0.258451472,0};

//-------------------------------------------------------------------------------------//
```

```
float standard ws [24] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float ps1 1 [24]       = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float ps 11 [24]       = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.002737226,
                          0.796437659,0};
float ps 111 [24]      = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.00729927,1,1};
float ps 101 [24]      = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float ps 1101 [24]     = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float ps 1011 [24]     = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float ps 10111 [24]    = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float ps 11011 [24]    = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float ps 110111 [24]   = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float ps 1 1 1 [24]    = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float ps 1 1 11 [24]   = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.002737226,
                          0.632860778,0};
float ps 11 1 1 [24]   = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.006386861,
                          0.810977826,0};
float ps 11 1 11 [24]  = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.006386861,
                          0.810977826,0};*/

/*float standard [24]    = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

float standard fa [24] = {197,199,200,202,205,208,212,219,227,230,235,241,246,247,
                          249,250,249,241,215,125,0,0,0,0};
float pf 10 [24]       = {259,262,263,266,270,275,282,293,310,316,326,343,358,367,
                          380,395,415,442,482,543,651,687,18,0};
float pf 110 [24]      = {261,263,265,268,272,276,284,295,312,318,329,345,361,370,
                          383,399,419,447,489,553,678,726,33,0};
float pf 101 [24]      = {259,262,263,266,270,275,282,293,310,316,326,343,358,367,
                          380,395,415,442,482,543,651,687,18,0};
float pf 1101 [24]     = {261,263,265,268,272,276,284,295,312,318,329,345,361,370,
                          383,399,419,447,489,553,678,727,38,0};

float standard er [24] = {303,303,305,308,312,316,323,325,348,354,365,382,399,409,
                          423,441,465,500,555,658,955,1096,711,0};
float pe1 1 [24]       = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float pe 11 [24]       = {241,243,244,246,249,252,257,264,277,281,289,301,313,319,
                          328,340,354,373,399,432,409,326,2,0};
float pe 111 [24]      = {239,242,243,244,247,249,254,268,275,280,287,300,311,317,
                          326,337,351,369,394,426,396,309,2,0};
float pe 101 [24]      = {299,302,304,305,309,312,319,332,346,352,363,380,397,407,
                          421,438,462,495,550,651,940,1076,684,0};
float pe 1101 [24]     = {299,302,304,305,308,311,318,338,346,352,363,380,397,407,
                          421,438,462,495,548,649,939,1074,683,0};
float pe 10101 [24]    = {303,303,305,308,312,316,323,325,348,354,365,382,399,409,
                          423,441,465,500,555,658,955,1096,711,0};

float standard ws [24] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float ps1 1 [24]       = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float ps 11 [24]       = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,2191,0};
float ps 111 [24]      = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,8,2751,29450};
float ps 101 [24]      = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float ps 1101 [24]     = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float ps 1011 [24]     = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float ps 10111 [24]    = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float ps 11011 [24]    = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float ps 110111 [24]   = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float ps 1 1 1 [24]    = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float ps 1 1 11 [24]   = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,1741,0};
float ps 11 1 1 [24]   = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,7,2231,0};
float ps_11_1_11 [24]  = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,7,2231,0};*/

/*float off standard fa [24] = {0.2724,0.2745,0.2761,0.2784,0.2821,0.2858,0.2926,
                                0.3028,0.3184,0.3241,0.3342,0.3501,0.3654,0.3742,
                                0.3868,0.4023,0.4227,0.451,0.4943,0.5666,0.7252,
                                0.7832,1.4045,2.945};
float off pf 10 [24]       = {0.2662,0.2682,0.2698,0.272,0.2756,0.2791,0.2856,
                              0.2954,0.3101,0.3155,0.3251,0.3399,0.3542,0.3622,
                              0.3737,0.3878,0.4061,0.4309,0.4676,0.5248,0.6601,
                              0.7145,1.4027,2.945};
float off pf 110 [24]      = {0.266,0.2681,0.2696,0.2718,0.2754,0.279,0.2854,
                              0.2952,0.3099,0.3153,0.3248,0.3397,0.3539,0.3619,
                              0.3734,0.3874,0.4057,0.4304,0.4669,0.5238,0.6574,
                              0.7106,1.4012,2.945};
float off pf 101 [24]      = {0.2662,0.2682,0.2698,0.272,0.2756,0.2791,0.2856,
                              0.2954,0.3101,0.3155,0.3251,0.3399,0.3542,0.3622,
                              0.3737,0.3878,0.4061,0.4309,0.4676,0.5248,0.6601,
                              0.7145,1.4027,2.945};
float off pf 1101 [24]     = {0.266,0.2681,0.2696,0.2718,0.2754,0.279,0.2854,
                              0.2952,0.3099,0.3153,0.3248,0.3397,0.3539,0.3619,
                              0.3734,0.3874,0.4057,0.4304,0.4669,0.5238,0.6574,
                              0.7105,1.4007,2.945};
```

```
float off standard er [24] = {0.2618,0.2641,0.2656,0.2678,0.2714,0.275,0.2815,
                              0.2922,0.3063,0.3117,0.3212,0.336,0.3501,0.358,
                              0.3694,0.3832,0.4011,0.4251,0.4603,0.5133,0.6297,
                              0.6736,1.3334,2.945};
float off pe1 1 [24]       = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float off pe 11 [24]       = {0.268,0.2701,0.2717,0.274,0.2777,0.2814,0.2881,
                              0.2983,0.3134,0.319,0.3288,0.3441,0.3587,0.367,
                              0.3789,0.3933,0.4122,0.4378,0.4759,0.5359,0.6843,
                              0.7506,1.4043,2.945,};
float off pe 111 [24]      = {0.2682,0.2702,0.2718,0.2742,0.2779,0.2817,0.2884,
                              0.2979,0.3136,0.3191,0.329,0.3442,0.3589,0.3672,
                              0.3791,0.3936,0.4125,0.4382,0.4764,0.5365,0.6856,
                              0.7523,1.4043,2.945};
float off pe 101 [24]      = {0.2622,0.2642,0.2657,0.2681,0.2717,0.2754,0.2819,
                              0.2915,0.3065,0.3119,0.3214,0.3362,0.3503,0.3582,
                              0.3696,0.3835,0.4014,0.4256,0.4608,0.514,0.6312,
                              0.6756,1.3361,2.945};
float off pe 1101 [24]     = {0.2622,0.2642,0.2657,0.2681,0.2718,0.2755,0.282,
                              0.2909,0.3065,0.3119,0.3214,0.3362,0.3503,0.3582,
                              0.3696,0.3835,0.4014,0.4256,0.461,0.5142,0.6313,
                              0.6758,1.3362,2.945};
float off pe 10101 [24]    = {0.2618,0.2641,0.2656,0.2678,0.2714,0.275,0.2815,
                              0.2922,0.3063,0.3117,0.3212,0.336,0.3501,0.358,
                              0.3694,0.3832,0.4011,0.4251,0.4603,0.5133,0.6297,
                              0.6736,1.3334,2.945};

float off standard ws [24] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float off ps1 1 [24]       = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float off ps 11 [24]       = {0.2921,0.2944,0.2961,0.2986,0.3026,0.3066,0.3138,
                              0.3247,0.3411,0.3471,0.3577,0.3742,0.39,0.3989,
                              0.4117,0.4273,0.4476,0.4751,0.5158,0.5791,0.7252,
                              0.7829,1.1854,2.945};
float off ps 111 [24]      = {0.2921,0.2944,0.2961,0.2986,0.3026,0.3066,0.3138,
                              0.3247,0.3411,0.3471,0.3577,0.3742,0.39,0.3989,
                              0.4117,0.4273,0.4476,0.4751,0.5158,0.5791,0.7252,
                              0.7824,1.1294,0};
float off ps 101 [24]      = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float off ps 1101 [24]     = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float off ps 1011 [24]     = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float off ps 10111 [24]    = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float off ps 11011 [24]    = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float off ps 110111 [24]   = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float off ps 1 1 1 [24]    = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float off ps 1 1 11 [24]   = {0.2921,0.2944,0.2961,0.2986,0.3026,0.3066,0.3138,
                              0.3247,0.3411,0.3471,0.3577,0.3742,0.39,0.3989,
                              0.4117,0.4273,0.4476,0.4751,0.5158,0.5791,0.7252,
                              0.7829,1.2304,2.945};
float off ps 11 1 1 [24]   = {0.2921,0.2944,0.2961,0.2986,0.3026,0.3066,0.3138,
                              0.3247,0.3411,0.3471,0.3577,0.3742,0.39,0.3989,
                              0.4117,0.4273,0.4476,0.4751,0.5158,0.5791,0.7252,
                              0.7825,1.1814,2.945};
float off ps 11 1 11 [24]  = {0.2921,0.2944,0.2961,0.2986,0.3026,0.3066,0.3138,
                              0.3247,0.3411,0.3471,0.3577,0.3742,0.39,0.3989,
                              0.4117,0.4273,0.4476,0.4751,0.5158,0.5791,0.7252,
                              0.7825,1.1814,2.945};
*/

/*float off standard fa [24] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                               0};
float off pf 10 [24]       = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float off pf 110 [24]      = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float off pf 101 [24]      = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float off_pf_1101 [24]     = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

float off standard er [24] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float off pe1 1 [24]       = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float off pe 11 [24]       = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float off pe 111 [24]      = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float off pe 101 [24]      = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float off pe 1101 [24]     = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float off_pe_10101 [24]    = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

float off standard ws [24] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float off ps1 1 [24]       = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float off ps 11 [24]       = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float off ps 111 [24]      = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float off ps 101 [24]      = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float off ps 1101 [24]     = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float off ps 1011 [24]     = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float off ps 10111 [24]    = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
float off_ps_11011 [24]    = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
```

```
 float off ps 110111 [24]   = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
 float off ps 1 1 1 [24]    = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
 float off ps 1 1 11 [24]   = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
 float off ps 11 1 1 [24]   = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
 float off_ps_11_1_11 [24]  = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};*/

//-------------------------------------------------------------------------------------------//

 question=0;

 while(question!=-1)
 {
  clrscr();

  control_panel ();//3T

  x=0; y=0; press1=0; press2=0; press3=0; press4=0; press5=0; press6=0;
  erasure bits=0; false alarm bits=0; wrong slot bits=0; pcm bits=0;
  aut pcm bits=0; start number=0; end number=0; extra pulses=0; start=0;
  finish=0; remainder=0; number=0; flag=0; counter=0; bit counter=0;
  row counter=0; row counter2=0; flag er=0; flag fa=0; flag ws=0; limit=0;
  pulse counter=0; pulse counter2=0; zero counter=0; i=0; j=0; k=0; l=0;
  m=0; w=0; p=0; n=0; n1=0; o=0; z=0; l1=0; c=0; question=0;

  a=0; photon energy=0; mark space correction=0; pulse energy=0; energy in frame=0;
  energy per pcm bit=0; B=0; dBm,minimum=0; erasure weight=0; erasure weight2=0;
  false alarm weight=0; false alarm_weight2=0; pasc=0; pasc2=0; pasc3=0; aut_pasc=0;
  aut_pasc2=0; aut_pasc3=0; //53T

  do
  {
   gotoxy(19,50); cout<<"Enter the number of slots please      "<<endl;
   gotoxy(19,10); cin>>x; if (x==-1) break; //2T
  }while (x<=1);

  if (x==-1) break; //T

//-------------------------------------------------------------------------------------------//

  do
  {
   gotoxy(19,50); cout<<"Enter the number of pulses please     "<<endl;
   gotoxy(20,11); cin>>y; if (y==-1) break;
  }while ( (y>x)||(y<=0) );

  if (y==-1) break;

  gotoxy(19,50); cout<<"Which error rate to calculate first?  "<<endl;

  gotoxy(33,23); cin>>press1; //---8*T

  while ( (press1!=1)&&(press1!=2)&&(press1!=3)&&(press1!=-1) )
        {
         gotoxy(19,50);cout<<"Wrong Selection.Please try again      "<<endl;
         gotoxy(33,23);cin>>press1;
        } if (press1==-1) break; //----4*T

  if (press1==1)
   { erasure bits=1; false alarm_bits=0; wrong_slot_bits=0;
     aut_pasc=pascal_calc(x,y);

     if (y!=1) {aut pasc2=pascal_calc (x,y-1);} //or erasure_bits);
     else      {aut_pasc2=1;}

     aut pasc3=0;
     erasure weight=(x-y+1)/2.0;
     false alarm weight=0.0;
     gotoxy(57,13);cout<<aut pasc<<endl;
     gotoxy(56,15);cout<<aut pasc2<<endl;
     gotoxy(60,16);cout<<aut pasc3<<endl;
   }//end of if-else press1(1)
    //------10*T

//-------------------------------------------------------------------------------------------//

  else if (press1==2)
       { false alarm bits=1; erasure bits=0; wrong_slot_bits=0; //2*T
         if ( x==(y+1) ) {
                          aut pasc3=1;
                          aut pasc=x;
                         }//3*T
```

8

```
          else           {
                      aut pasc3=pascal calc (x,y+1);
                          aut_pasc=pascal_calc(x,y);
                     }
          false alarm weight=(y+1.0)/2.0;
          aut pasc2=0;
          erasure weight=0.0;
          gotoxy(57,13);cout<<aut pasc<<endl;
          gotoxy(56,15);cout<<aut pasc2<<endl;
          gotoxy(60,16);cout<<aut pasc3<<endl;
        } //end of if-else press1(2)


  else if (press1==3)
       { wrong slot bits=1; erasure bits=1; false alarm bits=1;
         if ( x==(y+1) ) {
                          aut pasc3=1;
                          aut pasc=x;
                          aut_pasc2=pascal_calc (x,y-1);
                         }
         else if (y==1)  {
                          aut pasc3=pascal calc(x,y+1);
                          aut pasc=pascal_calc(x,y);
                          aut_pasc2=1;
                         }
              else       {
                          aut pasc3=pascal calc(x,y+1);
                          aut pasc=pascal calc(x,y);
                          aut pasc2=pascal_calc (x,y-1);
                         }
         false alarm weight= (y+1.0)/2.0; erasure_weight= (x-y+1)/2.0 ;
         gotoxy(57,13);cout<<aut pasc<<endl;
         gotoxy(56,15);cout<<aut pasc2<<endl;
         gotoxy(60,16);cout<<aut pasc3<<endl;
       }//end of if-else press1(3)

//-------------------------------------------------------------------------------------//

  aut_pcm_bits=0; //T

  for (i=0; i<aut_pasc; i++)
  {
   if (pow(2,aut pcm bits)<aut pasc) {aut pcm_bits++;}
   else                              {break;}
  } //2*T*aut_pasc

  gotoxy(50,14); cout<<aut_pcm_bits<<endl;

  gotoxy(61,10); cout<<pow(2,aut_pcm_bits)<<endl;

  gotoxy(72,10); cout<<pow(2,aut_pcm_bits)-aut_pasc<<endl;

 do
  {
   gotoxy(19,50); cout<<"Enter the Number of PCM bits please   "<<endl;

   gotoxy(12,13); cin>>pcm_bits; if (pcm_bits==-1) break;
  }while(pcm_bits<=0);

  if (pcm_bits==-1) break; //6T

  do
  {
   gotoxy(19,50); cout<<"Enter Pascal's Number please          "<<endl;

   gotoxy(19,12); cin>>pasc; if (pasc==-1) break;
  }while((pasc<=0)||(pasc!=pow(2,pcm_bits)));

  if (pcm_bits==-1) break;

  do
  {
   gotoxy(19,50); cout<<"Enter Erasure Pascal Number please    "<<endl;

   gotoxy(18,15); cin>>pasc2; if (pasc2==-1) break;
  }while((pasc2<0)||((pasc2!=1)&&(y==1)&&(press1==1))||((pasc2!=0)&&(press1==2))||
         ((pasc2==0)&&(press1==1))||((pasc2!=1)&&(press1==3)&&(y==1))||
         ((pasc2==0)&&(press1==3)));

  if (pasc2==-1) break;

//-------------------------------------------------------------------------------------//
```

```
  do
  {
   gotoxy(19,50); cout<<"Enter False Alarm Pascal Number please"<<endl;

   gotoxy(22,16); cin>>pasc3; if (pasc3==-1) break;
  }while((pasc3<0)||((pasc3!=1)&&(y==(x-1))&&(press1==2))||((pasc3!=0)&&(press1==1))||
        ((pasc3==0)&&(press1==2))||((pasc3!=1)&&(press1==3)&&(y==(x-1)))||((pasc3==0)
          &&(press1==3)));

  if (pasc3==-1) break;

  gotoxy(19,50); cout<<"Choose type of Data please           "<<endl;

  gotoxy(33,32); cin>>press2;

  mark_space_correction = x/y;

  photon_energy = ((6.63*pow(10,-34)*3*pow(10,8))/(1.55*pow(10,-6)));

  B=pow(10,9);
  //14*T

  while ( (press2!=1)&&(press2!=2)&&(press2!=3)&&(press2!=4)&&(press2!=5)&&
          (press2!=-1) )
        {
          if ( (press2!=1)&&(press2!=2)&&(press2!=3)&&(press2!=4)&&(press2!=-1) )
          {gotoxy(19,50);cout<<"Wrong Selection.Please try again      "
                    <<endl; gotoxy(33,32);cin>>press2;}
    } //3*T

  if (press2==1) { do { gotoxy(19,50);cout<<"Enter the start number 0<=start"
                        <<endl; gotoxy(10,32);cin>>start_number;
                        if (start number==-1) {break;}
                 }while(start_number!=0); } //2*T

  else if (press2==2) { do {gotoxy(19,50);cout<<"Enter the start number "
                        <<pow(2,pcm bits)-1<<"<=start       "<<endl;
                        gotoxy(10,32);cin>>start_number; if (start_number==-1)
                        {break;}
                       }while(start_number!=(pow(2,pcm_bits)-1)); }

  else if (press2==3) { do {gotoxy(19,50);cout<<"Enter the start number 0<=start"
                            <<endl; gotoxy(10,32);cin>>start_number;
                            if (start number==-1) {break;}
                           }while(start_number!=0); }

  else if (press2==4) { do {gotoxy(19,50);cout<<"Enter the minimum limit 0<=min   "
                            <<endl; gotoxy(62,32);cin>>start_number;
                            if (start number==-1) {break;}
                           }while(start number!=0); if (start number==-1) {break;}
                        do {gotoxy(19,50);cout<<"Enter the maximum limit max<="
                            <<pow(2,pcm bits)<<"      "<<endl; gotoxy(68,32);
                           cin>>end number; if (end number==-1)
                           {break;} }while (end_number!=pow(2,pcm_bits));

//range at least pow(2,pcm bits)+1 i.e. 12-2 start from
//0 range=0-64 (and not 63-infinite loop-)

  if ( (press2==-1)||(start_number==-1)||(end_number==-1)) break; //T

  mppm = new int [x*pow(2,pcm_bits)]; //T

//-------------------------------------------------------------------------------------//


  if (mppm==NULL) {gotoxy(19,50);cout<<"NOT ENOUGHT MEMORY FOR MPPM ARRAY!!!! "
                                     <<endl;} //T
  else { for (i=0; i<x*pow(2,pcm bits); i++) mppm[i] = 0; //x*pow(2,pcm_bits)*T
         for (i=0; i<y; i++)      mppm[i] = 1; //y*T

         i=0; j=0; l=0; m=0; extra_pulses=0; k=1; c=-1; //T

         for (m=0; m<c+pow(2,pcm bits); m++)
         { for (i=0; i<x-1; i++) if ( mppm[i+(k-1)*x]==1 ) j=i; //(x-1)*2*T

                  for (i=1; i<=y; i++)
           { if (mppm[k*x-i]==0) break;               //T
                else if (mppm[k*x-i]==1) extra pulses++; //2*T
           }

               if (extra_pulses==y) break; //T
```

```
                if ( j!=(x-1) ) //T
          { for (i=0; i<j; i++) mppm[i+k*x]=mppm[i+(k-1)*x]; //j*T

            mppm[j+k*X+1]=1; //T

            for ( i=1; i<=extra pulses; i++)
                    mppm[j+k*x+1+i]=1;//extra_pulses*T

            l=0;   extra pulses=0; //T
          }
             else if ( ( j==(x-1) ) && ( extra pulses!=0 ) ) //T
          { l=extra pulses; k--; extra_pulses=0; c++;}    //T

                k++; //T
        }
      }//if-else statement for mppm null
  //(c+pow(2,pcm_bits))*

//-------------------------------------------------------------------------------------//

  start=0; finish=0; number=0; remainder=0; flag=0; k=1; //T

  if (press2==1)       {(start=start number); (finish=pow(2,pcm bits)+start number-1);}
  else if (press2==2)  {(start=start number); (finish=start number-pow(2,pcm bits)+1);}
  else if (press2==3)  {(start=start_number); (finish=pow(2,pcm_bits)+start_number-1);}

  else if (press2==4)  {( srand(time(NULL)) ); (start=start_number);
                        (finish=end_number);}//8*T

  data = new int [pcm_bits*pow(2,pcm_bits)]; //T

  if ( (press2==4)||(press2==5) ) {random = new int [pow(2,pcm bits)];
                                   if (random==NULL) {gotoxy(19,50);
                                   cout<<"NOT ENOUGHT MEMORY FOR RANDOM ARRAY!!!  "
                                   <<endl;
                                   else {for (i=0; i<pow(2,pcm_bits); i++)
                                        {random[i]=0;} }
                } //3*T

  if (data==NULL) {gotoxy(19,50);cout<<"NOT ENOUGHT MEMORY FOR DATA ARRAY!!! "<<endl;}
  else { for (i=0; i<pcm_bits*pow(2,pcm_bits); i++) {data[i]=0;}

        if (press2==1) //3T
        { for (m=start; m<=finish; m++)
           { number=m;
             for (i=1; i<=pcm bits; i++) //5T*pcm_bits*(finish-star)
             { remainder=number%2;
               number/=2;
               data[k*pcm_bits-i]=remainder;
             }  k++;
           }
        }
        else if (press2==2)
             { for (m=start; m>=finish; m--)
                  { number=m;
                    for (i=1; i<=pcm bits; i++)
                    { remainder=number%2;
                        number/=2;
                        data[k*pcm_bits-i]=remainder;
                    } k++;
                  }
             }
        else if (press2==3)
             { for (m=start; m<=finish; m++)
                  { number = m;
                    for (i=1; i<=pcm bits; i++)
                    { remainder=number % 2;
                      number/=2;
                      data[k*pcm_bits-i]=remainder;
                    } k++;
                  }

               k=1;
              for (m=start; m<=finish; m++)
                  { for (i=1; i<=pcm bits; i++)
                    { if (i!=pcm bits)
                      (data[k*pcm bits-i]=data[k*pcm bits-i]^data[k*pcm bits-i-1]);
                      else (data[(k-1)*pcm_bits]=data[(k-1)*pcm_bits]^0);
                    } k++;
                  }
             }
```

```
//-------------------------------------------------------------------------------------//

         else if (press2==4)
              { for (m=0; m<pow(2,pcm bits); m++)
                  //4*pow(2,pcm bits)+3*pcm bits+4*pow(2,pcm_bits)^2
                { number=start+rand()%finish;
                  random[m]=number;
                  //gotoxy(2,58+m);cout<<number<<endl;
                  for (i=0; i<pow(2,pcm bits); i++)
                  if ( ( (random[m]==random[i])&&(i!=m) )||(random[m]>finish)
                       ||(random[m]<start) ) { if ( (m!=0)&&(k!=0) )
                         {m--; flag=1; k--;}

                                          else                  {flag=1;}

                                       }
                  if ( flag!=1 )
                  { for (i = 1; i <= pcm bits; i++)
                    { //gotoxy(38,58+m);cout<<number<<endl;
                      remainder=number%2;
                      number/=2;
                      data[k*pcm_bits-i]=remainder;
                    }
                  } flag=0; k++;
                }

     gotoxy(19,50);cout<<"DISPLAY RANDOM NUMBERS?PRESS9 (8-EXIT)    "<<endl;
         gotoxy(64,47);cin>>press3;

                while ( (press3!=9)||(press3!=8)||(press3!=-1) )
                { if (press3==9)                          {
                                                      if (pasc<280)
{gotoxy(0,80); cout<<"RANDOM NUMBERS "<<endl; print random array(data,pow(2,pcm bits),
pcm bits,x,y,aut pasc,aut pasc2,aut pasc3,aut pcm bits,pcm bits,pasc,pasc2,pasc3,
start number,end number,press1,press2,press3,press4,press5);}

else          {gotoxy(0,60); cout<<"RANDOM NUMBERS "<<endl;
print random array2(data,pow(2,pcm bits),pcm bits,x,y,aut pasc,aut pasc2,aut pasc3,
aut pcm bits,pcm_bits,pasc,pasc2,pasc3,start_number,end_number,press1,press2,press3,
press4,press5);}

//-------------------------------------------------------------------------------------//


fstream file_op2("random.txt",ios::out);
press6=0;
do
{
 gotoxy(19,50); cout<<"DO YOU WANT TO SAVE THE DATA IN A FILE?                "<<endl;
 gotoxy(33,47);cin>>press6;
if (press6==-1) {break;}
if (press6==2)  {break;}//add download operation here
if (press6==3)  {break;}
}while(press6!=1);

if (press6==3) {file_op2.close(); break;}
if (press6==1) {
        for (i=0; i<pow(2,pcm_bits); i++)
        {
         file op2<<"\n";
         file_op2<<random[i]<<"  ";
        }
        file_op2.close();
           }
          break;
         }
          else if ( (press3==8)||(press3==-1) ) {break;}
           else {gotoxy(19,50);
                 cout<<"WRONG SELECTION.TRY AGAIN                 "<<endl;
                 gotoxy(64,47);cout<<"             "<<endl;
gotoxy(64,47);cin>>press3;}
          }
            }

//-------------------------------------------------------------------------------------//

             if (press2==4) {delete [] random; random=0;}

             if (press3==-1) {break;}

             if (press6==-1) {break;}
```

```
        else if (press2==5) //21*T
             {
              for (m=0; m<pow(2,pcm_bits); m++)
              {
               file op2>>number; //input from file pointer or standard input
               random[m]=number;

               if ( flag!=1 ) {
                               for (i = 1; i <= pcm bits; i++)
                               {//gotoxy(38,58+m);cout<<number<<endl;
                               remainder=number%2;
                               number/=2;
                               data[k*pcm_bits-i]=remainder;
                              }
              } flag=0; k++;
             }
        } //4*pow(2,pcm_bits)*3*pcm_bits

        if (press2==5) {delete [] random; random=0;}

        clrscr2();

       }//if-else for data||random null

  mppm_data = new int [(x+pcm_bits)*pow(2,pcm_bits)];//3T

  if (mppm data==NULL)
  {gotoxy(19,50);cout<<"NOT ENOUGHT MEMORY FOR MPPM-DATA ARRAY!!  "<<endl;}
  else { for (i=0; i<(x+pcm_bits)*pow(2,pcm_bits); i++) mppm_data[i] = 0;//2*T

        k=0;

//-------------------------------------------------------------------------------------//

        for (m=0; m<pow(2,pcm_bits); m++)
        {
         for (i=0; i<x; i++) { mppm_data[k*(x+pcm_bits)+i] = mppm[k*x+i]; }
         k++; l++;
        } //pow(2,pcm_bits)*(x+2)*T

        k=0; l=0; //T
        for (m=0; m<pow(2,pcm_bits); m++)
        {
         for (i=x; i<x+pcm bits; i++) { mppm data[k*(x+pcm bits)+i] =
                                        data[(l*pcm_bits)-x+i]; }
         k++; l++;
        }

       }//if-else statement for dynamic memory allocation safe failure

       //((x+pcm bits)*pow(2,pcm bits)*T)+(2*T)+((x*T+T)*pow(2,pcm_bits))+
      //(((x+pcm_bits)*T+T)*pow(2,pcm_bits))

  if (y==1)
  {
   if (press1==1)
   {
    weighted er mppm data = new int [(x+pcm bits)];
    mppm er = new int [(x+1)*pow(2,pcm_bits)];
    sequence = new float [3];

    if ((weighted er mppm data==NULL)||(mppm er==NULL)||(sequence==NULL))
       {gotoxy(19,50);cout<<"NOT ENOUGHT MEMORY FOR ARRAY!!!!!!!!!!"<<endl;}
    {

     k=0; flag_er=1; counter=0; l=0;

     for (i=0; i<(x+pcm bits); i++) {weighted er mppm data[i]=0;}
     for (i=x; i<(x+pcm bits); i++) {weighted er mppm data[i]=2;}
     for (i=0; i<3; i++)            {sequence[i]=0;} //10T


//-------------------------------------------------------------------------------------//


     for (i=0; i<pow(2,pcm bits)*(x+pcm bits); i++) {mppm_er[i]=0;}
        //pow(2,pcm_bits)*(x+pcm_bits)*T

     for (i=0; i<pow(2,pcm_bits); i++) {mppm_er[(k*(x+y))+i]=1; k++;}
       //pow(2,pcm_bits)*T
```

```
      k=1;

      for (i=0; i<pow(2,pcm_bits); i++) {mppm_er[(k*(x+y))-1]=pcm_bits; k++;}
        //pow(2,pcm_bits)*T

      k=0;

      press4=0; //3*T

      do
      {
       gotoxy(19,50);   cout<<"DO YOU WANT TO DISPLAY THE SEQUENCES?   "<<endl;
       gotoxy(10,44);cin>>press4;
       if (press4==2)   {break;}
       if (press4==-1)  {break;} //3T
      }while(press4!=1);

      if (press4==-1) {break;}

//-------------------------------------------------------------------------------------//

      if (press4==1) { //2T
               for (i=2; i<pow(2,pcm_bits); i++) {counter=counter+3;}

               sequence[2]=(counter/((pow(2,pcm_bits)*pcm_bits)));

                        for (i=0; i<pow(2,pcm bits); i++) { if (mppm data[(k*pcm bits)+
                            ((k+1)*(x-1))+i]==1) {l++;} k++;} //2*pow(2,pcm_bits)*T

                       k=0; //2*T

                       if (l==0) {sequence[2]=sequence[2]+((2*pcm_bits)
                                 /((pow(2,pcm_bits)*pcm_bits)));}
              else       {
                      sequence[1]=(((1/pow(2,pcm bits))*pcm bits)
                                  /(((pow(2,pcm bits)*pcm bits))));
                 sequence[2]=sequence[2]+((((pow(2,pcm bits)-1)
                                 /pow(2,pcm bits))*pcm_bits)/(((pow(2,pcm_bits)
                                 *pcm_bits))));
                    }
                       l=0; k=0; counter=0;

                       for (i=0; i<pow(2,pcm bits); i++) {
                                                          if (mppm data[(k*pcm bits)+
                                                          ((k+1)*(x-1))+i-1]==1)
                                                          {l++;} k++;}

                       if (l==0) {sequence[2]=sequence[2]+((pcm bits)/
                                  ((pow(2,pcm_bits)*pcm_bits)));}
              else       {
                      sequence[3]=(((1/pow(2,pcm bits))*pcm bits)
                                  /(((pow(2,pcm bits)*pcm bits))));
                 sequence[2]=sequence[2]+((((pow(2,pcm bits)-1)
                                 /pow(2,pcm bits))*pcm_bits)/(((pow(2,pcm_bits)
                                  *pcm_bits))));
                    }

                       l=0; k=0; counter=0;

//-------------------------------------------------------------------------------------//


                       if (sequence[2]<0) {sequence[2]=sequence[2]*(-1);}
                       if (sequence[0]<0) {sequence[0]=sequence[0]*(-1);}
                       if (sequence[1]<0)
                                          {sequence[1]=sequence[1]*(-1);}

gotoxy(0,57); cout<<"Pe:"<<endl; gotoxy(3,57);cout<<sequence[2]<<endl;
gotoxy(0,58); cout<<"Pe.101:"<<endl; gotoxy(7,58);cout<<sequence[0]<<endl;

gotoxy(0,59); cout<<"Pe.110:"<<endl; gotoxy(7,59);cout<<sequence[1]<<endl; //15*T


    /*if (sequence[2]<=0.0001)      {sequence[2]=0.12*sequence[2];}
    else if (sequence[2]<=0.001)  {sequence[2]=0.27*sequence[2];}
    else if (sequence[2]<=0.01)   {sequence[2]=0.31*sequence[2];}
    else if (sequence[2]<=0.1)    {sequence[2]=0.09*sequence[2];}
    else if (sequence[2]<=0.2)    {sequence[2]=0.05*sequence[2];}
    else if (sequence[2]<=0.3)    {sequence[2]=0.04*sequence[2];}
    else if (sequence[2]<=0.4)    {sequence[2]=0.03*sequence[2];}
    else if (sequence[2]<=0.5)    {sequence[2]=0.02*sequence[2];}
    else if (sequence[2]<=0.6)    {sequence[2]=0.02*sequence[2];}
```

```
    else if (sequence[2]<=0.7)     {sequence[2]=0.01*sequence[2];}
    else if (sequence[2]<=0.8)     {sequence[2]=0.02*sequence[2];}
    else if (sequence[2]<=0.9)     {sequence[2]=0.01*sequence[2];}
    else if (sequence[2]<=1)       {sequence[2]=0.16*sequence[2];}
         else                      {sequence[2]=0*sequence[2];}

        if (sequence[1]<=0.0001)       {sequence[1]=0.12*sequence[1];}
    else if (sequence[1]<=0.001)   {sequence[1]=0.27*sequence[1];}
    else if (sequence[1]<=0.01)    {sequence[1]=0.31*sequence[1];}
    else if (sequence[1]<=0.1)     {sequence[1]=0.09*sequence[1];}
    else if (sequence[1]<=0.2)     {sequence[1]=0.05*sequence[1];}
    else if (sequence[1]<=0.3)     {sequence[1]=0.04*sequence[1];}
    else if (sequence[1]<=0.4)     {sequence[1]=0.03*sequence[1];}
    else if (sequence[1]<=0.5)     {sequence[1]=0.02*sequence[1];}
    else if (sequence[1]<=0.6)     {sequence[1]=0.02*sequence[1];}
    else if (sequence[1]<=0.7)     {sequence[1]=0.01*sequence[1];}
    else if (sequence[1]<=0.8)     {sequence[1]=0.02*sequence[1];}
    else if (sequence[1]<=0.9)     {sequence[1]=0.01*sequence[1];}
    else if (sequence[1]<=1)       {sequence[1]=0.16*sequence[1];}
         else                      {sequence[1]=0*sequence[1];}

//-------------------------------------------------------------------------------------//

        if (sequence[0]<=0.0001)       {sequence[0]=0.12*sequence[0];}
    else if (sequence[0]<=0.001)   {sequence[0]=0.27*sequence[0];}
    else if (sequence[0]<=0.01)    {sequence[0]=0.31*sequence[0];}
    else if (sequence[0]<=0.1)     {sequence[0]=0.09*sequence[0];}
    else if (sequence[0]<=0.2)     {sequence[0]=0.05*sequence[0];}
    else if (sequence[0]<=0.3)     {sequence[0]=0.04*sequence[0];}
    else if (sequence[0]<=0.4)     {sequence[0]=0.03*sequence[0];}
    else if (sequence[0]<=0.5)     {sequence[0]=0.02*sequence[0];}
    else if (sequence[0]<=0.6)     {sequence[0]=0.02*sequence[0];}
    else if (sequence[0]<=0.7)     {sequence[0]=0.01*sequence[0];}
    else if (sequence[0]<=0.8)     {sequence[0]=0.02*sequence[0];}
    else if (sequence[0]<=0.9)     {sequence[0]=0.01*sequence[0];}
    else if (sequence[0]<=1)       {sequence[0]=0.16*sequence[0];}
         else                      {sequence[0]=0*sequence[0];}

    for (i=0; i<24; i++) {standard er[i]=standard er[i]*sequence[2];}
        for (i=0; i<24; i++) {pe1 1[i]=pe1 1[i]*0;}
        for (i=0; i<24; i++) {pe 11[i]=pe 11[i]*sequence[1];}
        for (i=0; i<24; i++) {pe 111[i]=pe 111[i]*0;}
        for (i=0; i<24; i++) {pe 101[i]=pe 101[i]*sequence[0];}
        for (i=0; i<24; i++) {pe 1101[i]=pe 1101[i]*0;}
        for (i=0; i<24; i++) {pe_10101[i]=pe_10101[i]*0;}

        for (i=0; i<24; i++) {standard er[i] = standard er[i] - off standard er[i]
                                             + pe1 1[i] - off pe1 1[i] + pe 11[i] -
                                               off pe 11[i] + pe 111[i] - off pe 111[i]
                                             + pe 101[i] - off pe 101[i] + pe 1101[i]
                                             - off pe 1101[i] + pe_10101[i] -
                                               off_pe_10101[i];}

//-------------------------------------------------------------------------------------//

    for (i=0; i<24; i++) {
                          if (standard er[i]<0) {standard er[i]=standard er[i]
                              *(-1);}
                   gotoxy(0,65+i); cout<<"\n"<<standard_er[i]<<endl;
                  }


        for (i=0; i<24; i++) {
                   if (standard er[i]<0) {minimum=0;}
                   else                  {minimum=standard_er[i];}

                              pulse energy = minimum * photon energy;
                   energy in frame = pulse energy * mark space correction;
                              energy per pcm bit = (energy in frame/pcm bits);
                              dBm = 10 * (log((energy_per_pcm_bit*B)/pow(10,-3))
                              /log(10));
                   gotoxy(0,95+i); cout<<"\n"<<dBm<<endl;
                                    }*/
                     }

//-------------------------------------------------------------------------------------//

          gotoxy(19,50);cout<<"DISPLAY INPUT ARRAYS? PRESS1,2 ELSE 8 "<<endl;
          gotoxy(64,47);cin>>press3;

          while ( (press3!=1)||(press3!=2)||(press3!=8)||(press3!=-1) )
```

```
                {
             if (press3==1) {
   clrscr2(); gotoxy(0,56); cout<<"ERASURE MPPM-DATA ARRAY                          "
<<endl;
   print input arrays(mppm data,pow(2,pcm bits),x+pcm bits,x,y,aut pasc,aut pasc2,
                      aut pasc3,aut pcm bits,pcm bits,pasc,pasc2,pasc3,start_number,
                      end_number,press1,press2,press3,press4,press5);
   press6=0;
    gotoxy(19,50); cout<<"DO YOU WANT TO SAVE THE DATA IN A FILE?                "
        <<endl; gotoxy(33,47);cin>>press6;
    if (press6==-1) {break;}
        if (press6==3) {file_op1.close(); break;}
    if (press6==1) {
                  k=1;
           for (i=0; i<(x+pcm_bits)*pow(2,pcm_bits); i++)
           {
            if (i==k*(x+pcm bits)) {file op1<<"\n"; k++;}
            file_op1<<mppm_data[i]<<"  ";
           }
           file op1.close();
              }


              break;
                   }
             else if (press3==2) {
                      clrscr2();
                                        gotoxy(0,56);
                       cout<<"WEIGTHED ERASURE ARRAY                          "<<endl;
                      print input arrays(weighted er mppm data,1,
      x+pcm bits,x,y,aut pasc,aut pasc2,aut pasc3,aut pcm bits,pcm bits,pasc,pasc2,
      pasc3,start number,end number,press1,press2,press3,press4,press5);


//-------------------------------------------------------------------------------------//

  do
  { press6=0;
    gotoxy(19,50); cout<<"DO YOU WANT TO SAVE THE DATA IN A FILE?                "
    <<endl; gotoxy(33,47);cin>>press6;
    if (press6==-1) {break;}
    if (press6==2)  {break;}//add download operation here
    if (press6==3)  {break;}
  }while(press6!=1);

  if (press6==3) {file_op1.close(); break;}
  if (press6==1) {
         k=1;
         for (i=0; i<(x+pcm_bits)*1; i++)
         {
          if (i==k*(x+pcm bits)) {file op1<<"\n"; k++;}
          file_op1<<weighted_er_mppm_data[i]<<"  ";
         }
        file op1.close();
       }


       break;
          }
          else if ( (press3==8)||(press3==-1) ) {break;}
           else                                 {gotoxy(19,50);
              cout<<"WRONG SELECTION.TRY AGAIN                      "<<endl;
              gotoxy(64,47);cin>>press3;}
                }

          if (press3==-1) {break;}
          if (press6==-1) {break;}



//-------------------------------------------------------------------------------------//

          clrscr2();

          //delete [] mppm data; mppm data=0;
          //delete [] weighted_er_mppm_data; weighted_er_mppm_data=0;

          delete [] mppm; mppm=0;
          delete [] data; data=0;
```

```
          delete [] sequence; sequence=0; //40*T
      }
    }//end of if-press1(1) and y==1

    else if (press1==2)
    {
      mppm = new int [x*pasc3]; data = new int [pcm bits*pasc3];
      weighted_fa_mppm_data = new int [(x+pcm_bits)*pasc3];


      if (mppm==NULL) {
            gotoxy(19,50);cout<<"NOT ENOUGHT MEMORY FOR MPPM3 ARRAY!!!!        "
            <<endl;}
      else { gotoxy(56,11); cout<<erasure weight<<endl;
            gotoxy(60,12); cout<<false_alarm_weight<<endl;

            for (i=0; i<x*pasc3; i++) mppm[i] = 0;   //7*T
            for (i=0; i<y+false alarm_bits; i++) mppm[i] = 1;
            //(y+false_alarm_bits)*T

            i=0; j=0; l=0; m=0; extra pulses=0; k=1;
            for (m=0; m<2*pasc3; m++)
            { for (i=0; i<x-1; i++) if ( mppm[i+(k-1)*x]==1 ) j=i;

              for (i=1; i<=y+false alarm bits; i++)
              { if (mppm[k*x-i]==0) break;
                else if (mppm[k*x-i]==1) extra pulses++;
              } if (extra_pulses==y+false_alarm_bits) break;

              if ( j!=(x-1) )
              { for (i=0; i<j; i++) mppm[i+k*x]=mppm[i+(k-1)*x];
                 mppm[j+k*x+1]=1;

                for (i=1; i<=extra pulses; i++)
                 mppm[j+k*x+1+i]=1;

                l=0; extra pulses=0;
              }
              else if ( (j==(x-1) ) && (extra pulses!=0) )
                     {l=extra_pulses; k--; extra_pulses=0;}

              k++;
            } //2*pasc3*T

          }//end of if-else mppm null statement
//-------------------------------------------------------------------------------------------//
    false alarm weight2=0;
    if (data==NULL) {gotoxy(19,50);
    cout<<"NOT ENOUGHT MEMORY FOR DATA3 ARRAY!!!!      "<<endl;}
    else { for (i=0; i<pcm bits*pasc3; i++) data[i] = 0;
            w=0; row counter=0; bit counter=0; k=1;
            for (l=1; l<=pasc3; l++)
            { for (j=x; j<(x+pcm bits); j++)
              { for (m=0; m<pow(2,pcm bits); m++)
                { for (i=0; i<x; i++) { if ( mppm[i+(l-1)*x]==
                       mppm_data[i+(k-1)*(pcm_bits+x)] ) row_counter++; }

                  if ( ( row counter==(x-false alarm bits) ) &&
                  ( mppm data[j+(k-1)*(pcm bits+x)]==1 ) ) bit counter++;
                  if ( row counter==(x-false_alarm_bits) ) false_alarm_weight2++;
                     row_counter=0; k++;
                } k=1;
                if      ( bit counter > (false_alarm_weight2/2.0) )
                  data[w+(l-1)*pcm bits]=1;
                else if ( bit counter < (false_alarm_weight2/2.0) )
                  data[w+(l-1)*pcm bits]=0;
                else if ( bit counter = (false_alarm_weight2/2.0) )
                  data[w+(l-1)*pcm bits]=2;
                bit counter=0; w++; false_alarm_weight2=0;
              } k=1; w=0;
            }

          }//end of if-else data null statement

    if (weighted fa mppm data==NULL) {gotoxy(19,50);
        cout<<"NOT ENOUGHT MEMORY FOR WEIGHT-FA ALARM     "<<endl;}
    else { for (i=0; i<(x+pcm_bits)*pasc3; i++) weighted_fa_mppm_data[i] = 0;

            k=0;
            for (m=0; m<pasc3; m++)
            { for (i=0; i<x; i++)
              { weighted_fa_mppm_data[k*(x+pcm_bits)+i] = mppm[k*x+i]; }
```

17

```
            k++; l++;
          }

          k=0; l=0;
          for (m=0; m<pasc3; m++)
          { for (i=x; i<x+pcm bits; i++)
            { weighted fa mppm data[k*(x+pcm bits)+i] =
                    data[(l*pcm_bits)-x+i]; }
            k++; l++;
          }

         }//end of if-else weighted_fa_mppm_data null statement

    mppm_fa = new int [(2*x+pcm_bits-y)*pow(2,pcm_bits)];

//-------------------------------------------------------------------------------------//

    if (mppm fa==NULL) {gotoxy(19,50);
    cout<<"NOT ENOUGHT MEMORY FOR MPPM-FA ARRAY    "<<endl;}
    else { for (i=0; i<(2*x+pcm_bits-y)*pow(2,pcm_bits); i++) mppm_fa[i] = 0;

           flag_fa=1;

           k=0;
           for (m=0; m<pow(2,pcm bits); m++)
           { for (i=0; i<x; i++) mppm fa[k*(2*x+pcm bits-y)+i]
           =mppm data[k*(x+pcm bits)+i];
                                k++;
           }

           l=0; k=0; j=0; row counter=0; bit counter=0; w=0;
           for ( m=0; m<pow(2,pcm bits); m++)
           { for (n=0; n<pasc3; n++)
             { for (i=0; i<x; i++) if ( mppm data[i+l*(x+pcm bits)]==
             weighted fa mppm data[i+k*(x+pcm bits)] ) row counter++;
                   if ( row counter==(x-false alarm bits) )
                   { for ( i=x; i<x+pcm bits; i++)
                     { mppm fa[i+l*(2*x+pcm bits-y)]=(
                                 mppm data[i+l*(x+pcm bits)]
                          ^weighted_fa_mppm_data[i+k*(x+pcm_bits)] );

                       if ( mppm fa[i+l*(2*x+pcm bits-y)]!=0 )
                                          bit counter++;
                       else bit counter=bit counter;
                     } mppm fa[(j+1)*(x+pcm bits)+j*(x-y)+w]=
                           mppm fa[(j+1)*(x+pcm bits)+j
                           *(x-y)+w]+bit counter;
                     for (i=x; i<x+pcm_bits; i++) mppm_fa[i+l*(2*x+pcm_bits-y)]=0;
                     w++;
                   }    row counter=0; bit counter=0; k++;
             } l++; k=0; j++; bit_counter=0; w=0;
           }

           press5=1;

//-------------------------------------------------------------------------------------//

       press4=0;
           gotoxy(19,50);
           cout<<"DO YOU WANT TO DISPLAY THE SEQUENCE RESULTS?      "<<endl;
           gotoxy(10,44);cin>>press4;

           while ( (press4!=1)||(press4!=2)||(press4!=-1) )
           { if (press4==-1)                          {break;}
             else if ( (press4==1)||(press4==2) ) {break;}
             else                                 {gotoxy(19,50);
                   cout<<"WRONG SELECTION.TRY AGAIN                "<<endl;
                   gotoxy(10,44);cout<<"          "<<endl;
             gotoxy(10,44);cin>>press4;}
           }

           if (press4==-1) {break;}

           l1=0; zero counter=0;
           for ( m=0; m<pow(2,pcm_bits); m++)
           {
            for ( i=0; i<2; i++) {
                          if (mppm fa[i+m*(2*x+pcm_bits-y)]==0)
                    {zero counter++;}
                          }
            if (zero counter==2) {break;}
            else                 {zero_counter=0; l1++;}
```

```
            }

            bit counter=0; row counter=0; i=0;
            for ( m=0; m<pow(2,pcm bits); m++)
            {  if (mppm_fa[i+m*(2*x+pcm_bits-y)]==1) {bit_counter++;}//1-bit_counter=11//
               i=x-1;
               if (mppm_fa[i+m*(2*x+pcm_bits-y)]==1) {row_counter++;}//1-row_counter=9//
               i=0;
            }

//-------------------------------------------------------------------------------------//

             sequence = new float [2*(y+1)];
             if (sequence==NULL) {gotoxy(19,50);
             cout<<"NOT ENOUGHT MEMORY FOR FA SEQUENCE ARRAY!!!!      "<<endl;}
             else          {for (i=0; i<2*(y+1); i++) sequence[i] = 0;}

             zero_counter=0; pulse_counter=0; l=0; j=0; flag=1; limit=0;

             for (w=0; w<y+1; w++)
             {
              for (n=0; n<pow(2,pcm_bits); n++)
              {
               for (i=0; i<x; i++)
          {
    if ( (mppm_fa[i+l*(2*x+pcm_bits-y)]==0)&&(i==0)&&(w==0) )
        {
     zero counter=0; pulse counter=0;
     if (mppm fa[i+l*(2*x+pcm bits-y)+1]==0) {
     sequence[1]=sequence[1]+(mppm fa[l*(2*x+pcm_bits-y)+x+pcm_bits+j]
         *(row counter/pow(2,pcm bits)));
     sequence[0]=sequence[0]+(mppm fa[l*(2*x+pcm bits-y)+x+pcm_bits+j]
         *((pow(2,pcm bits)-row counter)/pow(2,pcm bits)));
    }
    else                                    {
     sequence[y+2]=sequence[y+2]+(mppm fa[l*(2*x+pcm_bits-y)+x+pcm_bits+j]
         *(row counter/pow(2,pcm bits)));
     sequence[0]=sequence[0]+(mppm fa[l*(2*x+pcm bits-y)+x+pcm_bits+j]
         *((pow(2,pcm_bits)-row_counter)/pow(2,pcm_bits)));
     }
      j++; flag=0;
    }
    else if ( (mppm_fa[i+l*(2*x+pcm_bits-y)]==0)&&(i!=0)&(i!=(x-1)) ){

//-------------------------------------------------------------------------------------//

    zero counter=0; pulse_counter=0;

    for (m=i-1; m>=0; m--)    {
    if (mppm fa[m+l*(2*x+pcm bits-y)]==1) {pulse counter++;}
    else                                  {flag=0; break;}
    }
    if ( (pulse counter==limit)&&(flag==0) ) {
    if (pulse counter==0)                    {sequence[0]=sequence[0]
                                                  +mppm fa[l*(2*x+pcm_bits-y)+
                                                  x+pcm_bits+j];}
    else {
    if (mppm fa[i+l*(2*x+pcm bits-y)+1]==0) {sequence[limit]=sequence[limit]+
                                                 mppm fa[l*(2*x+pcm_bits-y)+
                                                 x+pcm bits+j];}
    else                                   {sequence[limit+y+1]=
                                               sequence[limit+y+1]
                                               +mppm fa[l*(2*x+pcm_bits-y)+x
                                               +pcm_bits+j];}
    }
    }



    else if ( (pulse counter==limit)&&(flag==1) ) {
    if (mppm fa[i+l*(2*x+pcm bits-y)+1]==0) {
    sequence[limit+1]=sequence[limit+1]+(mppm fa[l*(2*x+pcm bits-y)+x+
                                          pcm bits+j]*(row counter/
                                          pow(2,pcm bits)));
    sequence[limit]=sequence[limit]+(mppm fa[l*(2*x+pcm bits-y)+x+pcm bits+j]*
                                    ((pow(2,pcm bits)-row counter)/
                                     pow(2,pcm_bits)));
    }
    else                                   {
    sequence[limit+y+2]=sequence[limit+y+2]+(mppm fa[l*(2*x+pcm_bits-y)+x+
                                              pcm bits+j]
                                              *(row_counter/pow(2,pcm_bits)));
```

```
    sequence[limit+y+1]=sequence[limit+y+1]+(mppm fa[l*(2*x+pcm_bits-y)+x+
                                              pcm bits+j]
                                             *((pow(2,pcm bits)-row_counter)/
                                              pow(2,pcm_bits)));
     }
      }
//-------------------------------------------------------------------------------------//

    j++; flag=0;

    }
    else if ( (mppm fa[i+l*(2*x+pcm bits-y)]==0)&&(i==(x-1)) )       {
    zero counter=0; pulse_counter=0;

    for (m=i-1; m>=0; m--)    {
    if (mppm fa[m+l*(2*x+pcm bits-y)]==1) {pulse counter++;}
    else                                  {break;}
    }
    if ( (pulse counter==limit)&&(x!=(y+1)) )      {
    if (pulse counter==0) {sequence[0]=sequence[0]+mppm fa[l*
                                        (2*x+pcm bits-y)+x+pcm bits+j]; flag=0;}
    else {

    sequence[limit+y+1]=sequence[limit+y+1]+(mppm fa[l*(2*x+pcm bits-y)+x+
                                    pcm bits+j]*(bit counter/pow(2,pcm bits)));
    sequence[limit]=sequence[limit]+(mppm fa[l*(2*x+pcm bits-y)+x+pcm bits+j]*
                                ((pow(2,pcm_bits)-bit_counter)/pow(2,pcm_bits)));
    }
    }
    else if ( (pulse counter==limit)&&(x==(y+1)) ) {
    sequence[limit+1]=sequence[limit+1]+(mppm fa[l*(2*x+pcm bits-y)+x+pcm bits+j]
                                                *(row counter/pow(2,pcm_bits)));
    sequence[limit+y+1]=sequence[limit+y+1]+(mppm fa[l*(2*x+pcm bits-y)+x
                                                 +pcm bits+j]*
                                                (bit counter/pow(2,pcm bits)));
    sequence[limit]=sequence[limit]+(mppm fa[l*(2*x+pcm bits-y)+x+pcm bits+j]*
                                    ((pow(2,pcm bits)-bit_counter-row_counter)/
                                     pow(2,pcm_bits)));
    }
     j++; flag=0;
      }
    if (n==pasc)          {break;}
     }l++; j=0; if (l<=l1) {flag=1;}
    else        {flag=0;}
        if (n==pasc)                                                    {break;}
            }l=0; j=0; flag=1; limit++;
           } //56*(y+1)*pow(2,pcm_bits)*x

//-------------------------------------------------------------------------------------//

    for (i=0; i<2*(y+1); i++) {sequence[i]=(sequence[i]/((x-y)*
                                  pcm_bits*pow(2,pcm_bits)));} //2*(y+1)*T

    if ( (press4==1)&&(press5==2) ) {
    gotoxy(0,57); cout<<"Pf:"<<endl; gotoxy(3,57);cout<<sequence[0]<<endl;
    for (i=0; i<(y+1); i++) { gotoxy(0,58+i); cout<<"Pf.10:"<<endl;
                              gotoxy(6,58+i);cout<<sequence[i+1]<<endl;}
    for (i=0; i<y; i++) { gotoxy(0,58+y+1+i); cout<<"Pf.101:"<<endl;
                              gotoxy(7,58+y+1+i);cout<<sequence[i+y+2]<<endl;}


    k=1; n=0; w=0; z=0;
    for (i=0; i<2*(y+1); i++) {
    gotoxy(0+w,87+n); cout<<sequence[i]<<endl;

    if (sequence[i]!=0) {w=w+13;}
    else                {w=w+5;}
    if (i==k*y+z) {k++; n++; w=0; z++;}
    }
    delete [] sequence; sequence=0;
    } //7*T

    if (press5==1)
    {
    sequence2 = new float [5];
    if (sequence2==NULL) {gotoxy(19,50);
          cout<<"NOT ENOUGHT MEMORY FOR FA SEQUENCE ARRAY!!!!      "<<endl;}
            else                {for (i=0; i<5; i++) sequence2[i] = 0;}

//-------------------------------------------------------------------------------------//
```

```
            for (i=0; i<2; i++)       {sequence2[i]=sequence[i];}
        sequence2[3]=sequence[y+2];

        for (i=2; i<y+2; i++)   {sequence2[2]=sequence2[2]+sequence[i];}
        for (i=y+3; i<2*(y+1); i++) {sequence2[4]=sequence2[4]+sequence[i];}

            sequence2[2]= (pow(2,pcm bits)-1)/pow(2,pcm_bits)*(mppm_fa[x+pcm_bits]
            /(pow(2,pcm bits)*pcm bits));
            sequence2[3]= (pow(2,pcm bits)-1)/pow(2,pcm_bits)*
            (mppm fa[(2*(x+pcm bits))+x+1]
            /(pow(2,pcm_bits)*pcm_bits));

            if (press4==1) {
                    if (sequence2[0]<0) {sequence2[0]=sequence2[0]*(-1);}
                        if (sequence2[1]<0) {sequence2[1]=sequence2[1]*(-1);}
                    if (sequence2[2]<0) {sequence2[2]=sequence2[2]*(-1);}
                        if (sequence2[3]<0) {sequence2[3]=sequence2[3]*(-1);}
                    if (sequence2[4]<0) {sequence2[4]=sequence2[4]*(-1);}

    gotoxy(0,57); cout<<"Pf:"<<endl; gotoxy(3,57);cout<<sequence2[0]<<endl;
    gotoxy(0,58); cout<<"Pf.10:"<<endl; gotoxy(6,58);cout<<sequence2[1]<<endl;

        gotoxy(0,59); cout<<"Pf.110:"<<endl; gotoxy(7,59);cout<<sequence2[2]<<endl;

    gotoxy(0,60); cout<<"Pf.101:"<<endl; gotoxy(7,60);cout<<sequence2[3]<<endl;

    gotoxy(0,61); cout<<"Pf.1101:"<<endl; gotoxy(8,61);cout<<sequence2[4]<<endl;


//-------------------------------------------------------------------------------------//

    /*if (sequence[4]<=0.0001)      {sequence[4]=0.12*sequence[4];}
    else if (sequence[4]<=0.001)  {sequence[4]=0.27*sequence[4];}
    else if (sequence[4]<=0.01)   {sequence[4]=0.31*sequence[4];}
        else if (sequence[4]<=0.1)    {sequence[4]=0.09*sequence[4];}
        else if (sequence[4]<=0.2)    {sequence[4]=0.05*sequence[4];}
    else if (sequence[4]<=0.3)    {sequence[4]=0.04*sequence[4];}
    else if (sequence[4]<=0.4)    {sequence[4]=0.03*sequence[4];}
    else if (sequence[4]<=0.5)    {sequence[4]=0.02*sequence[4];}
    else if (sequence[4]<=0.6)    {sequence[4]=0.02*sequence[4];}
    else if (sequence[4]<=0.7)    {sequence[4]=0.01*sequence[4];}
        else if (sequence[4]<=0.8)    {sequence[4]=0.02*sequence[4];}
        else if (sequence[4]<=0.9)    {sequence[4]=0.01*sequence[4];}
        else if (sequence[4]<=1)      {sequence[4]=0.16*sequence[4];}
         else                     {sequence[4]=0*sequence[4];}


    if (sequence[3]<=0.0001)      {sequence[3]=0.12*sequence[3];}
        else if (sequence[3]<=0.001)  {sequence[3]=0.27*sequence[3];}
    else if (sequence[3]<=0.01)   {sequence[3]=0.31*sequence[3];}
    else if (sequence[3]<=0.1)    {sequence[3]=0.09*sequence[3];}
        else if (sequence[3]<=0.2)    {sequence[3]=0.05*sequence[3];}
        else if (sequence[3]<=0.3)    {sequence[3]=0.04*sequence[3];}
        else if (sequence[3]<=0.4)    {sequence[3]=0.03*sequence[3];}
        else if (sequence[3]<=0.5)    {sequence[3]=0.02*sequence[3];}
    else if (sequence[3]<=0.6)    {sequence[3]=0.02*sequence[3];}
    else if (sequence[3]<=0.7)    {sequence[3]=0.01*sequence[3];}
    else if (sequence[3]<=0.8)    {sequence[3]=0.02*sequence[3];}
    else if (sequence[3]<=0.9)    {sequence[3]=0.01*sequence[3];}
    else if (sequence[3]<=1)      {sequence[3]=0.16*sequence[3];}
         else                     {sequence[3]=0*sequence[3];}

       if (sequence[2]<=0.0001)      {sequence[2]=0.12*sequence[2];}
       else if (sequence[2]<=0.001)  {sequence[2]=0.27*sequence[2];}
       else if (sequence[2]<=0.01)   {sequence[2]=0.31*sequence[2];}
       else if (sequence[2]<=0.1)    {sequence[2]=0.09*sequence[2];}
       else if (sequence[2]<=0.2)    {sequence[2]=0.05*sequence[2];}
       else if (sequence[2]<=0.3)    {sequence[2]=0.04*sequence[2];}
       else if (sequence[2]<=0.4)    {sequence[2]=0.03*sequence[2];}
       else if (sequence[2]<=0.5)    {sequence[2]=0.02*sequence[2];}
       else if (sequence[2]<=0.6)    {sequence[2]=0.02*sequence[2];}
       else if (sequence[2]<=0.7)    {sequence[2]=0.01*sequence[2];}
       else if (sequence[2]<=0.8)    {sequence[2]=0.02*sequence[2];}
       else if (sequence[2]<=0.9)    {sequence[2]=0.01*sequence[2];}
       else if (sequence[2]<=1)      {sequence[2]=0.16*sequence[2];}
        else                     {sequence[2]=0*sequence[2];}

       if (sequence[1]<=0.0001)      {sequence[1]=0.12*sequence[1];}
       else if (sequence[1]<=0.001)  {sequence[1]=0.27*sequence[1];}
       else if (sequence[1]<=0.01)   {sequence[1]=0.31*sequence[1];}
```

```
        else if (sequence[1]<=0.1)     {sequence[1]=0.09*sequence[1];}
        else if (sequence[1]<=0.2)     {sequence[1]=0.05*sequence[1];}
        else if (sequence[1]<=0.3)     {sequence[1]=0.04*sequence[1];}
        else if (sequence[1]<=0.4)     {sequence[1]=0.03*sequence[1];}
        else if (sequence[1]<=0.5)     {sequence[1]=0.02*sequence[1];}
        else if (sequence[1]<=0.6)     {sequence[1]=0.02*sequence[1];}
        else if (sequence[1]<=0.7)     {sequence[1]=0.01*sequence[1];}
        else if (sequence[1]<=0.8)     {sequence[1]=0.02*sequence[1];}
        else if (sequence[1]<=0.9)     {sequence[1]=0.01*sequence[1];}
        else if (sequence[1]<=1)       {sequence[1]=0.16*sequence[1];}
             else                      {sequence[1]=0*sequence[1];}

        if (sequence[0]<=0.0001)       {sequence[0]=0.12*sequence[0];}
        else if (sequence[0]<=0.001)   {sequence[0]=0.27*sequence[0];}
        else if (sequence[0]<=0.01)    {sequence[0]=0.31*sequence[0];}
        else if (sequence[0]<=0.1)     {sequence[0]=0.09*sequence[0];}
        else if (sequence[0]<=0.2)     {sequence[0]=0.05*sequence[0];}
        else if (sequence[0]<=0.3)     {sequence[0]=0.04*sequence[0];}
        else if (sequence[0]<=0.4)     {sequence[0]=0.03*sequence[0];}
        else if (sequence[0]<=0.5)     {sequence[0]=0.02*sequence[0];}
        else if (sequence[0]<=0.6)     {sequence[0]=0.02*sequence[0];}
        else if (sequence[0]<=0.7)     {sequence[0]=0.01*sequence[0];}
        else if (sequence[0]<=0.8)     {sequence[0]=0.02*sequence[0];}
        else if (sequence[0]<=0.9)     {sequence[0]=0.01*sequence[0];}
        else if (sequence[0]<=1)       {sequence[0]=0.16*sequence[0];}
         else                    {sequence[0]=0*sequence[0];}

        //for (i=0; i<5; i++) {gotoxy(0,87+i); cout<<sequence2[i]<<endl;}

        for (i=0; i<24; i++) {standard fa[i]=(standard fa[i]*sequence2[0]);}
        for (i=0; i<24; i++) {pf 10[i]=(pf 10[i]*sequence2[1]);}
        for (i=0; i<24; i++) {pf 110[i]=(pf 110[i]*sequence2[2]);}
        for (i=0; i<24; i++) {pf_101[i]=(pf_101[i]*sequence2[3]);}

        for (i=0; i<24; i++) {pf_1101[i]=(pf_1101[i]*sequence2[4]);}

//-------------------------------------------------------------------------------------------//

        for (i=0; i<24; i++) {standard fa[i] = standard fa[i] - off standard fa [i] +
                                               pf 10[i] - off pf 10 [i] + pf_110[i] -
                                               off pf 110 [i] + pf 101[i] -
                                               off pf 101 [i] + pf_1101[i] -
                                               off_pf_1101[i];}

         for (i=0; i<24; i++) {
                   if (standard fa[i]<0) {standard_fa[i]=standard_fa[i]
                                *(-1);}
                   gotoxy(0,67+i); cout<<"\n"<<standard_fa[i]<<endl;
                  }

    for (i=0; i<24; i++) {
                   if (standard fa[i]<0) {minimum=0;}
                   else                  {minimum=standard_fa[i];}

                              pulse energy = minimum * photon_energy;
                         energy in frame = pulse energy *
                              mark space correction;
                              energy per pcm bit = (energy in frame/pcm bits);
                              dBm = 10 * (log((energy_per_pcm_bit*B)/pow(10,-3))
                              /log(10));
                   gotoxy(0,95+i); cout<<"\n"<<dBm<<endl;
                 }*/
                  }

              delete [] sequence; sequence=0;
              delete [] sequence2; sequence2=0;
             }

//-------------------------------------------------------------------------------------------//

         gotoxy(19,50);cout<<"DISPLAY INPUT ARRAYS? PRESS1,3 ELSE 8   "<<endl;
         gotoxy(64,47);cin>>press3;

      while ( (press3!=1)||(press3!=3)||(press3!=8)||(press3!=-1) )
            {
         if (press3==1) {
             clrscr2();
         gotoxy(0,56); cout<<"FALSE ALARM MPPM-DATA ARRAY            "<<endl;
      print_input_arrays(mppm_data,pow(2,pcm_bits),
```

```
        x+pcm bits,x,y,aut pasc,aut pasc2,aut pasc3,
        aut pcm bits,pcm bits,pasc,pasc2,pasc3,start number,
        end number,press1,press2,press3,press4,press5);

     press6=0;

     do
         {
    gotoxy(19,50);
        cout<<"DO YOU WANT TO SAVE THE DATA IN A FILE?                "<<endl;
        gotoxy(33,47);cin>>press6;
    if (press6==-1) {break;}
    if (press6==2)  {break;}
    if (press6==3)  {break;}
                           }while(press6!=1);

                           if (press6==3) {file_op1.close(); break;}
                           if (press6==1) {
                                  k=1;
                            for(i=0; i<(x+pcm_bits)*pow(2,pcm_bits); i++)
                {
                 if (i==k*(x+pcm bits)) {file op1<<"\n"; k++;}
                 file_op1<<mppm_data[i]<<"  ";
                }
                file op1.close();
                                      }


                           break;
                          }
//-------------------------------------------------------------------------------------//

            else if (press3==3) {
                               clrscr2();
                    gotoxy(0,56);
                    cout<<"WEIGTHED FALSE ALARM ARRAY              "<<endl;
            print input arrays(weighted fa mppm data,pasc3,
                    x+pcm bits,x,y,aut pasc,aut pasc2,aut pasc3,aut pcm bits,
                    pcm bits,pasc,pasc2,pasc3,start_number,end_number,press1,
                    press2,press3,press4,press5);

            press6=0;
                    do
            {
            gotoxy(19,50);
                    cout<<"DO YOU WANT TO SAVE THE DATA IN A FILE?                "
                    <<endl; gotoxy(33,47);cin>>press6;
            if (press6==-1) {break;}
            if (press6==2)  {break;}
            if (press6==3)  {break;}
                    }while(press6!=1);
                    if (press6==3) {file_op1.close(); break;}
            if (press6==1) {
            k=1;
            for (i=0; i<(x+pcm_bits)*pasc3; i++)
            {
            if (i==k*(x+pcm bits)) {file op1<<"\n"; k++;}
            file_op1<<weighted_fa_mppm_data[i]<<"  ";
            }
            file op1.close();
               }


              break;
             }
            else if ( (press3==8)||(press3==-1) ) {break;}
            else
                   {gotoxy(19,50);
                   cout<<"WRONG SELECTION.TRY AGAIN                      "<<endl;
                   gotoxy(64,47);cin>>press3;}
           }

           if (press3==-1) {break;}
               if (press6==-1) {break;}

               clrscr2();

           delete [] mppm data; mppm data=0;
                   delete [] weighted_fa_mppm_data; weighted_fa_mppm_data=0;
//-------------------------------------------------------------------------------------//
```

```
          delete [] mppm; mppm=0;
          delete [] data; data=0; //66*T


        }//end of if-else mppm_fa null statement

  }//end of if-press1(2) and (y=1)

  else if (press1==3)
  {
    mppm = new int [x*pasc2]; data = new int [pcm bits*pasc2];
    weighted_er_mppm_data = new int [(x+pcm_bits)*pasc2];

    if (mppm==NULL) {gotoxy(19,50);
        cout<<"NOT ENOUGHT MEMORY FOR MPPM2 ARRAY!!!!    "<<endl;}
    else { gotoxy(56,11); cout<<erasure weight<<endl;
           gotoxy(60,12); cout<<false_alarm_weight<<endl;

           for (i=0; i<x*pasc2; i++) mppm[i] = 0;
           for (i=0; i<y-erasure_bits; i++) mppm[i] = 1;

           i=0; j=0; l=0; m=0; extra_pulses=0; k=1;

           for (m=0; m<2*pasc2; m++)
           { for (i=0; i<x-1; i++) if ( mppm[i+(k-1)*x]==1 ) j=i;

             for (i=1; i<=y-erasure bits; i++)
             { if (mppm[k*x-i]==0) break;
               else if (mppm[k*x-i]==1) extra pulses++;
             } if (extra_pulses==y-erasure_bits) break;

             if ( j!=(x-1) )
             { for (i=0; i<j; i++) mppm[i+k*x]=mppm[i+(k-1)*x];
               mppm[j+k*x+1]=1;

               for (i=1; i<=extra pulses; i++)
               mppm[j+k*x+1+i]=1;

               l=0; extra pulses=0;
             }
             else if ( (j==(x-1) ) && (extra pulses!=0) )
                     {l=extra_pulses; k--; extra_pulses=0;}
             k++;
           } //22*T*(x*pasc2)*(y-erasure bits)
         }//end of if-else mppm null statement

    erasure_weight2=0;

//-------------------------------------------------------------------------------------------//

    if (data==NULL) {gotoxy(19,50);
    cout<<"NOT ENOUGHT MEMORY FOR DATA2 ARRAY!!!!   "<<endl;}
    else { for (i=0; i<pcm bits*pasc2; i++) data[i] = 0;
           w=0; row counter=0; bit counter=0; k=1;
           for (l=1; l<=pasc2; l++)
           { for (j=x; j<(x+pcm bits); j++)
             { for (m=0; m<pow(2,pcm bits); m++)
               { for (i=0; i<x; i++) {
                 if ( mppm[i+(l-1)*x]==mppm data[i+(k-1)*(pcm bits+x)] )
                    row counter++;}
                    if ( ( row counter==(x-erasure bits) ) && (
                          mppm data[j+(k-1)*(pcm bits+x)]==1 ) ) bit_counter++;
                 if (row counter==x-erasure_bits) erasure_weight2++;
                    row_counter=0; k++;
               } k=1;
                 if ( bit counter>(erasure weight2/2.0) )
                 data[w+(l-1)*pcm bits]=1;
                 else if ( bit counter<(erasure_weight2/2.0) )
                 data[w+(l-1)*pcm bits]=0;
                 else if ( bit counter=(erasure_weight2/2.0) )
                 data[w+(l-1)*pcm bits]=2;
                 bit counter=0; w++; erasure_weight2=0;
             } k=1; w=0;
           }

         }//end of if-else data null statement

    if (weighted er mppm data==NULL) {gotoxy(19,50);
    cout<<"NOT ENOUGHT MEMORY FOR WEIGHT-ERASURE    "<<endl;}
    else { for (i=0; i<(x+pcm_bits)*pasc2; i++) weighted_er_mppm_data[i] = 0;

           k=0;
```

```
          for (m=0; m<pasc2; m++)
          {
            for (i=0; i<x; i++)
            { weighted_er_mppm_data[k*(x+pcm_bits)+i] = mppm[k*x+i]; }
            k++; l++;
          }

          k=0; l=0;
          for (m=0; m<pasc2; m++)
          {
            for (i=x; i<x+pcm bits; i++)
            { weighted er mppm data[k*(x+pcm bits)+i]
                        = data[(l*pcm_bits)-x+i]; }
            k++; l++;
          }

        }//end of if-else weighted_er_mppm_data null statement

//-------------------------------------------------------------------------------------//

    mppm = new int [x*pasc3]; data = new int [pcm bits*pasc3];
    weighted_fa_mppm_data = new int [(x+pcm_bits)*pasc3];

    if (mppm==NULL) {gotoxy(19,50);
    cout<<"NOT ENOUGHT MEMORY FOR MPPM3 ARRAY!!!!        "<<endl;}
    else { for (i=0; i<x*pasc3; i++) mppm[i] = 0;
           for (i=0; i<y+false_alarm_bits; i++) mppm[i] = 1;

           i=0; j=0; l=0; m=0; extra pulses=0; k=1;
           for (m=0; m<2*pasc3; m++)
           { for (i=0; i<x-1; i++) if ( mppm[i+(k-1)*x]==1 ) j=i;

             for (i=1; i<=y+false alarm bits; i++)
             { if (mppm[k*x-i]==0) break;
               else if (mppm[k*x-i]==1) extra pulses++;
             } if (extra_pulses==y+false_alarm_bits) break;

             if ( j!=(x-1) )
             { for (i=0; i<j; i++) mppm[i+k*x]=mppm[i+(k-1)*x];
                mppm[j+k*x+1]=1;

               for (i=1; i<=extra pulses; i++)
                mppm[j+k*x+1+i]=1;

               l=0; extra pulses=0;
             }
             else if ( (j==(x-1) ) && (extra pulses!=0) )
                     {l=extra_pulses; k--; extra_pulses=0;}

             k++;
           }

        }//end of if-else mppm null statement

//-------------------------------------------------------------------------------------//

    false alarm weight2=0;
    if (data==NULL) {gotoxy(19,50);
    cout<<"NOT ENOUGHT MEMORY FOR DATA3 ARRAY!!!!      "<<endl;}
    else { for (i=0; i<pcm bits*pasc3; i++) data[i] = 0;
           w=0; row counter=0; bit counter=0; k=1;
           for (l=1; l<=pasc3; l++)
           { for (j=x; j<(x+pcm bits); j++)
             { for (m=0; m<pow(2,pcm bits); m++)
               { for (i=0; i<x; i++)
                 { if ( mppm[i+(l-1)*x]==mppm data[i+(k-1)*(pcm bits+x)] )
                   row counter++; }
                 if ( ( row counter==(x-false alarm bits) ) &&
                 ( mppm data[j+(k-1)*(pcm bits+x)]==1 ) ) bit counter++;
                 if ( row counter==(x-false_alarm_bits) ) false_alarm_weight2++;
                      row_counter=0; k++;
               } k=1;
               if      ( bit counter > (false_alarm_weight2/2.0) )
               data[w+(l-1)*pcm bits]=1;
               else if ( bit counter < (false_alarm_weight2/2.0) )
               data[w+(l-1)*pcm bits]=0;
               else if ( bit counter = (false_alarm_weight2/2.0) )
               data[w+(l-1)*pcm bits]=2;
               bit counter=0; w++; false_alarm_weight2=0;
             } k=1; w=0;
           }//66*T*pcm bits*pasc3
        }//end of if-else data null statement
```

```
    if (weighted fa mppm data==NULL) {gotoxy(19,50);
    cout<<"NOT ENOUGHT MEMORY FOR WEIGHT-FA ALARM     "<<endl;}
    else { for (i=0; i<(x+pcm_bits)*pasc3; i++) weighted_fa_mppm_data[i] = 0;

           k=0;
           for (m=0; m<pasc3; m++)
           { for (i=0; i<x; i++)
             { weighted_fa_mppm_data[k*(x+pcm_bits)+i] = mppm[k*x+i]; }
             k++; l++;
           }

           k=0; l=0;
           for (m=0; m<pasc3; m++)
           { for (i=x; i<x+pcm bits; i++)
             { weighted fa mppm data[k*(x+pcm bits)+i] =
                       data[(l*pcm_bits)-x+i]; }
             k++; l++;
           }//15*T*(pasc3*x)
         }//end of if-else weighted_fa_mppm_data null statement

    mppm_ws = new int [(x+pcm_bits+2*y)*pow(2,pcm_bits)];

//-------------------------------------------------------------------------------------//

    if (mppm ws == NULL) {gotoxy(19,50);
    cout<<"NOT ENOUGHT MEMORY FOR MPPM-WS ARRAY            "<<endl;}
    else { for (i=0; i<(x+pcm_bits+2*y)*pow(2,pcm_bits); i++) mppm_ws[i] = 0;

           flag_ws=1;

           k=0;
           for (m=0; m<pow(2,pcm bits); m++)
           { for (i=0; i<x; i++) mppm ws[k*(x+pcm bits+2*y)+i]=
             mppm data[k*(x+pcm_bits)+i];
                  k++;
           }

           a=0;
           if (y%2==0) a=y/2;
           else        a=(y/2)+1;

           k=0; l=0; n=0; row counter=0; n1=0; counter=0; flag=0; o=0; w=0;
           for (m=0; m<pow(2,pcm bits); m++)
           { for (p=0; p<a; p++)
             { for (j=flag; j<x; j++)
               { if ( (mppm data[k*(x+pcm bits)+j]==1) &&
                           (counter!=0) ) {n1=j; flag=n1+1; break;}
                 else if ( (mppm data[k*(x+pcm bits)+j]==1)
                         && (counter==0) ) {n=j; counter++;}
               }

               if (p==0) {o=0;}

               if ( (n!=0) && (n!=flag) ) {
               mppm data[k*(x+pcm bits)+n-1]=1; for (z=0; z<pasc3; z++)
           { for (j=0; j<x; j++) if (mppm data[k*(x+pcm bits)+j]==
                weighted fa mppm data[l*(x+pcm bits)+j]) row_counter++;
       if (row counter==x) { for (j=x; j<x+pcm bits; j++)
               { mppm ws[j+k*(x+pcm bits)+w*2*y]=
                 weighted fa mppm data[l*(x+pcm bits)+j]^
                 mppm_data[k*(x+pcm_bits)+j];

               if (mppm ws[j+k*(x+pcm bits)+w*2*y]!=0)
               {mppm ws[(k+1)*(x+pcm bits)+w*2*y+o]++;}
       mppm ws[j+k*(x+pcm bits)+w*2*y]=0;
       } row counter=0; l=0; mppm_data[k*(x+pcm_bits)+n-1]=0; break;
                 }
          else l++; row_counter=0;
         }
       } l=0; o+=2;

           if ( ( n1!=(n+1) ) && (n1!=0) ) { mppm_data[k*(x+pcm_bits)+n1-1]=1;
           for (z=0; z<pasc3; z++)
       { for (j=0; j<x; j++) if (mppm data[k*(x+pcm bits)+j]==
            weighted fa mppm data[l*(x+pcm bits)+j]) row counter++;
         if (row counter==x) { for (j=x; j<x+pcm_bits; j++)
             { mppm ws[j+k*(x+pcm bits)+w*2*y]=
               weighted fa mppm data[l*(x+pcm bits)+j]^
               mppm data[k*(x+pcm_bits)+j];

             if (mppm_ws[j+k*(x+pcm_bits)+w*2*y]!=0) {mppm_ws[(k+1)*
```

```
            (x+pcm bits)+w*2*y+o]++;}
          mppm ws[j+k*(x+pcm bits)+w*2*y]=0;
        } row_counter=0; l=0; mppm_data[k*(x+pcm_bits)+n1-1]=0; break;
      }
      else l++; row_counter=0;
     }
        } n=0; n1=0; counter=0; row_counter=0; l=0; o+=2;

  } k++; w++; flag=0; n=0; n1=0; row_counter=0; l=0; counter=0;
  }

//-------------------------------------------------------------------------------------//

        k=0; l=0; n=0; row_counter=0; n1=0; counter=0; flag=0; bit_counter=0;
        l1=1; o=1; w=0;
        for (m=0; m<pow(2,pcm_bits); m++)
        { for (p=0; p<a; p++)
          { for (j=flag; j<x; j++)
            { if ( (mppm data[k*(x+pcm bits)+j]==1)&&(counter!=0) )
              {n1=j; flag=n1+1; break;}
              else if ( (mppm data[k*(x+pcm_bits)+j]==1)&&(counter==0) )
                  {n=j; counter++;}
            }

            if (p==0) {o=1;}

            if (n!=-1) { if (n!=(x-1)) {mppm_data[k*(x+pcm_bits)+n+1]=1;
            mppm data[k*(x+pcm bits)+n]=0;}
                        else           mppm_data[k*(x+pcm_bits)+n]=0;

            for (j=0; j<x; j++)  if (mppm_data[k*(x+pcm_bits)+j]==1)
            bit_counter++;

         if (bit counter==y) { for (z=0; z<pow(2,pcm bits); z++)
      { for (j=0; j<x; j++) if ( mppm data[k*(x+pcm bits)+j]==
              mppm data[l1*(x+pcm bits)+j]) row counter++;
      if (row counter==x) { for (j=x; j<x+pcm bits; j++)
      { mppm ws[j+k*(x+pcm bits)+w*2*y]=mppm data[l1*(x+pcm bits)+j]^
              mppm data[k*(x+pcm_bits)+j];

             if (mppm ws[j+k*(x+pcm bits)+w*2*y]!=0)
             {mppm ws[(k+1)*(x+pcm bits)+w*2*y+o]++;}
      mppm ws[j+k*(x+pcm bits)+w*2*y]=0;
      } row counter=0; l1=k+1; bit counter=0; if (n!=(x-1))
             {mppm data[k*(x+pcm bits)+n+1]=0; mppm_data[k*(x+pcm_bits)+n]=1;}
      else {mppm_data[k*(x+pcm_bits)+n]=1;} break;
      }
      else l1++; row_counter=0;
      }
            }
        else if ( bit counter==(y-1) ) { for (z=0; z<pasc2; z++)
      { for (j=0; j<x; j++) if ( mppm data[k*(x+pcm bits)+j]==
             weighted er mppm data[l*(x+pcm bits)+j]) row_counter++;
      if (row counter==x) { for (j=x; j<x+pcm_bits; j++)
      { mppm ws[j+k*(x+pcm bits)+w*2*y]=
              weighted er mppm data[l*(x+pcm bits)+j]^mppm data[k*(x+pcm bits)+j];

             if (mppm ws[j+k*(x+pcm bits)+w*2*y]!=0)
             {mppm ws[(k+1)*(x+pcm bits)+w*2*y+o]++;}
      mppm ws[j+k*(x+pcm bits)+w*2*y]=0;
      } row counter=0; l=0; mppm data[k*(x+pcm_bits)+n]=1;
             bit_counter=0; break;
      }
      else l++; row_counter=0;
      }
                                                          }
                        } o+=2;

//-------------------------------------------------------------------------------------//

  if (n1!=0) { if ( n1!=(x-1) ) {mppm data[k*(x+pcm_bits)+n1+1]=1;
             mppm data[k*(x+pcm bits)+n1]=0;}
             else mppm_data[k*(x+pcm_bits)+n1]=0;

          for (j=0; j<x; j++) if (mppm data[k*(x+pcm bits)+j]==1) bit counter++;
             if (bit counter==y) { for (z=0; z<pow(2,pcm bits); z++)
             { for (j=0; j<x; j++) if ( mppm data[k*(x+pcm bits)+j]==
               mppm data[l1*(x+pcm bits)+j]) row counter++;
             if (row counter==x) { for (j=x; j<x+pcm bits; j++)
             { mppm ws[j+k*(x+pcm bits)+w*2*y]=mppm data[l1*(x+pcm bits)+j]^
               mppm data[k*(x+pcm_bits)+j];
```

```
                if (mppm ws[j+k*(x+pcm bits)+w*2*y]!=0)
                {mppm ws[(k+1)*(x+pcm bits)+w*2*y+o]++;}
                mppm ws[j+k*(x+pcm bits)+w*2*y]=0;
                } row counter=0; l1=k+1; bit counter=0; if (n1!=(x-1))
               {mppm data[k*(x+pcm bits)+n1+1]=0; mppm data[k*(x+pcm_bits)+n1]=1;}
               else {mppm_data[k*(x+pcm_bits)+n1]=1;} break;
               }
                else l1++; row_counter=0;
               }
                }
               else if ( bit counter==(y-1) ) { for (z=0; z<pasc2; z++)
                    { for (j=0; j<x; j++) if ( mppm data[k*(x+pcm bits)+j]==
                      weighted er mppm data[l*(x+pcm bits)+j]) row counter++;
                    if (row counter==x) { for (j=x; j<x+pcm_bits; j++)
                    { mppm ws[j+k*(x+pcm bits)+w*2*y]=
                      weighted er mppm data[l*(x+pcm bits)+j]
                      ^mppm data[k*(x+pcm bits)+j];

                    if (mppm ws[j+k*(x+pcm bits)+w*2*y]!=0)
                    {mppm ws[(k+1)*(x+pcm bits)+w*2*y+o]++;}
                    mppm ws[j+k*(x+pcm bits)+w*2*y]=0;
                    } row counter=0; l=0; mppm_data[k*(x+pcm_bits)+n1]=1;
                      bit_counter=0; break;
                   }
                    else l++; row_counter=0;
                      }
                    }
                  } n=0; n1=0; counter=0; row_counter=0; l=0; bit_counter=0;
                    l1=k+1; o+=2;
                 } k++; w++; counter=0; n=0; n1=0; flag=0; o+=2; row_counter=0;
                   l=0; bit counter=0; l1=k+1;
                }  //176*T*pow(2,pcm_bits)*x

//-------------------------------------------------------------------------------------//

          press5=1;

          press4=0;
          gotoxy(19,50);
          cout<<"DO YOU WANT TO DISPLAY THE SEQUENCE RESULTS?      "<<endl;
          gotoxy(10,44);cin>>press4;

       while ( (press4!=1)||(press4!=2)||(press4!=-1) )
       { if (press4==-1)                        {break;}
         else if ( (press4==1)||(press4==2) ) {break;}
          else                                {gotoxy(19,50);
             cout<<"WRONG SELECTION.TRY AGAIN                "<<endl;
             gotoxy(10,44);cout<<"           "<<endl;
             gotoxy(10,44);cin>>press4;}
          }

          if (press4==-1) {break;}

          l1=0; zero counter=0;
          for ( m=0; m<pow(2,pcm_bits); m++)
          {
           for ( i=0; i<2; i++) {
                            if (mppm ws[i+m*(x+pcm bits+2*y)]==0)
                                        {zero_counter++;}
                            }
           if (zero counter==2) {break;}
           else                 {zero_counter=0; l1++;}
          }

//-------------------------------------------------------------------------------------//


          bit counter=0; row counter=0; i=0;
          for ( m=0; m<pow(2,pcm bits); m++)
          {  if (mppm ws[i+m*(x+pcm_bits+2*y)]==1) {bit_counter++;}
             //1-bit_counter=11//
             i=x-1;
             if (mppm ws[i+m*(x+pcm_bits+2*y)]==1) {row_counter++;}
             //1-row_counter=9//
             i=0;
          }

          sequence = new float [4];
          if (sequence==NULL) {gotoxy(19,50);
          cout<<"NOT ENOUGHT MEMORY FOR WS SEQUENCE ARRAY!!!!      "<<endl;}
          else                {for (i=0; i<4; i++) sequence[i] = 0;}
```

```
          zero counter=0; pulse counter=0; pulse_counter2=0; l=0;
          j=0; flag=1; limit=0;

          for (i=0; i<pow(2,pcm bits); i++)
          {if (mppm ws[(i*(x+y+1))+(x-1)]==1) {pulse_counter++;}}
          for (i=0; i<pow(2,pcm bits); i++)
          {if (mppm_ws[i*(x+y+1)]==1)         {pulse_counter2++;}}

//-------------------------------------------------------------------------------------//

          for (i=0; i<pow(2,pcm bits); i++) {
                          for (l=0; l<x; l++) {
                     if (mppm ws[(i*(x+2))+l]==1) {
                     if (l==0)            {

                                          sequence[0]=sequence[0]+
          (((pow(2,pcm bits)-pulse counter2)/
          pow(2,pcm bits))*mppm ws[i*(x+y+1)+x]); //Ps
    }

    else if (l==1)       {
    sequence[2]=sequence[2]+((pulse counter2/pow(2,pcm bits))
        *mppm ws[i*(x+y+1)+x]);                    //Ps 11
        sequence[0]=sequence[0]+(((pow(2,pcm_bits)-pulse_counter2)/pow(2,pcm_bits))
        *mppm_ws[i*(x+y+1)+x]); //Ps1_1
        }
    else if ( l==(x-1) ) {
    sequence[3]=sequence[3]+((pulse counter/pow(2,pcm bits))*
         mppm ws[i*(x+y+1)+x]);                    //Ps1 1
    sequence[0]=sequence[0]+(((pow(2,pcm_bits)-pulse_counter)/pow(2,pcm_bits))*
        mppm ws[i*(x+y+1)+x]);  //Ps
    }

    else            {sequence[0]=sequence[0]+mppm_ws[i*(x+y+1)+x];} //Ps
     }
      }
       }

          for (i=0; i<pow(2,pcm_bits); i++) {
       for (l=0; l<x; l++) {
       if (mppm ws[(i*(x+2))+l]==1) {
       if (l==0)            {
       sequence[3]=sequence[3]+((pulse counter2/pow(2,pcm bits))*
           mppm ws[(i*(x+y+1))+(x+1)]);                  //Ps1 1
           sequence[0]=sequence[0]+(((pow(2,pcm bits)-pulse counter2)
           /pow(2,pcm bits))*mppm ws[(i*(x+y+1))+(x+1)]); //Ps
          }


    else if ( l==(x-1) ) {
    sequence[3]=sequence[3]+((pulse counter/pow(2,pcm_bits))*
        mppm ws[(i*(x+y+1))+(x+1)]);//Ps1 1

                        sequence[0]=sequence[0]+(((pow(2,pcm_bits)-pulse_counter)/pow(2,pcm_b
its))*mppm ws[(i*(x+y+1))+(x+1)]);  //Ps
//-------------------------------------------------------------------------------------//

                 }


        else            {sequence[0]=sequence[0]+mppm ws[(i*(x+y+2))+(x+1)];} //Ps
                                                                                    }
    }                                          }
        pulse_counter=0; pulse_counter2=0;

        for (i=0; i<y*(y+2)+1; i++) {sequence[i]=(sequence[i]/
        (pcm_bits*pow(2,pcm_bits)));} //156*x*pow(2,pcm_bits)*T

        if ( (press4==1)&&(press5==2) ) {
                    //gotoxy(0,57); cout<<"Ps:"<<endl;
                                    //gotoxy(3,57);cout<<sequence[0]<<endl;

                    //for (i=0; i<y; i++) { gotoxy(0,58+y+i);
                                    //cout<<"Ps1.1:"<<endl; gotoxy(6,58+y+i);
                                    //cout<<sequence[(i+1)*(y+1)]<<endl;}



                    k=1; n=0; w=0; z=0;
                    for (i=0; i<y*(y+2)+1; i++) {
```

```
                if (w>60) {n++; w=0;}
                gotoxy(0+w,77+n); cout<<sequence[i]<<endl;

                w=w+12;
                if (i==k*y+z) {k++; n++; w=0; z++;}

                }
                                    }
          else if (press5==1)
          {
           if (press4==1) {

               gotoxy(0,57); cout<<"Ps:"<<endl;
                           gotoxy(3,57);cout<<sequence[0]<<endl;

               gotoxy(0,58); cout<<"Ps.11:"<<endl;
                           gotoxy(6,59);cout<<sequence[2]<<endl;

               gotoxy(0,59); cout<<"Ps1.1:"<<endl;
                           gotoxy(6,60);cout<<sequence[3]<<endl;
                   gotoxy(0,60); cout<<"Ps.101:"<<endl;
                           gotoxy(7,58);cout<<sequence[1]<<endl;


//-------------------------------------------------------------------------------------//

  /*if (sequence[8]<=0.0001)       {sequence[8]=0.12*sequence[8];}
  else if (sequence[8]<=0.001)   {sequence[8]=0.27*sequence[8];}
  else if (sequence[8]<=0.01)    {sequence[8]=0.31*sequence[8];}
  else if (sequence[8]<=0.1)     {sequence[8]=0.09*sequence[8];}
  else if (sequence[8]<=0.2)     {sequence[8]=0.05*sequence[8];}
  else if (sequence[8]<=0.3)     {sequence[8]=0.04*sequence[8];}
  else if (sequence[8]<=0.4)     {sequence[8]=0.03*sequence[8];}
  else if (sequence[8]<=0.5)     {sequence[8]=0.02*sequence[8];}
  else if (sequence[8]<=0.6)     {sequence[8]=0.02*sequence[8];}
  else if (sequence[8]<=0.7)     {sequence[8]=0.01*sequence[8];}
  else if (sequence[8]<=0.8)     {sequence[8]=0.02*sequence[8];}
  else if (sequence[8]<=0.9)     {sequence[8]=0.01*sequence[8];}
  else if (sequence[8]<=1)       {sequence[8]=0.16*sequence[8];}
       else                      {sequence[8]=0*sequence[8];}
  if (sequence[7]<=0.0001)       {sequence[7]=0.12*sequence[7];}
  else if (sequence[7]<=0.001)   {sequence[7]=0.27*sequence[7];}
  else if (sequence[7]<=0.01)    {sequence[7]=0.31*sequence[7];}
  else if (sequence[7]<=0.1)     {sequence[7]=0.09*sequence[7];}
  else if (sequence[7]<=0.2)     {sequence[7]=0.05*sequence[7];}
  else if (sequence[7]<=0.3)     {sequence[7]=0.04*sequence[7];}
  else if (sequence[7]<=0.4)     {sequence[7]=0.03*sequence[7];}
  else if (sequence[7]<=0.5)     {sequence[7]=0.02*sequence[7];}
  else if (sequence[7]<=0.6)     {sequence[7]=0.02*sequence[7];}
  else if (sequence[7]<=0.7)     {sequence[7]=0.01*sequence[7];}
  else if (sequence[7]<=0.8)     {sequence[7]=0.02*sequence[7];}
  else if (sequence[7]<=0.9)     {sequence[7]=0.01*sequence[7];}
  else if (sequence[7]<=1)       {sequence[7]=0.16*sequence[7];}
       else                      {sequence[7]=0*sequence[7];}

  if (sequence[6]<=0.0001)       {sequence[6]=0.12*sequence[6];}
  else if (sequence[6]<=0.001)   {sequence[6]=0.27*sequence[6];}
  else if (sequence[6]<=0.01)    {sequence[6]=0.31*sequence[6];}
  else if (sequence[6]<=0.1)     {sequence[6]=0.09*sequence[6];}
  else if (sequence[6]<=0.2)     {sequence[6]=0.05*sequence[6];}
  else if (sequence[6]<=0.3)     {sequence[6]=0.04*sequence[6];}
  else if (sequence[6]<=0.4)     {sequence[6]=0.03*sequence[6];}
  else if (sequence[6]<=0.5)     {sequence[6]=0.02*sequence[6];}
  else if (sequence[6]<=0.6)     {sequence[6]=0.02*sequence[6];}
  else if (sequence[6]<=0.7)     {sequence[6]=0.01*sequence[6];}
  else if (sequence[6]<=0.8)     {sequence[6]=0.02*sequence[6];}
  else if (sequence[6]<=0.9)     {sequence[6]=0.01*sequence[6];}
  else if (sequence[6]<=1)       {sequence[6]=0.16*sequence[6];}
       else                      {sequence[6]=0*sequence[6];}

  if (sequence[5]<=0.0001)       {sequence[5]=0.12*sequence[5];}
  else if (sequence[5]<=0.001)   {sequence[5]=0.27*sequence[5];}
  else if (sequence[5]<=0.01)    {sequence[5]=0.31*sequence[5];}
  else if (sequence[5]<=0.1)     {sequence[5]=0.09*sequence[5];}
  else if (sequence[5]<=0.2)     {sequence[5]=0.05*sequence[5];}
  else if (sequence[5]<=0.3)     {sequence[5]=0.04*sequence[5];}
  else if (sequence[5]<=0.4)     {sequence[5]=0.03*sequence[5];}
  else if (sequence[5]<=0.5)     {sequence[5]=0.02*sequence[5];}
  else if (sequence[5]<=0.6)     {sequence[5]=0.02*sequence[5];}
```

```
else if (sequence[5]<=0.7)     {sequence[5]=0.01*sequence[5];}
else if (sequence[5]<=0.8)     {sequence[5]=0.02*sequence[5];}
else if (sequence[5]<=0.9)     {sequence[5]=0.01*sequence[5];}
else if (sequence[5]<=1)       {sequence[5]=0.16*sequence[5];}
     else                      {sequence[5]=0*sequence[5];}

if (sequence[4]<=0.0001)       {sequence[4]=0.12*sequence[4];}
else if (sequence[4]<=0.001)   {sequence[4]=0.27*sequence[4];}
else if (sequence[4]<=0.01)    {sequence[4]=0.31*sequence[4];}
else if (sequence[4]<=0.1)     {sequence[4]=0.09*sequence[4];}
else if (sequence[4]<=0.2)     {sequence[4]=0.05*sequence[4];}
else if (sequence[4]<=0.3)     {sequence[4]=0.04*sequence[4];}
else if (sequence[4]<=0.4)     {sequence[4]=0.03*sequence[4];}
else if (sequence[4]<=0.5)     {sequence[4]=0.02*sequence[4];}
else if (sequence[4]<=0.6)     {sequence[4]=0.02*sequence[4];}
else if (sequence[4]<=0.7)     {sequence[4]=0.01*sequence[4];}
else if (sequence[4]<=0.8)     {sequence[4]=0.02*sequence[4];}
else if (sequence[4]<=0.9)     {sequence[4]=0.01*sequence[4];}
else if (sequence[4]<=1)       {sequence[4]=0.16*sequence[4];}
     else                      {sequence[4]=0*sequence[4];}


if (sequence[3]<=0.0001)       {sequence[3]=0.12*sequence[3];}
else if (sequence[3]<=0.001)   {sequence[3]=0.27*sequence[3];}
else if (sequence[3]<=0.01)    {sequence[3]=0.31*sequence[3];}
else if (sequence[3]<=0.1)     {sequence[3]=0.09*sequence[3];}
else if (sequence[3]<=0.2)     {sequence[3]=0.05*sequence[3];}
else if (sequence[3]<=0.3)     {sequence[3]=0.04*sequence[3];}
else if (sequence[3]<=0.4)     {sequence[3]=0.03*sequence[3];}
else if (sequence[3]<=0.5)     {sequence[3]=0.02*sequence[3];}
else if (sequence[3]<=0.6)     {sequence[3]=0.02*sequence[3];}
else if (sequence[3]<=0.7)     {sequence[3]=0.01*sequence[3];}
else if (sequence[3]<=0.8)     {sequence[3]=0.02*sequence[3];}
else if (sequence[3]<=0.9)     {sequence[3]=0.01*sequence[3];}
else if (sequence[3]<=1)       {sequence[3]=0.16*sequence[3];}
else                      {sequence[3]=0*sequence[3];}

if (sequence[2]<=0.0001)       {sequence[2]=0.12*sequence[2];}
else if (sequence[2]<=0.001)   {sequence[2]=0.27*sequence[2];}
else if (sequence[2]<=0.01)    {sequence[2]=0.31*sequence[2];}
else if (sequence[2]<=0.1)     {sequence[2]=0.09*sequence[2];}
else if (sequence[2]<=0.2)     {sequence[2]=0.05*sequence[2];}
else if (sequence[2]<=0.3)     {sequence[2]=0.04*sequence[2];}
else if (sequence[2]<=0.4)     {sequence[2]=0.03*sequence[2];}
else if (sequence[2]<=0.5)     {sequence[2]=0.02*sequence[2];}
else if (sequence[2]<=0.6)     {sequence[2]=0.02*sequence[2];}
else if (sequence[2]<=0.7)     {sequence[2]=0.01*sequence[2];}
else if (sequence[2]<=0.8)     {sequence[2]=0.02*sequence[2];}
else if (sequence[2]<=0.9)     {sequence[2]=0.01*sequence[2];}
else if (sequence[2]<=1)       {sequence[2]=0.16*sequence[2];}
else                      {sequence[2]=0*sequence[2];}

if (sequence[1]<=0.0001)       {sequence[1]=0.12*sequence[1];}
else if (sequence[1]<=0.001)   {sequence[1]=0.27*sequence[1];}
else if (sequence[1]<=0.01)    {sequence[1]=0.31*sequence[1];}
else if (sequence[1]<=0.1)     {sequence[1]=0.09*sequence[1];}
else if (sequence[1]<=0.2)     {sequence[1]=0.05*sequence[1];}
else if (sequence[1]<=0.3)     {sequence[1]=0.04*sequence[1];}
else if (sequence[1]<=0.4)     {sequence[1]=0.03*sequence[1];}
else if (sequence[1]<=0.5)     {sequence[1]=0.02*sequence[1];}
else if (sequence[1]<=0.6)     {sequence[1]=0.02*sequence[1];}
else if (sequence[1]<=0.7)     {sequence[1]=0.01*sequence[1];}
else if (sequence[1]<=0.8)     {sequence[1]=0.02*sequence[1];}
else if (sequence[1]<=0.9)     {sequence[1]=0.01*sequence[1];}
else if (sequence[1]<=1)       {sequence[1]=0.16*sequence[1];}
     else                      {sequence[1]=0*sequence[1];}

if (sequence[0]<=0.0001)       {sequence[0]=0.12*sequence[0];}
else if (sequence[0]<=0.001)   {sequence[0]=0.27*sequence[0];}
else if (sequence[0]<=0.01)    {sequence[0]=0.31*sequence[0];}
else if (sequence[0]<=0.1)     {sequence[0]=0.09*sequence[0];}
else if (sequence[0]<=0.2)     {sequence[0]=0.05*sequence[0];}
else if (sequence[0]<=0.3)     {sequence[0]=0.04*sequence[0];}
else if (sequence[0]<=0.4)     {sequence[0]=0.03*sequence[0];}
else if (sequence[0]<=0.5)     {sequence[0]=0.02*sequence[0];}
else if (sequence[0]<=0.6)     {sequence[0]=0.02*sequence[0];}
else if (sequence[0]<=0.7)     {sequence[0]=0.01*sequence[0];}
else if (sequence[0]<=0.8)     {sequence[0]=0.02*sequence[0];}
else if (sequence[0]<=0.9)     {sequence[0]=0.01*sequence[0];}
else if (sequence[0]<=1)       {sequence[0]=0.16*sequence[0];}
     else                      {sequence[0]=0*sequence[0];}
```

```
  //for (i=0; i<9; i++) {gotoxy(0,77+i); cout<<sequence2[i]<<endl;}

//-------------------------------------------------------------------------------------------//

   for (i=0; i<24; i++) {standard ws[i]=(standard ws[i]*sequence2[0]);}
   for (i=0; i<24; i++) {ps 11[i]=(ps 11[i]*sequence2[1]);}
   for (i=0; i<24; i++) {ps 111[i]=(ps 111[i]*sequence2[2]);}
   for (i=0; i<24; i++) {ps1 1[i]=(ps1 1[i]*sequence2[3]);}
   for (i=0; i<24; i++) {ps 1 1 1[i]=(ps 1 1 1[i]*sequence2[5]);}
   for (i=0; i<24; i++) {ps 11 1 1[i]=(ps 11 1 1[i]*sequence2[6]);}
   for (i=0; i<24; i++) {ps 1 1 11[i]=(ps 1 1 11[i]*sequence2[7]);}
   for (i=0; i<24; i++) {ps_11_1_11[i]=(ps_11_1_11[i]*sequence2[8]);}

for (i=0; i<24; i++) {standard ws[i] = standard ws[i] - off standard ws [i] +
                      ps 11[i] - off ps 11[i] + ps 111[i] - off ps 111[i] +
                      ps1 1[i] - off ps1 1[i] + ps 1 1 1[i] - off ps 1 1_1[i] +
                      ps 11 1 1[i] - off ps 11 1 1 [i] + ps 1 1 11[i] -
                      off_ps_1_1_11[i] + ps_11_1_11[i] - off_ps_11_1_11[i];}

for (i=0; i<24; i++) {
            if (standard ws[i]<0) {standard ws[i]=standard ws[i]*(-1);}
            gotoxy(0,77+i); cout<<"\n"<<standard ws[i]<<endl;
                                                  }

                          for (i=0; i<24; i++) {
                        if (standard ws[i]<0) {minimum=0;}
                        else
                                                  {minimum=standard_ws[i];}

                                                  pulse energy = minimum *
                                                  photon energy;
                        energy in frame = pulse energy *
                                                  mark space correction;
                                                  energy per pcm bit =
                                                  (energy_in_frame/pcm_bits);

                        dBm = 10*(log((energy per pcm bit*B)
                                                  /pow(10,-3))/log(10));
                        gotoxy(0,105+i);
                                                  cout<<"\n"<<dBm<<endl;
                                                 }

//-------------------------------------------------------------------------------------------//

               for (i=0; i<24; i++) {ps 101[i]=(ps 101[i]*sequence2[0]);}
               for (i=0; i<24; i++) {ps_1101[i]=(ps_1101[i]*sequence2[1]);}

               for (i=0; i<24; i++) {ps 1011[i]=(ps 1011[i]*sequence2[2]);}
               for (i=0; i<24; i++) {ps_10111[i]=(ps_10111[i]*sequence2[3]);}

       for (i=0; i<24; i++) {ps 11011[i]=(ps 11011[i]*sequence2[4]);}
       for (i=0; i<24; i++) {ps_110111[i]=(ps_110111[i]*sequence2[5]);}

                     for (i=0; i<24; i++) {standard_ws[i] = standard_ws[i] -
                           off standard ws[i]
                         + ps 101[i] - off ps_101[i] + ps_1101[i] -
                           off ps 1101[i] +
                           ps 1011[i] - off ps_1011[i] + ps_10111[i] -
                           off ps 10111[i] +
                           ps 11011[i] - off ps_11011[i] + ps_110111[i] -
                           off_ps_110111[i];}

                       for (i=0; i<24; i++) {
                         if (standard ws[i]<0)
                                                  {standard ws[i]=
                                                   standard_ws[i]*(-1);}
                         gotoxy(0,77+i);
                                                  cout<<"\n"<<
                                                  standard_ws[i]<<endl;
                          }

//-------------------------------------------------------------------------------------------//


                          for (i=0; i<24; i++) {
                        if (standard ws[i]<0) {minimum=0;}
                        else     {minimum=standard_ws[i];}

                                                  pulse energy = minimum *
                                                  photon_energy;
```

```
                        energy in frame = pulse energy *
                                          mark space correction;
                                          energy per pcm bit =
                                          (energy_in_frame/pcm_bits);

                        dBm = 10 *
                                         (log((energy per pcm bit*B)/
                                          pow(10,-3))/log(10));
                        gotoxy(0,105+1);
                                          cout<<"\n"<<dBm<<endl;
                           }*/
                  }
          }
          delete [] mppm data; mppm data=0;
          delete [] weighted er mppm data; weighted er mppm data=0;
          delete [] weighted fa mppm data; weighted_fa_mppm_data=0;
          delete [] sequence; sequence=0;
          delete [] mppm; mppm=0;
          delete [] data; data=0;
        }//end of if-else mppm ws null statement
 }//end of if-press1(3) for y==1
}//end of y==1

//-------------------------------------------------------------------------------------//

 else if ( y==(x-1) )
      {
       if (press1==1)
       {
         mppm = new int [x*pasc2]; data = new int [pcm bits*pasc2];
         weighted_er_mppm_data = new int [(x+pcm_bits)*pasc2];

         if (mppm==NULL) {gotoxy(19,50);
         cout<<"NOT ENOUGHT MEMORY FOR MPPM-ER ARRAY!!!!"<<endl;}
         else { gotoxy(56,11); cout<<erasure weight<<endl;
                gotoxy(60,12); cout<<false_alarm_weight<<endl;

                for (i=0; i<x*pasc2; i++) mppm[i] = 0;
                for (i=0; i<y-erasure_bits; i++) mppm[i] = 1;

                i=0; j=0; l=0; m=0; extra_pulses=0; k=1;

                for (m=0; m<2*pasc2; m++)
                { for (i=0; i<x-1; i++)
                if ( mppm[i+(k-1)*x]==1 ) j=i;

                  for (i=1; i<=y-erasure bits; i++)
                  { if (mppm[k*x-i]==0) break;
                    else if (mppm[k*x-i]==1) extra pulses++;
                  } if (extra_pulses==y-erasure_bits) break;

                  if ( j!=(x-1) )
           { for (i=0; i<j; i++) mppm[i+k*x]=mppm[i+(k-1)*x];
                    mppm[j+k*x+1]=1;

                    for (i=1; i<=extra pulses; i++)
                    mppm[j+k*x+1+i]=1;

                    l=0; extra pulses=0;
                  }
                  else if ( (j==(x-1) ) && (extra pulses!=0) )
                  {l=extra_pulses; k--; extra_pulses=0;}
                  k++;
                }
              }//end of if-else mppm null statement

         erasure_weight2=0;//36*T

//-------------------------------------------------------------------------------------//

         if (data==NULL) {gotoxy(19,50);
         cout<<"NOT ENOUGHT MEMORY FOR DATA2 ARRAY!!!!   "<<endl;}
         else { for (i=0; i<pcm bits*pasc2; i++) data[i] = 0; //pcm_bits*pasc2*T
                w=0; row counter=0; bit counter=0; k=1;
                for (l=1; l<=pasc2; l++)
                { for (j=x; j<(x+pcm bits); j++)
                  { for (m=0; m<pow(2,pcm bits); m++)
                    { for (i=0; i<x; i++) { if ( mppm[i+(l-1)*x]==
                      mppm_data[i+(k-1)*(pcm_bits+x)] ) row_counter++; }

                      if ( ( row counter==(x-erasure bits) ) &&
                      ( mppm_data[j+(k-1)*(pcm_bits+x)]==1 ) ) bit_counter++;
```

```
                    if (row counter==x-erasure_bits)
                            erasure weight2++;
                    row counter=0; k++;
                  } k=1;
                  if ( bit counter>(erasure weight2/2.0) )
                  data[w+(l-1)*pcm bits]=1;
                  else if ( bit counter<(erasure_weight2/2.0) )
                  data[w+(l-1)*pcm bits]=0;
                  else if ( bit counter=(erasure_weight2/2.0) )
                  data[w+(l-1)*pcm bits]=2;
                  bit counter=0; w++; erasure_weight2=0;
                } k=1; w=0;
              }//5*T
            }//end of if-else data null statement

        if (weighted er mppm data==NULL) {gotoxy(19,50);
        cout<<"NOT ENOUGHT MEMORY FOR WEIGHT-ERASURE    "<<endl;}
        else { for (i=0; i<(x+pcm_bits)*pasc2; i++) weighted_er_mppm_data[i] = 0;

               k=0;
               for (m=0; m<pasc2; m++)
               {
                for (i=0; i<x; i++)
                { weighted er mppm_data[k*(x+pcm_bits)+i] =
                  mppm[k*x+i];}
                  k++; l++;
                }

                k=0; l=0;
                for (m=0; m<pasc2; m++)
                {
                 for (i=x; i<x+pcm bits; i++) {
                 weighted er mppm data[k*(x+pcm_bits)+i] =
                 data[(l*pcm_bits)-x+i]; }
                 k++; l++;
                }
            }//end of if-else weighted_er_mppm_data null statement

//-------------------------------------------------------------------------------------//

        mppm_er = new int [(x+pcm_bits+y)*pow(2,pcm_bits)];

        if (mppm er==NULL) {gotoxy(19,50);
        cout<<"NOT ENOUGHT MEMORY FOR MPPM-ER ARRAY  "<<endl;}
        else { for (i=0; i<(x+pcm_bits+y)*pow(2,pcm_bits); i++) mppm_er[i] = 0;
               flag_er=1;

               k=0;
               for (m=0; m<pow(2,pcm bits); m++)
               { for (i=0; i<x; i++) mppm er[k*(x+pcm bits+y)+i]
               =mppm data[k*(x+pcm_bits)+i];
                 k++;
               }

               l=0; k=0; j=0; p=0; row counter=0; bit_counter=0; w=y-1;
               //can be also w=0; w=y-1
               for ( m=0; m<pow(2,pcm bits); m++)
               { for (n=0; n<pasc2; n++)
                 { for (i=0; i<x; i++) if ( mppm data[i+l*(x+pcm_bits)]
                   ==weighted er mppm data[i+k*(x+pcm bits)] )
                   row counter++;
                   if ( row counter==(x-erasure bits) )
                   { for ( i=x; i<x+pcm bits; i++)
                     { mppm er[i+l*(x+pcm bits)+y*p]=
                        ( mppm data[i+l*(x+pcm bits)]^
                          weighted_er_mppm_data[i+k*(x+pcm_bits)] );

                    if ( mppm er[i+l*(x+pcm bits)+y*p]!=0 ) bit counter++;
                       else bit counter=bit counter;
                     } mppm er[(j+1)*(x+pcm bits)+y*j+w]=
                       mppm er[(j+1)*(x+pcm bits)+y*j+w]+bit counter;
                   for (i=x; i<x+pcm bits; i++) mppm_er[i+l*(x+pcm_bits)+y*p]=0;
                   w--; //instead of w++
                 } row counter=0; bit counter=0; k++;
               } l++; k=0; p++; j++; bit_counter=0; w=y-1; //instead of w=0 ; w=y-1
             }

//-------------------------------------------------------------------------------------//

     press5=1;

     press4=0;
```

```
        gotoxy(19,50);
        cout<<"DO YOU WANT TO DISPLAY THE SEQUENCES?   "<<endl;
        gotoxy(10,44);cin>>press4;

        while ( (press4!=1)||(press4!=2)||(press4!=-1) )
        { if (press4==-1)                          {break;}
          else if ( (press4==1)||(press4==2) ) {break;}
               else
                 {gotoxy(19,50);
                  cout<<"WRONG SELECTION.TRY AGAIN                "<<endl;
                 gotoxy(10,44);cout<<"  "<<endl; gotoxy(10,44);cin>>press4;}
        }

        if (press4==-1) {break;}

        l1=0; zero counter=0;
        for ( m=0; m<pow(2,pcm_bits); m++)
        {
         for ( i=0; i<2; i++) {
                     if (mppm er[i+m*(x+pcm bits+y)]==0)
                                  {zero_counter++;}
                  }
         if (zero counter==2) {break;}
         else                 {zero_counter=0; l1++;}
        }

        bit counter=0; row counter=0; row counter2=0; i=0;
        for ( m=0; m<pow(2,pcm_bits); m++)
        {
           if (mppm er[i+m*(x+pcm_bits+y)]==1) {bit_counter++;}
                    //1st bit//
           i=x-2;
           if (mppm er[i+m*(x+pcm_bits+y)]==1) {row_counter2++;}
           //Xth-1 bit//
           i=x-1;
           if (mppm er[i+m*(x+pcm_bits+y)]==1) {row_counter++;}
           //Xth bit//
           i=0;
        }

//-------------------------------------------------------------------------------------------//

        sequence = new float [2*y+3];
        if (sequence==NULL)
        {gotoxy(19,50);
        cout<<"NOT ENOUGHT MEMORY FOR ER SEQUENCE ARRAY!!!!      "<<endl;}
        else               {for (i=0; i<2*y+3; i++) sequence[i] = 0;}

        zero counter=0; pulse_counter=0; pulse_counter2=0; l=0; j=0; flag=1;
        limit=0;

        for (w=0; w<(y-1); w++)
        {
         for (n=0; n<pow(2,pcm_bits); n++)
         {
          for (i=0; i<x; i++)
          {
               if ( (mppm er[i+l*(x+pcm bits+y)]==1)&&(i==0)&&(w==0) ) {

         zero counter=0; pulse counter=0; pulse counter2=0;


         sequence[y+1]=sequence[y+1]+(mppm_er[(l+1)*(x+pcm_bits)+l*y+j]*
         (row counter2/pow(2,pcm bits)));
         sequence[l]=sequence[l]+(mppm er[(l+1)*(x+pcm bits)+l*y+j]*
         (row counter/pow(2,pcm bits)));

      sequence[0]=sequence[0]+(mppm er[(l+1)*(x+pcm bits)+l*y+j]*
          ((pow(2,pcm bits)-row counter2-row counter)/pow(2,pcm bits)));

      j++;

      }
          else if ( (mppm er[i+l*(x+pcm bits+y)]==1)&&(i!=0)&(i!=(x-1)) ){
      zero counter=0; pulse counter=0; pulse_counter2=0;

      for (m=i-1; m>=0; m--) {

    if (mppm er[m+l*(x+pcm bits+y)]==1) {pulse counter++;}
    else                                {zero counter=m; break;}
    }
    if ( (pulse_counter==0)&&(i!=1) ) {
```

35

```
pulse counter=0;
for (m=zero counter-1; m>=0; m--) {

if (mppm er[m+l*(x+pcm bits+y)]==1) {pulse counter++;}
else                                {break;}
}
    if ( (pulse counter==0)&&(w==0) )
    {sequence[0]=sequence[0]+mppm er[(l+1)*(x+pcm_bits)+l*y+j]; j++;
     flag=0;}
else if (pulse counter==limit+1)  {
if (flag==0) {sequence[y+pulse counter]=sequence[y+pulse_counter]+
    mppm er[(l+1)*(x+pcm_bits)+l*y+j]; j++;}
else         {
if ( (i==2)&&(pulse counter==1) ) {
sequence[2*y+2]=sequence[2*y+2]+
    (mppm er[(l+1)*(x+pcm bits)+l*y+j]*(row counter2/pow(2,pcm_bits)));
sequence[y+2]=sequence[y+2]+(mppm er[(l+1)*(x+pcm bits)+l*y+j]
    *(row counter/pow(2,pcm bits)));

sequence[y+1]=sequence[y+1]+(mppm er[(l+1)*(x+pcm bits)+l*y+j]*
    ((pow(2,pcm bits)-row counter2-row counter)/pow(2,pcm bits)));

j++; flag=0;
}
else                              {
sequence[y+1+pulse counter]=sequence[y+1+pulse counter]+
    (mppm er[(l+1)*(x+pcm bits)+l*y+j]*(row counter/pow(2,pcm_bits)));
sequence[y+pulse counter]=sequence[y+pulse counter]+
    (mppm er[(l+1)*(x+pcm_bits)+l*y+j]*((pow(2,pcm_bits)-row_counter)
    /pow(2,pcm_bits)));
j++; flag=0;
 }
  }
   }
  }
```

```
//-------------------------------------------------------------------------------------//
```

```
else if ( (pulse counter==0)&&(i==1)&&(w==0) ) {
sequence[y+1]=sequence[y+1]+(mppm er[(l+1)*(x+pcm bits)+l*y+j]*(row counter
    /pow(2,pcm bits)));

sequence[0]=sequence[0]+(mppm er[(l+1)*(x+pcm bits)+l*y+j]*
    ((pow(2,pcm bits)-row counter)/pow(2,pcm bits)));

j++; flag=0;
}


else if (pulse counter==limit+1) {

if (flag==0) {sequence[pulse counter]=sequence[pulse counter]+
    mppm er[(l+1)*(x+pcm bits)+l*y+j]; j++;}

else         {
sequence[pulse counter+1]=sequence[pulse counter+1]+
    (mppm er[(l+1)*(x+pcm bits)+l*y+j]*(row counter/pow(2,pcm_bits)));
sequence[pulse counter]=sequence[pulse counter]+
    (mppm er[(l+1)*(x+pcm_bits)+l*y+j]*((pow(2,pcm_bits)-row_counter)
    /pow(2,pcm_bits)));
j++;
 }

  }
   }
else if ( (mppm er[i+l*(x+pcm bits+y)]==1)&&(i==(x-1)) )       {
zero counter=0; pulse counter=0; pulse_counter2=0;

for (m=i-1; m>=0; m--) {

if (mppm er[m+l*(x+pcm bits+y)]==1) {pulse counter++;}
else                                {zero counter=m; break;}
}
if ( x!=(y+1) ) {
if (pulse counter==0)  {
pulse counter=0;
for (m=zero counter-1; m>=0; m--) {

if (mppm_er[m+l*(x+pcm_bits+y)]==1) {pulse_counter++;}
```

```
else                                {break;}
}
    if (pulse counter==0) {


flag=0;

sequence[2*y+1]=sequence[2*y+1]+(mppm_er[(l+1)*(x+pcm_bits)+l*y+j]
    *(bit counter/pow(2,pcm bits)));
    sequence[0]=sequence[0]+(mppm er[(l+1)*(x+pcm bits)+l*y+j]*
   ((pow(2,pcm_bits)-bit_counter)/pow(2,pcm_bits)));
j++;
 }
else if (pulse counter==limit+1) {



sequence[2*y+1]=sequence[2*y+1]+(mppm_er[(l+1)*(x+pcm_bits)+l*y+j]
    *(bit counter/pow(2,pcm bits)));
sequence[y+1]=sequence[y+1]+(mppm er[(l+1)*(x+pcm bits)+l*y+j]
    *((pow(2,pcm_bits)-bit_counter)/pow(2,pcm_bits)));
j++;
}
}
else if (pulse counter==limit+1) {
sequence[2*y+1]=sequence[2*y+1]+
    (mppm er[(l+1)*(x+pcm bits)+l*y+j]*(bit counter/pow(2,pcm bits)));
sequence[pulse counter]=sequence[pulse_counter]+(mppm_er[(l+1)
    *(x+pcm bits)+l*y+j]
    *((pow(2,pcm_bits)-bit_counter)/pow(2,pcm_bits)));
j++;
 }
  }
else            {
if (pulse counter==0)  {
pulse counter=y;

if (pulse counter==limit+1) {



sequence[2*y]=sequence[2*y]+(mppm er[(l+1)*(x+pcm_bits)+l*y+j]*
                  (row counter/pow(2,pcm bits)));
sequence[2*y-1]=sequence[2*y-1]+(mppm er[(l+1)*(x+pcm bits)+l*y+j]*
                  ((pow(2,pcm bits)-bit counter-row counter)/pow(2,pcm_bits)));
sequence[2*y+1]=sequence[2*y+1]+(mppm_er[(l+1)*(x+pcm_bits)+l*y+j]*
    (bit_counter/pow(2,pcm_bits)));
j++;
 }
  }
else if (pulse counter==limit+1) {


sequence[2*y+1]=sequence[2*y+1]+(mppm er[(l+1)*(x+pcm bits)+l*y+j]
                   *(bit counter/pow(2,pcm bits)));
sequence[pulse counter]=sequence[pulse counter]+
    (mppm er[(l+1)*(x+pcm_bits)+l*y+j]*((pow(2,pcm_bits)-bit_counter)
    /pow(2,pcm_bits)));
j++;
}


}
}
    if (n==pasc)                                                    {break;}
}l++; j=0; if (l<=l1) {flag=1;}
else       {flag=0;}
if (n==pasc)                                                  {break;}
    }l=0; j=0; flag=1; limit++;
   }

   for (i=0; i<2*y+3; i++) {sequence[i]=(sequence[i]/(pcm_bits*pow(2,pcm_bits)));}

//-------------------------------------------------------------------------------//
```

```
        if ( (press4==1)&&(press5==2) ) {
        gotoxy(0,57); cout<<"Pe:"<<endl; gotoxy(3,57);cout<<sequence[0]<<endl;
        for (i=0; i<y; i++) { gotoxy(0,58+i); cout<<"Pe.11:"<<endl;
        gotoxy(6,58+i);cout<<sequence[i+1]<<endl;}
        for (i=0; i<y; i++) { gotoxy(0,58+y+i);
        cout<<"Pe.101:"<<endl; gotoxy(7,58+y+i);cout<<sequence[i+y+1]<<endl;}

        gotoxy(0,58+2*y); cout<<"Pe1.1:"<<endl; gotoxy(6,58+2*y);
        cout<<sequence[2*y+1]<<endl;
        gotoxy(0,58+2*y+1); cout<<"Pe10101:"<<endl;
        gotoxy(8,58+2*y+1);cout<<sequence[2*(y+1)]<<endl;

        k=1; n=0; w=0; z=0;
        for (i=0; i<2*y+3; i++) {
                        gotoxy(0+w,87+n); cout<<sequence[i]<<endl;

                                w=w+13;
                   if (i==k*y+z) {k++; n++; w=0; z++;}
                   }
                   delete [] sequence; sequence=0;
                }
        else if (press5==1)
                {
//-------------------------------------------------------------------------------------------//

                sequence2 = new float [7];
                if (sequence2==NULL) {gotoxy(19,50);
                cout<<"NOT ENOUGHT MEMORY FOR ER SEQUENCE ARRAY!!!!      "<<endl;}
                else                    {for (i=0; i<7; i++) sequence2[i] = 0;}

                k=0; counter=0;

                for (i=(k*(x+pcm_bits+y)); i<(x+(k*(x+pcm_bits+y))); i++)
                {
                 if (counter==12) {counter=0; k++;}

                 if ((mppm er[i]==1)&&(i!=(k*(x+pcm bits+y)))&&
                             (i!=((k+1)*(x+pcm_bits+y))))
                 {
                  if ((mppm er[i-1]==1)&&(mppm er[i+1]==1)) {
                  sequence2[2]=sequence2[2]+mppm er[i+(x+pcm bits)];
                                                          }
                  if ((mppm er[i-1]==0)&&(mppm er[i+1]==1)) {
                      sequence2[4]=sequence2[4]+mppm er[i+(x+pcm_bits)];
                                                           }

                 }
                 if ((mppm er[i]==1)&&(i==(k*(x+pcm bits+y)))
                             &&(i!=((k+1)*(x+pcm_bits+y))))
                 {
                  sequence2[2]=sequence2[2]+(((pow(2,pcm bits)-1)
                               /pow(2,pcm bits))*mppm er[i+(x+pcm bits)]);
                               sequence2[5]=sequence2[5]+mppm_er[i+(x+pcm_bits)];
                 }
                 if ((mppm er[i]==1)&&(i!=(k*(x+pcm bits+y)))
                             &&(i==((k+1)*(x+pcm_bits+y))))
                 {
                  sequence2[5]=sequence2[5]+(((pow(2,pcm bits)-1)
                  /pow(2,pcm bits))*mppm er[i+(x+pcm bits)]);
                  sequence2[2]=sequence2[2]+mppm_er[i+(x+pcm_bits)];
                 }
                 counter++;
                }

                k=0; counter=0;

//-------------------------------------------------------------------------------------------//


                for (i=0; i<7; i++) {sequence2[i]=(sequence2[i]/
                (pow(2,pcm_bits)*pcm_bits));}

                if (press4==1) {
                                 if (sequence2[0]<0) {sequence2[0]=sequence2[0]*(-1);}
                                 if (sequence2[1]<0) {sequence2[1]=sequence2[1]*(-1);}
                             if (sequence2[2]<0) {sequence2[2]=sequence2[2]*(-1);}
                                 if (sequence2[3]<0) {sequence2[3]=sequence2[3]*(-1);}
                        if (sequence2[4]<0) {sequence2[4]=sequence2[4]*(-1);}
                                 if (sequence2[5]<0) {sequence2[5]=sequence2[5]*(-1);}
                                 if (sequence2[6]<0) {sequence2[6]=sequence2[6]*(-1);}
```

```
                gotoxy(0,57); cout<<"Pe:"<<endl;
                              gotoxy(3,57);cout<<sequence2[0]<<endl;
                gotoxy(0,58); cout<<"Pe.11:"<<endl;
                              gotoxy(6,58);cout<<sequence2[1]<<endl;
                gotoxy(0,59); cout<<"Pe.111:"<<endl;
                              gotoxy(7,59);cout<<sequence2[2]<<endl;
                gotoxy(0,60); cout<<"Pe.101:"<<endl;
                              gotoxy(7,60);cout<<sequence2[3]<<endl;

                              gotoxy(0,61); cout<<"Pe.1101:"<<endl;
                              gotoxy(8,61);cout<<sequence2[4]<<endl;

                gotoxy(0,62); cout<<"Pe1.1:"<<endl;
                              gotoxy(6,62);cout<<sequence2[5]<<endl;

                gotoxy(0,63); cout<<"Pe10101:"<<endl;
                              gotoxy(8,63);cout<<sequence2[6]<<endl;//15*T


//-------------------------------------------------------------------------------------------//

  /*if (sequence[6]<=0.0001)       {sequence[6]=0.12*sequence[6];}
  else if (sequence[6]<=0.001)   {sequence[6]=0.27*sequence[6];}
  else if (sequence[6]<=0.01)    {sequence[6]=0.31*sequence[6];}
  else if (sequence[6]<=0.1)     {sequence[6]=0.09*sequence[6];}
  else if (sequence[6]<=0.2)     {sequence[6]=0.05*sequence[6];}
  else if (sequence[6]<=0.3)     {sequence[6]=0.04*sequence[6];}
  else if (sequence[6]<=0.4)     {sequence[6]=0.03*sequence[6];}
  else if (sequence[6]<=0.5)     {sequence[6]=0.02*sequence[6];}
  else if (sequence[6]<=0.6)     {sequence[6]=0.02*sequence[6];}
  else if (sequence[6]<=0.7)     {sequence[6]=0.01*sequence[6];}
  else if (sequence[6]<=0.8)     {sequence[6]=0.02*sequence[6];}
  else if (sequence[6]<=0.9)     {sequence[6]=0.01*sequence[6];}
  else if (sequence[6]<=1)       {sequence[6]=0.16*sequence[6];}
       else                      {sequence[6]=0*sequence[6];}

  if (sequence[5]<=0.0001)       {sequence[5]=0.12*sequence[5];}
  else if (sequence[5]<=0.001)   {sequence[5]=0.27*sequence[5];}
  else if (sequence[5]<=0.01)    {sequence[5]=0.31*sequence[5];}
  else if (sequence[5]<=0.1)     {sequence[5]=0.09*sequence[5];}
  else if (sequence[5]<=0.2)     {sequence[5]=0.05*sequence[5];}
  else if (sequence[5]<=0.3)     {sequence[5]=0.04*sequence[5];}
  else if (sequence[5]<=0.4)     {sequence[5]=0.03*sequence[5];}
  else if (sequence[5]<=0.5)     {sequence[5]=0.02*sequence[5];}
  else if (sequence[5]<=0.6)     {sequence[5]=0.02*sequence[5];}
  else if (sequence[5]<=0.7)     {sequence[5]=0.01*sequence[5];}
  else if (sequence[5]<=0.8)     {sequence[5]=0.02*sequence[5];}
  else if (sequence[5]<=0.9)     {sequence[5]=0.01*sequence[5];}
  else if (sequence[5]<=1)       {sequence[5]=0.16*sequence[5];}
       else                      {sequence[5]=0*sequence[5];}

  if (sequence[4]<=0.0001)       {sequence[4]=0.12*sequence[4];}
  else if (sequence[4]<=0.001)   {sequence[4]=0.27*sequence[4];}
  else if (sequence[4]<=0.01)    {sequence[4]=0.31*sequence[4];}
  else if (sequence[4]<=0.1)     {sequence[4]=0.09*sequence[4];}
  else if (sequence[4]<=0.2)     {sequence[4]=0.05*sequence[4];}
  else if (sequence[4]<=0.3)     {sequence[4]=0.04*sequence[4];}
  else if (sequence[4]<=0.4)     {sequence[4]=0.03*sequence[4];}
  else if (sequence[4]<=0.5)     {sequence[4]=0.02*sequence[4];}
  else if (sequence[4]<=0.6)     {sequence[4]=0.02*sequence[4];}
  else if (sequence[4]<=0.7)     {sequence[4]=0.01*sequence[4];}
  else if (sequence[4]<=0.8)     {sequence[4]=0.02*sequence[4];}
  else if (sequence[4]<=0.9)     {sequence[4]=0.01*sequence[4];}
  else if (sequence[4]<=1)       {sequence[4]=0.16*sequence[4];}
       else                      {sequence[4]=0*sequence[4];}

  if (sequence[3]<=0.0001)       {sequence[3]=0.12*sequence[3];}
  else if (sequence[3]<=0.001)   {sequence[3]=0.27*sequence[3];}
  else if (sequence[3]<=0.01)    {sequence[3]=0.31*sequence[3];}
  else if (sequence[3]<=0.1)     {sequence[3]=0.09*sequence[3];}
  else if (sequence[3]<=0.2)     {sequence[3]=0.05*sequence[3];}
  else if (sequence[3]<=0.3)     {sequence[3]=0.04*sequence[3];}
  else if (sequence[3]<=0.4)     {sequence[3]=0.03*sequence[3];}
  else if (sequence[3]<=0.5)     {sequence[3]=0.02*sequence[3];}
  else if (sequence[3]<=0.6)     {sequence[3]=0.02*sequence[3];}
  else if (sequence[3]<=0.7)     {sequence[3]=0.01*sequence[3];}
  else if (sequence[3]<=0.8)     {sequence[3]=0.02*sequence[3];}
  else if (sequence[3]<=0.9)     {sequence[3]=0.01*sequence[3];}
  else if (sequence[3]<=1)       {sequence[3]=0.16*sequence[3];}
```

```
     else                        {sequence[3]=0*sequence[3];}

if (sequence[2]<=0.0001)         {sequence[2]=0.12*sequence[2];}
else if (sequence[2]<=0.001)     {sequence[2]=0.27*sequence[2];}
else if (sequence[2]<=0.01)      {sequence[2]=0.31*sequence[2];}
else if (sequence[2]<=0.1)       {sequence[2]=0.09*sequence[2];}
else if (sequence[2]<=0.2)       {sequence[2]=0.05*sequence[2];}
else if (sequence[2]<=0.3)       {sequence[2]=0.04*sequence[2];}
else if (sequence[2]<=0.4)       {sequence[2]=0.03*sequence[2];}
else if (sequence[2]<=0.5)       {sequence[2]=0.02*sequence[2];}
else if (sequence[2]<=0.6)       {sequence[2]=0.02*sequence[2];}
else if (sequence[2]<=0.7)       {sequence[2]=0.01*sequence[2];}
else if (sequence[2]<=0.8)       {sequence[2]=0.02*sequence[2];}
else if (sequence[2]<=0.9)       {sequence[2]=0.01*sequence[2];}
else if (sequence[2]<=1)         {sequence[2]=0.16*sequence[2];}
     else                        {sequence[2]=0*sequence[2];}

if (sequence[1]<=0.0001)         {sequence[1]=0.12*sequence[1];}
else if (sequence[1]<=0.001)     {sequence[1]=0.27*sequence[1];}
else if (sequence[1]<=0.01)      {sequence[1]=0.31*sequence[1];}
else if (sequence[1]<=0.1)       {sequence[1]=0.09*sequence[1];}
else if (sequence[1]<=0.2)       {sequence[1]=0.05*sequence[1];}
else if (sequence[1]<=0.3)       {sequence[1]=0.04*sequence[1];}
else if (sequence[1]<=0.4)       {sequence[1]=0.03*sequence[1];}
else if (sequence[1]<=0.5)       {sequence[1]=0.02*sequence[1];}
else if (sequence[1]<=0.6)       {sequence[1]=0.02*sequence[1];}
else if (sequence[1]<=0.7)       {sequence[1]=0.01*sequence[1];}
else if (sequence[1]<=0.8)       {sequence[1]=0.02*sequence[1];}
else if (sequence[1]<=0.9)       {sequence[1]=0.01*sequence[1];}
else if (sequence[1]<=1)         {sequence[1]=0.16*sequence[1];}
     else                        {sequence[1]=0*sequence[1];}

if (sequence[0]<=0.0001)         {sequence[0]=0.12*sequence[0];}
else if (sequence[0]<=0.001)     {sequence[0]=0.27*sequence[0];}
else if (sequence[0]<=0.01)      {sequence[0]=0.31*sequence[0];}
else if (sequence[0]<=0.1)       {sequence[0]=0.09*sequence[0];}
else if (sequence[0]<=0.2)       {sequence[0]=0.05*sequence[0];}
else if (sequence[0]<=0.3)       {sequence[0]=0.04*sequence[0];}
else if (sequence[0]<=0.4)       {sequence[0]=0.03*sequence[0];}
else if (sequence[0]<=0.5)       {sequence[0]=0.02*sequence[0];}
else if (sequence[0]<=0.6)       {sequence[0]=0.02*sequence[0];}
else if (sequence[0]<=0.7)       {sequence[0]=0.01*sequence[0];}
else if (sequence[0]<=0.8)       {sequence[0]=0.02*sequence[0];}
else if (sequence[0]<=0.9)       {sequence[0]=0.01*sequence[0];}
else if (sequence[0]<=1)         {sequence[0]=0.16*sequence[0];}
     else                        {sequence[0]=0*sequence[0];}

//for (i=0; i<7; i++) {gotoxy(0,87+i); cout<<sequence2[i]<<endl;}

for (i=0; i<24; i++) {standard_er[i]=(standard_er[i]*sequence2[0]);}

for (i=0; i<24; i++) {pe 11[i]=(pe 11[i]*sequence2[1]);}
for (i=0; i<24; i++) {pe 111[i]=(pe 111[i]*sequence2[2]);}
for (i=0; i<24; i++) {pe 101[i]=(pe 101[i]*sequence2[3]);}
for (i=0; i<24; i++) {pe 1101[i]=(pe 1101[i]*sequence2[4]);}
for (i=0; i<24; i++) {pe1 1[i]=(pe1 1[i]*sequence2[5]);}
for (i=0; i<24; i++) {pe_10101[i]=(pe_10101[i]*sequence2[6]);}

for (i=0; i<24; i++) {standard er[i] = standard er[i] - off standard_er[i] +
pe 11[i] - off pe 11 [24] + pe 111[i] - off pe 111[i] + pe 101[i] -
off pe 101[i] + pe 1101[i] - off_pe_1101[i]  + pe1_1[i] - off_pe1_1[i] +
pe_10101[i] - off_pe_10101[i];}

for (i=0; i<24; i++) {
          if (standard er[i]<0) {standard er[i]=standard er[i]*(-1);}
          gotoxy(0,67+i); cout<<"\n"<<standard_er[i]<<endl;
             }
for (i=0; i<24; i++) {
          if (standard er[i]<0) {minimum=0;}
          else                  {minimum=standard_er[i];}

                    pulse energy = minimum * photon energy;
          energy in frame = pulse energy * mark space correction;
                    energy per pcm bit = (energy in frame/pcm bits);
                    dBm = 10 * (log((energy per_pcm_bit*B)/pow(10,-3))/log(10));
          gotoxy(0,95+i); cout<<"\n"<<dBm<<endl;
          }*/
                   }
           delete [] sequence; sequence=0;
           delete [] sequence2; sequence2=0;
          }
```

```
//-------------------------------------------------------------------------------------//
       gotoxy(19,50);cout<<"DISPLAY INPUT ARRAYS? PRESS1,2 or 3 ELSE 8 "<<endl;
           gotoxy(64,47);cin>>press3;

       while ( (press3!=1)||(press3!=2)||(press3!=8)||(press3!=-1) )
       {
       if (press3==1)                          {
       clrscr2(); gotoxy(0,56);
           cout<<"ERASURE MPPM-DATA ARRAY                          "<<endl;
       print input arrays(mppm data,pow(2,pcm bits),x+pcm bits,x,y,
           aut pasc,aut pasc2,aut pasc3,aut pcm bits,pcm bits,pasc,pasc2,pasc3,
           start_number,end_number,press1,press2,press3,press4,press5);
       press6=0;
       do
       {
       gotoxy(19,50);
           cout<<"DO YOU WANT TO SAVE THE DATA IN A FILE?                "<<endl;
           gotoxy(33,47);cin>>press6;
       if (press6==-1) {break;}
       if (press6==2)  {break;}
       if (press6==3)  {break;}
       }while(press6!=1);      if (press6==3) {file_op1.close(); break;}
       if (press6==1) {
       k=1;
//-------------------------------------------------------------------------------------//
       for (i=0; i<(x+pcm_bits)*pow(2,pcm_bits); i++)
       {
       if (i==k*(x+pcm bits)) {file op1<<"\n"; k++;}
       file_op1<<mppm_data[i]<<"  ";
       }
       file op1.close();
       }


       break;
       }
       else if (press3==2)                     {
       clrscr2(); gotoxy(0,56);
           cout<<"WEIGTHED ERASURE ARRAY                           "<<endl;
       print input arrays(weighted er mppm data,pasc2,x+pcm bits,x,
           y,aut pasc,aut pasc2,aut pasc3,aut pcm bits,pcm bits,pasc,pasc2,pasc3,
           start_number,end_number,press1,press2,press3,press4,press5);
       press6=0;
//-------------------------------------------------------------------------------------//
       do
       {
       gotoxy(19,50);
           cout<<"DO YOU WANT TO SAVE THE DATA IN A FILE?                "<<endl;
           gotoxy(33,47);cin>>press6;
       if (press6==-1) {break;}
       if (press6==2)  {break;}
       if (press6==3)  {break;}
             }while(press6!=1);
       if (press6==3) {file_op1.close(); break;}
       if (press6==1) {
       k=1;
       for (i=0; i<(x+pcm_bits)*pasc2; i++)
       {
       if (i==k*(x+pcm bits)) {file op1<<"\n"; k++;}
       file_op1<<weighted_er_mppm_data[i]<<"  ";
       }
       file op1.close();
       }


       break;
       }
//-------------------------------------------------------------------------------------//

       else if ( (press3==8)||(press3==-1) ) {break;}
       else
           {gotoxy(19,50);
           cout<<"WRONG SELECTION.TRY AGAIN                        "<<endl;
           gotoxy(64,47);cin>>press3;}
       }

       if (press3==-1) {break;}
       if (press6==-1) {break;}

       clrscr2();
```

```
        delete [] mppm data; mppm data=0;
        delete [] weighted_er_mppm_data; weighted_er_mppm_data=0;

        delete [] mppm; mppm=0;
        delete [] data; data=0;


          }//end of if-else mppm er null statement
        }//end of erasure when y=X-1

//-------------------------------------------------------------------------------------------//

         else if (press1==2)
         {
          weighted fa mppm data = new int [x+pcm_bits];
          sequence = new float [2];
          mppm fa = new int [(x+1)*pow(2,pcm_bits)];

          if ( (mppm fa==NULL)||(sequence==NULL)||
                 (weighted fa mppm_data==NULL) )
                 {gotoxy(19,50);
                  cout<<"NOT ENOUGHT MEMORY FOR ARRAY!!!!!!!!!!"<<endl;}
          {

           flag_fa=1; k=0; l=0; counter=0;

           for (i=0; i<x+pcm bits; i++)             {weighted fa mppm data[i]=0;}
           for (i=0; i<x; i++)                      {weighted fa mppm data[i]=1;}
       for (i=x; i<x+pcm bits; i++)             {weighted_fa_mppm_data[i]=2;}
           //2*x*pcm bits*T
           for (i=0; i<(x+1)*pow(2,pcm bits); i++)  {mppm fa[i]=0;}
           for (i=0; i<2; i++)                      {sequence[i]=0;} //(x+2)*T

           for (i=0; i<pow(2,pcm_bits); i++)
           {
         for (j=0; j<x; j++) {mppm fa[(i*(x+1))+j]=
           mppm_data[(i*(x+pcm_bits))+j];} //36*T
           }

           for (i=0; i<pow(2,pcm bits); i++) {
                                  mppm fa[((i+1)*(x))+i]=pcm_bits;
                                     } //pow(2,pcm_bits)*T

//-------------------------------------------------------------------------------------------//

           press4=0;

           do
           {
            gotoxy(19,50); cout<<"DO YOU WANT TO DISPLAY THE SEQUENCES?   "<<endl;
            gotoxy(10,44);cin>>press4;
            if (press4==-1) {break;}
            if (press4==2)  {break;}
           }while(press4!=1);

           if (press4==-1) {break;}

           if (press4==1)
           {

            for (i=0; i<pow(2,pcm bits); i++) {
                   if ((k*(x+pcm_bits))==0) {l++;} k++;}

            if (l==0) {
                  for (i=1; i<pow(2,pcm bits); i++) {counter=counter+3;}
               sequence[1]=counter/(pow(2,pcm_bits)*pcm_bits);
                     }
            else {
                  for (i=1; i<pow(2,pcm bits)-1; i++) {counter=counter+3;}
                  sequence[1]=counter/(pow(2,pcm bits)*pcm bits);
                  sequence[0]=sequence[0]+(l/(pow(2,pcm_bits)*
                         (pow(2,pcm bits)*pcm bits)));
                  sequence[1]=sequence[1]+((1-pow(2,pcm_bits))/(pow(2,pcm_bits)
                  *(pow(2,pcm_bits)*pcm_bits)));
                 }

            k=0; l=0; counter=0;

            if (sequence[0]<0) {sequence[0]=sequence[0]*(-1);}
            if (sequence[1]<0) {sequence[1]=sequence[1]*(-1);}

            gotoxy(0,57); cout<<"Pf..110:"<<endl;
```

```
        gotoxy(8,57);cout<<sequence[0]<<endl;
      gotoxy(0,58); cout<<"Pf.1101:"<<endl;
        gotoxy(8,58);cout<<sequence[1]<<endl;

//-------------------------------------------------------------------------------------//

        /*if (sequence[1]<=0.0001)      {sequence[1]=0.12*sequence[1];}
        else if (sequence[1]<=0.001)  {sequence[1]=0.27*sequence[1];}
        else if (sequence[1]<=0.01)   {sequence[1]=0.31*sequence[1];}
        else if (sequence[1]<=0.1)    {sequence[1]=0.09*sequence[1];}
        else if (sequence[1]<=0.2)    {sequence[1]=0.05*sequence[1];}
        else if (sequence[1]<=0.3)    {sequence[1]=0.04*sequence[1];}
        else if (sequence[1]<=0.4)    {sequence[1]=0.03*sequence[1];}
        else if (sequence[1]<=0.5)    {sequence[1]=0.02*sequence[1];}
        else if (sequence[1]<=0.6)    {sequence[1]=0.02*sequence[1];}
        else if (sequence[1]<=0.7)    {sequence[1]=0.01*sequence[1];}
        else if (sequence[1]<=0.8)    {sequence[1]=0.02*sequence[1];}
        else if (sequence[1]<=0.9)    {sequence[1]=0.01*sequence[1];}
        else if (sequence[1]<=1)      {sequence[1]=0.16*sequence[1];}
             else                     {sequence[1]=0*sequence[1];}

        if (sequence[0]<=0.0001)      {sequence[0]=0.12*sequence[0];}
        else if (sequence[0]<=0.001)  {sequence[0]=0.27*sequence[0];}
        else if (sequence[0]<=0.01)   {sequence[0]=0.31*sequence[0];}
        else if (sequence[0]<=0.1)    {sequence[0]=0.09*sequence[0];}
        else if (sequence[0]<=0.2)    {sequence[0]=0.05*sequence[0];}
        else if (sequence[0]<=0.3)    {sequence[0]=0.04*sequence[0];}
        else if (sequence[0]<=0.4)    {sequence[0]=0.03*sequence[0];}
        else if (sequence[0]<=0.5)    {sequence[0]=0.02*sequence[0];}
        else if (sequence[0]<=0.6)    {sequence[0]=0.02*sequence[0];}
        else if (sequence[0]<=0.7)    {sequence[0]=0.01*sequence[0];}
        else if (sequence[0]<=0.8)    {sequence[0]=0.02*sequence[0];}
        else if (sequence[0]<=0.9)    {sequence[0]=0.01*sequence[0];}
        else if (sequence[0]<=1)      {sequence[0]=0.16*sequence[0];}
             else                     {sequence[0]=0*sequence[0];}

        for (i=0; i<24; i++) {pf_110[i]=(pf_110[i]*sequence[0]);}
        for (i=0; i<24; i++) {pf_1101[i]=(pf_1101[i]*sequence[1]);}

        for (i=0; i<24; i++) {pf_110[i] = pf_110[i] - off_pf_110[i] +
        pf_1101[i] - off_pf_1101[i];}

        for (i=0; i<24; i++) {
                    if (pf_110[i]<0) {pf_110[i]=pf_110[i]*(-1);}
                    gotoxy(0,65+i); cout<<"\n"<<pf_110[i]<<endl;
                       }

//-------------------------------------------------------------------------------------//

        for (i=0; i<24; i++) {
        if (pf_110[i]<0) {minimum=0;}
        else             {minimum=pf_110[i];}
        pulse energy = minimum * photon energy;
        energy in frame = pulse energy * mark space correction;
        energy_per_pcm_bit = (energy_in_frame/pcm_bits);

        dBm = 10 * (log((energy per pcm bit*B)/pow(10,-3))/log(10));
        gotoxy(0,95+i); cout<<"\n"<<dBm<<endl;
        }*/
//-------------------------------------------------------------------------------------//

        gotoxy(19,50);cout<<"DISPLAY INPUT ARRAYS? PRESS1,2 ELSE 8 "<<endl;
        gotoxy(64,47);cin>>press3;

        while ( (press3!=1)||(press3!=3)||(press3!=8)||(press3!=-1) )
        {
        if (press3==1)                          {
        clrscr2();
        gotoxy(0,56);
        cout<<"FALSE ALARM MPPM-DATA ARRAY                          "<<endl;
        print input arrays(mppm data,pow(2,pcm bits),x+pcm bits,x,y,aut pasc,
        aut pasc2,aut pasc3,aut pcm bits,pcm bits,pasc,pasc2,pasc3,start_number,
        end number,press1,press2,press3,press4,press5);
        press6=0;
//-------------------------------------------------------------------------------------//
        do
        {
        gotoxy(19,50);
        cout<<"DO YOU WANT TO SAVE THE DATA IN A FILE?                "<<endl;
        gotoxy(33,47);cin>>press6;
        if (press6==-1) {break;}
        if (press6==2)  {break;}
```

```
            if (press6==3)  {break;}
            }while(press6!=1);
            if (press6==3) {file_op1.close(); break;}
            if (press6==1) {
            k=1;
            for (i=0; i<(x+pcm_bits)*pow(2,pcm_bits); i++)
            {
            if (i==k*(x+pcm bits)) {file op1<<"\n"; k++;}
            file_op1<<mppm_data[i]<<"  ";
            }
            file op1.close();
            }
//-------------------------------------------------------------------------------------------//

            break;
            }
            else if (press3==3)                  {
            clrscr2(); gotoxy(0,56);
            cout<<"WEIGTHED FALSE ALARM ARRAY                          "<<endl;
            print input arrays(weighted fa mppm data,1,x+pcm bits,x,y,aut pasc,
            aut pasc2,aut pasc3,aut pcm bits,pcm bits,pasc,pasc2,pasc3,start_number,
            end number,press1,press2,press3,press4,press5);
            press6=0;

            do
            {
            gotoxy(19,50);
            cout<<"DO YOU WANT TO SAVE THE DATA IN A FILE?                "<<endl;
            gotoxy(33,47);cin>>press6;
            if (press6==-1) {break;}
            if (press6==2)  {break;}
            if (press6==3)  {break;}
            }while(press6!=1);
            if (press6==3) {file_op1.close(); break;}
            if (press6==1) {
            k=1;
//-------------------------------------------------------------------------------------------//
            for (i=0; i<(x+pcm_bits)*1; i++)
            {
            if (i==k*(x+pcm bits)) {file op1<<"\n"; k++;}
            file_op1<<weighted_fa_mppm_data[i]<<"  ";
            }
            file op1.close();
            }


            break;
            }
            else if ( (press3==8)||(press3==-1) ) {break;}
            else                                  {gotoxy(19,50);
            cout<<"WRONG SELECTION.TRY AGAIN                          "<<endl;
            gotoxy(64,47);cin>>press3;}
            }
//-------------------------------------------------------------------------------------------//
            if (press3==-1) {break;}
            if (press6==-1) {break;}

            clrscr2();

            //delete [] mppm data; mppm data=0;
            //delete [] weighted_fa_mppm_data; weighted_fa_mppm_data=0;

            delete [] mppm; mppm=0;

            delete [] sequence; sequence=0;
          }
         }
        }//end of false alarm when y=x-1

//-------------------------------------------------------------------------------------------//

   else if (press1==3)
   {
         mppm = new int [x*pasc2]; data = new int [pcm bits*pasc2];
         weighted_er_mppm_data = new int [(x+pcm_bits)*pasc2];

         flag_ws=0;

         if (mppm==NULL) {gotoxy(19,50);
         cout<<"NOT ENOUGHT MEMORY FOR MPPM-ER ARRAY!!!!"<<endl;}
         else { gotoxy(56,11); cout<<erasure_weight<<endl;
```

```
            gotoxy(60,12); cout<<false_alarm_weight<<endl;

            for (i=0; i<x*pasc2; i++) mppm[i] = 0;
            for (i=0; i<y-erasure_bits; i++) mppm[i] = 1;

            i=0; j=0; l=0; m=0; extra_pulses=0; k=1;

            for (m=0; m<2*pasc2; m++)
            {
              for (i=0; i<x-1; i++) if ( mppm[i+(k-1)*x]==1 ) j=i;

              for (i=1; i<=y-erasure bits; i++)
              { if (mppm[k*x-i]==0) break;
                else if (mppm[k*x-i]==1) extra pulses++;
              } if (extra_pulses==y-erasure_bits) break;

              if ( j!=(x-1) )
              {
                        for (i=0; i<j; i++) mppm[i+k*x]=mppm[i+(k-1)*x];

                mppm[j+k*x+1]=1;

                for (i=1; i<=extra pulses; i++)
                mppm[j+k*x+1+i]=1;

                l=0; extra pulses=0;
              }
              else if ( (j==(x-1) ) && (extra pulses!=0) )
             {l=extra_pulses; k--; extra_pulses=0;}
              k++;
            }
          }//end of if-else mppm null statement

        erasure_weight2=0;

//-------------------------------------------------------------------------------------//

        if (data==NULL) {gotoxy(19,50);
        cout<<"NOT ENOUGHT MEMORY FOR DATA2 ARRAY!!!!   "<<endl;}
        else { for (i=0; i<pcm bits*pasc2; i++) data[i] = 0;
               w=0; row counter=0; bit counter=0; k=1;
               for (l=1; l<=pasc2; l++)
               { for (j=x; j<(x+pcm bits); j++)
                 { for (m=0; m<pow(2,pcm bits); m++)
                   { for (i=0; i<x; i++) { if ( mppm[i+(l-1)*x]==
                     mppm_data[i+(k-1)*(pcm_bits+x)] ) row_counter++; }

                     if ( ( row counter==(x-erasure bits) ) &&
                     ( mppm data[j+(k-1)*(pcm bits+x)]==1 ) ) bit_counter++;
            if (row counter==x-erasure bits) erasure_weight2++;
                     row counter=0; k++;
                   } k=1;
                   if ( bit counter>(erasure weight2/2.0) )
                   data[w+(l-1)*pcm bits]=1;
                   else if ( bit counter<(erasure_weight2/2.0) )
                   data[w+(l-1)*pcm bits]=0;
                   else if ( bit counter=(erasure_weight2/2.0) )
                   data[w+(l-1)*pcm bits]=2;
                   bit counter=0; w++; erasure_weight2=0;
                 } k=1; w=0;
               }
             }//end of if-else data null statement

        if (weighted er mppm data==NULL)
        {gotoxy(19,50);cout<<"NOT ENOUGHT MEMORY FOR WEIGHT-ERASURE    "<<endl;}
        else { for (i=0; i<(x+pcm bits)*pasc2; i++)
        weighted_er_mppm_data[i] = 0;

               k=0;
               for (m=0; m<pasc2; m++)
               {
                for (i=0; i<x; i++)
                {
                weighted_er_mppm_data[k*(x+pcm_bits)+i] = mppm[k*x+i]; }
            k++; l++;
                }

                k=0; l=0;
                for (m=0; m<pasc2; m++)
                {
                 for (i=x; i<x+pcm_bits; i++) {
```

```
                weighted er mppm data[k*(x+pcm_bits)+i] =
                data[(l*pcm_bits)-x+i]; }
                k++; l++;
               }
             }//end of if-else weighted_er_mppm_data null statement

//-------------------------------------------------------------------------------------//

        mppm_er = new int [(x+pcm_bits+y)*pow(2,pcm_bits)];

        if (mppm er==NULL)
        {gotoxy(19,50);cout<<"NOT ENOUGHT MEMORY FOR MPPM-ER ARRAY  "<<endl;}
        else { for (i=0; i<(x+pcm_bits+y)*pow(2,pcm_bits); i++) mppm_er[i] = 0;
               flag_er=1;

               k=0;
               for (m=0; m<pow(2,pcm_bits); m++)
               {
               for (i=0; i<x; i++) mppm er[k*(x+pcm bits+y)+i]=
               mppm data[k*(x+pcm_bits)+i];
                 k++;
               }

               l=0; k=0; j=0; p=0; row counter=0;
               bit counter=0; w=y-1; //can be also w=0; w=y-1
               for ( m=0; m<pow(2,pcm bits); m++)
               { for (n=0; n<pasc2; n++)
                 { for (i=0; i<X; i++)
                 if ( mppm data[i+l*(x+pcm bits)]==
                 weighted er mppm data[i+k*(x+pcm bits)] )
                   row counter++;
                   if ( row counter==(x-erasure bits) )
                   { for ( i=x; i<x+pcm bits; i++)
                     { mppm er[i+l*(x+pcm bits)+y*p]=
                       ( mppm data[i+l*(x+pcm bits)]^
                       weighted_er_mppm_data[i+k*(x+pcm_bits)] );

               if ( mppm er[i+l*(x+pcm bits)+y*p]!=0 ) bit counter++;
                     else bit counter=bit counter;
           } mppm er[(j+1)*(x+pcm bits)+y*j+w]=
               mppm er[(j+1)*(x+pcm bits)+y*j+w]+bit counter;
            for (i=x; i<x+pcm bits; i++) mppm_er[i+l*(x+pcm_bits)+y*p]=0;
              w--; //instead of w++
             } row counter=0; bit counter=0; k++;
            } l++; k=0; p++; j++; bit counter=0; w=y-1;
              //instead of w=0 ; w=y-1
          }

//-------------------------------------------------------------------------------------//

        l1=0; zero counter=0;
        for ( m=0; m<pow(2,pcm_bits); m++)
        {
         for ( i=0; i<2; i++) {
     if (mppm er[i+m*(x+pcm bits+y)]==0) {zero_counter++;}
                                         }
         if (zero counter==2) {break;}
         else                 {zero_counter=0; l1++;}
        }

        bit counter=0; row counter=0; row counter2=0; i=0;
        for ( m=0; m<pow(2,pcm_bits); m++)
        {
           if (mppm er[i+m*(x+pcm bits+y)]==1)
               {bit_counter++;}//1st bit//
           i=x-2;
           if (mppm_er[i+m*(x+pcm_bits+y)]==1) {row_counter2++;} //Xth-1 bit//
           i=x-1;
           if (mppm_er[i+m*(x+pcm_bits+y)]==1) {row_counter++;} //Xth bit//
           i=0;
        }

        delete [] weighted_er_mppm_data; weighted_er_mppm_data=0;

        delete [] mppm; mppm=0;
        delete [] data; data=0;
       }

//-------------------------------------------------------------------------------------//
```

```
        weighted fa mppm_data = new int [x+pcm_bits];

        if ( (mppm fa==NULL)||(sequence==NULL)||(weighted fa mppm data==NULL) )
        {gotoxy(19,50);cout<<"NOT ENOUGHT MEMORY FOR ARRAY!!!!!!!!!!"<<endl;}
        {

         flag_fa=1; k=0; l=0; counter=0;

         for (i=0; i<x+pcm bits; i++)     {weighted fa mppm data[i]=0;}
         for (i=0; i<x; i++)                    {weighted fa mppm data[i]=1;}
         for (i=x; i<x+pcm bits; i++)     {weighted_fa_mppm_data[i]=2;}

        }

    mppm fa = new int [(x+1)*pow(2,pcm_bits)];

        if ( (mppm fa==NULL)||(weighted fa mppm data==NULL) )
        {gotoxy(19,50);cout<<"NOT ENOUGHT MEMORY FOR ARRAY!!!!!!!!!!"<<endl;}
         {

          for (i=0; i<(x+1)*pow(2,pcm_bits); i++)  {mppm_fa[i]=0;}

          for (i=0; i<pow(2,pcm bits); i++) {

          for (j=0; j<x; j++) {mppm fa[(i*(x+1))+j]=mppm_data[(i*(x+pcm_bits))+j];}
                                                }

          for (i=0; i<pow(2,pcm bits); i++) {
                                  mppm fa[((i+1)*(x))+i]=pcm_bits;
                                                }
          delete [] mppm; mppm=0;
         }

//-------------------------------------------------------------------------------------//


         mppm_ws = new int [(x+pcm_bits+(2*y))*pow(2,pcm_bits)];


         if (mppm ws==NULL) {
           gotoxy(19,50);cout<<"NOT ENOUGHT MEMORY FOR MPPM-WS ARRAY!!!!"<<endl;}
         else {
           for (i=0; i<(x+pcm bits+(2*y))*pow(2,pcm_bits); i++) {mppm_ws[i]=0;}
               for (i=0; i<pow(2,pcm bits); i++) {
        for (l=0; l<x; l++) {mppm ws[(i*(x+pcm bits+(2*y)))+l]=
              mppm_er[(i*(x+pcm_bits+y))+l];}
        }
        for (i=0; i<pow(2,pcm bits); i++) {
        for (l=0; l<x-1; l++) {mppm ws[((i+1)*(x+pcm bits+y))+(i*y)+l]=
              mppm_er[((i+1)*(x+pcm_bits))+(i*y)+l];}
        }
        for (i=1; i<pow(2,pcm bits); i++) {
            mppm ws[((i+1)*(x+pcm_bits+y))+(i*(y))-i]=pcm_bits;
                }
        counter=0;
               for (i=0; i<pow(2,pcm bits)-1; i++) {
                                       for (l=0; l<pcm bits; l++) {
                if ((mppm data[((i+1)*(x))+(i*pcm bits)+l]
                    ^mppm_data[((i+2)*(x))+((i+1)*pcm_bits)+l])==1) {counter++;}
               }
            mppm ws[((i+1)*(x+pcm_bits+2*y))-(i+1)]=counter;
            counter=0;
                }



               counter=0; pulse_counter=0; pulse_counter2=0; c=0;

//-------------------------------------------------------------------------------------//


               for (i=0; i<pow(2,pcm bits); i++) {
                                                  if (mppm data[(i*(x+pcm_bits))]==1)
                                                  counter++;}
           for (i=0; i<pow(2,pcm bits); i++) {
                                                 if (mppm_data[(i*(x+pcm_bits))+1]==1)
                                                  c++;}
               for (i=0; i<pow(2,pcm bits); i++) {
                  if (mppm data[((i+1)*x)+(i*pcm bits)-1]==1) pulse_counter++;}
               for (i=0; i<pow(2,pcm_bits); i++) {
```

```
                if (mppm data[((i+1)*x)+(i*pcm_bits)-2]==1) pulse_counter2++;}

              sequence2 = new float [16];

              if (sequence2==NULL) {gotoxy(19,50);
                 cout<<"NOT ENOUGHT MEMORY FOR WS SEQUENCE ARRAY!!!!      "<<endl;}
              else                 {for (i=0; i<16; i++) sequence2[i] = 0;}


              press4=0;
              gotoxy(19,50);
              cout<<"DO YOU WANT TO DISPLAY THE SEQUENCE RESULTS?      "<<endl;
              gotoxy(10,44);cin>>press4;

              while ( (press4!=1)||(press4!=2)||(press4!=-1) )
            { if (press4==-1)                       {break;}
          else if ( (press4==1)||(press4==2) ) {break;}
          else
                {gotoxy(19,50);
                 cout<<"WRONG SELECTION.TRY AGAIN                 "<<endl;
                 gotoxy(10,44);cout<<"          "<<endl;
          gotoxy(10,44);cin>>press4;}
              }

//-------------------------------------------------------------------------------------//


              if (press4==-1) {break;}

              for (i=0; i<pow(2,pcm_bits); i++)
              {
              for (l=0; l<x; l++)
              {
              if ((mppm ws[(i*(x+pcm bits+(2*y)))+l]==1)&&(l==0)) {
              if (mppm_ws[(i*(x+pcm_bits+(2*y)))+l+1]!=0)
              {
              sequence2[2]=sequence2[2]+(mppm ws[((i+1)*(x+pcm bits+y))+
              (i*(y))+l]*((pulse counter2)/(pow(2,pcm bits))));
              sequence2[11]=sequence2[11]+(mppm ws[((i+1)*(x+pcm_bits+y))+
              (i*(y))+l]*((pulse counter)/(pow(2,pcm bits))));
              sequence2[3]=sequence2[3]+(mppm ws[((i+1)*(x+pcm bits+y))+(i*(y))+l]*
              ((pow(2,pcm_bits)-pulse_counter2-pulse_counter)/pow(2,pcm_bits)));
              }
//-------------------------------------------------------------------------------------//

              else {

              sequence2[1]=sequence2[1]+(mppm ws[((i+1)*(x+pcm_bits+y))+(i*(y))+l]*
              ((pulse counter2)/pow(2,pcm bits)));
              sequence2[7]=sequence2[7]+(mppm ws[((i+1)*(x+pcm_bits+y))+(i*(y))+l]*
              ((pulse counter)/pow(2,pcm bits)));
              sequence2[0]=sequence2[0]+(mppm ws[((i+1)*(x+pcm bits+y))+(i*(y))+l]*
              ((pow(2,pcm bits)-pulse counter2-pulse counter)/pow(2,pcm bits)));

                                                                                    }
              }
              if ((mppm ws[(i*(x+pcm bits+(2*y)))+l]==1)&&(l!=0)&&(l!=(x-1))) {
              if ( (mppm ws[(i*(x+pcm bits+(2*y)))+l+1]==1)&&(mppm_ws[(i*
              (x+pcm_bits+(2*y)))+l-1]!=0) )
              {
              sequence2[2]=sequence2[2]+mppm_ws[((i+1)*(x+pcm_bits+y))+(i*(y))+l];
              }
//-------------------------------------------------------------------------------------//

                  if (mppm_ws[(i*(x+pcm_bits+(2*y)))+l+1]==0)
              {
              sequence2[1]=sequence2[1]+mppm_ws[((i+1)*(x+pcm_bits+y))+(i*(y))+l];
              }
              if ( (mppm ws[(i*(x+pcm bits+(2*y)))+l+1]==1)&&
              (mppm_ws[(i*(x+pcm_bits+(2*y)))+l-1]==0) )
              {
              sequence2[3]=sequence2[3]+mppm_ws[((i+1)*(x+pcm_bits+y))+(i*(y))+l];
              }
              }
              if ((mppm ws[(i*(x+pcm bits+(2*y)))+l]==1)&&(l==(x-1)))         {
//-------------------------------------------------------------------------------------//

              if (mppm_ws[(i*(x+pcm_bits+(2*y)))+l-1]==1)
              {
              sequence2[2]=sequence2[2]+(mppm_ws[((i+1)*(x+pcm_bits+y))+(i*(y))+l]*
              ((counter)/pow(2,pcm bits)));
              sequence2[10]=sequence2[10]+(mppm_ws[((i+1)*(x+pcm_bits+y))+(i*(y))+l]*
```

```
                ((c)/pow(2,pcm bits)));
                sequence2[3]=sequence2[3]+(mppm ws[((i+1)*(x+pcm_bits+y))+(i*(y))+l]*
                ((pow(2,pcm_bits)-counter-c)/pow(2,pcm_bits)));
                }
                else
                {
                sequence2[3]=sequence2[3]+(mppm_ws[((i+1)*(x+pcm_bits+y))+(i*(y))+l]*
                ((counter)/pow(2,pcm bits)));
                sequence2[8]=sequence2[8]+(mppm_ws[((i+1)*(x+pcm_bits+y))+(i*(y))+l]*
               ((c)/pow(2,pcm bits)));
                sequence2[0]=sequence2[0]+(mppm ws[((i+1)*(x+pcm_bits+y))+(i*(y))+l]*
               ((pow(2,pcm_bits)-counter-c)/pow(2,pcm_bits)));
                   }
                  }
                 }
                }
//-------------------------------------------------------------------------------------//


           for (i=0; i<16; i++)
                {
        sequence2[i]=(sequence2[i]/(pow(2,pcm bits)*pcm bits));
        if (sequence2[i]<0) {sequence2[i]=sequence2[i]*(-1);}
                }

                sequence2[2]=(sequence2[2]/pow(10,6));
                sequence2[3]=(sequence2[3]/pow(10,6));
                sequence2[10]=(sequence2[10]/pow(10,6));

                if (press4==1)
                {
                 gotoxy(0,58); cout<<"Ps:"<<endl;
                               gotoxy(3,58);cout<<sequence2[0]<<endl;
                 gotoxy(0,59); cout<<"Ps.111:"<<endl;
                               gotoxy(7,59);cout<<sequence2[1]<<endl;

                 gotoxy(0,60); cout<<"Ps11.1.11:"<<endl;
                               gotoxy(10,60);cout<<sequence2[2]<<endl;
                 gotoxy(0,61); cout<<"Ps1.1:"<<endl;
                               gotoxy(6,61);cout<<sequence2[3]<<endl;

                 gotoxy(0,62); cout<<"Ps.101:"<<endl;
                               gotoxy(7,62);cout<<sequence2[4]<<endl;

                 gotoxy(0,63); cout<<"Ps.1101:"<<endl;
                               gotoxy(8,63);cout<<sequence2[5]<<endl;
                 gotoxy(0,74); cout<<"Ps.11011:"<<endl;
                               gotoxy(9,74);cout<<sequence2[6]<<endl;

                        gotoxy(0,65); cout<<"Ps.11:"<<endl;
                               gotoxy(6,65);cout<<sequence2[7]<<endl;
                 gotoxy(0,66); cout<<"Ps1.11:"<<endl;
                               gotoxy(7,66);cout<<sequence2[8]<<endl;
                 gotoxy(0,67); cout<<"Ps1.1.1:"<<endl;
                               gotoxy(8,67);cout<<sequence2[9]<<endl;
                 gotoxy(0,68); cout<<"Ps11.1.1:"<<endl;
                               gotoxy(9,68);cout<<sequence2[10]<<endl;
                 gotoxy(0,69); cout<<"Ps1.1.11:"<<endl;
                               gotoxy(9,69);cout<<sequence2[11]<<endl;

                 gotoxy(0,70); cout<<"Ps.101:"<<endl;
                               gotoxy(7,70);cout<<sequence2[12]<<endl;

                 gotoxy(0,71); cout<<"Ps.1101:"<<endl;
                               gotoxy(8,71);cout<<sequence2[13]<<endl;

                 gotoxy(0,72); cout<<"Ps.11011:"<<endl;
                               gotoxy(9,72);cout<<sequence2[14]<<endl;
                 gotoxy(0,73); cout<<"Ps.110111:"<<endl;
                               gotoxy(10,73);cout<<sequence2[15]<<endl;
                }

                delete [] sequence2; sequence2=0;

                counter=0; pulse_counter=0; pulse_counter2=0; c=0;

                flag_ws=1;

//-------------------------------------------------------------------------------------//

              }
```

```
          //delete [] mppm ws;   mppm ws=0;
          //delete [] mppm_data; mppm_data=0;

    }//end of wrong slot when x=y-1
   }//end of x=y-1

        else if ( (y!=1)&&(y!=(x-1)) )
        {
         //erasure weighted array calculation for y!=x-1 and y!=1
         if (press1==1)
         {
          mppm = new int [x*pasc2]; data = new int [pcm bits*pasc2];
          weighted_er_mppm_data = new int [(x+pcm_bits)*pasc2];

          if (mppm==NULL) {gotoxy(19,50);
          cout<<"NOT ENOUGHT MEMORY FOR MPPM-ER ARRAY!!!!"<<endl;}
          else { gotoxy(56,11); cout<<erasure weight<<endl;
                 gotoxy(60,12); cout<<false_alarm_weight<<endl;

                 for (i=0; i<x*pasc2; i++) mppm[i] = 0;
                 for (i=0; i<y-erasure_bits; i++) mppm[i] = 1;

                 i=0; j=0; l=0; m=0; extra_pulses=0; k=1;

//-------------------------------------------------------------------------------------//

                 for (m=0; m<2*pasc2; m++)
                 { for (i=0; i<x-1; i++)
                 if ( mppm[i+(k-1)*x]==1 ) j=i;

                   for (i=1; i<=y-erasure bits; i++)
                   { if (mppm[k*x-i]==0) break;
                     else if (mppm[k*x-i]==1) extra pulses++;
                   } if (extra_pulses==y-erasure_bits) break;

                   if ( j!=(x-1) )
                   { for (i=0; i<j; i++) mppm[i+k*x]=mppm[i+(k-1)*x];
                     mppm[j+k*x+1]=1;

                     for (i=1; i<=extra pulses; i++)
                     mppm[j+k*x+1+i]=1;

                     l=0; extra pulses=0;
                   }
                   else if ( (j==(x-1) ) && (extra pulses!=0) )
                   {l=extra_pulses; k--; extra_pulses=0;}
                   k++;
                 }
               }//end of if-else mppm null statement

//-------------------------------------------------------------------------------------//

          erasure_weight2=0;

          if (data==NULL) {gotoxy(19,50);
          cout<<"NOT ENOUGHT MEMORY FOR DATA2 ARRAY!!!!   "<<endl;}
          else { for (i=0; i<pcm bits*pasc2; i++) data[i] = 0;
                 w=0; row counter=0; bit counter=0; k=1;
                 for (l=1; l<=pasc2; l++)
                 { for (j=x; j<(x+pcm bits); j++)
                   { for (m=0; m<pow(2,pcm bits); m++)
                     { for (i=0; i<x; i++)
                     { if ( mppm[i+(l-1)*x]==mppm data[i+(k-1)*(pcm bits+x)] )
                       row counter++; }
                   if ( ( row counter==(x-erasure bits) ) &&
                   ( mppm data[j+(k-1)*(pcm bits+x)]==1 ) ) bit_counter++;
            if (row counter==x-erasure bits) erasure_weight2++;
                        row counter=0; k++;
                     } k=1;
                     if ( bit counter>(erasure weight2/2.0) )
                     data[w+(l-1)*pcm bits]=1;
                     else if ( bit counter<(erasure_weight2/2.0) )
                     data[w+(l-1)*pcm bits]=0;
                     else if ( bit counter=(erasure_weight2/2.0) )
                     data[w+(l-1)*pcm bits]=2;
                     bit counter=0; w++; erasure_weight2=0;
                   } k=1; w=0;
                 }
               }//end of if-else data null statement
```

```
//-------------------------------------------------------------------------------------//

        if (weighted er mppm data==NULL) {gotoxy(19,50);
        cout<<"NOT ENOUGHT MEMORY FOR WEIGHT-ERASURE    "<<endl;}
        else { for (i=0; i<(x+pcm_bits)*pasc2; i++) weighted_er_mppm_data[i] = 0;

               k=0;
               for (m=0; m<pasc2; m++)
               {
                for (i=0; i<x; i++)
                { weighted er mppm data[k*(x+pcm_bits)+i] =
                                mppm[k*x+i]; }
                  k++; l++;
                }

                k=0; l=0;
                for (m=0; m<pasc2; m++)
                {
                 for (i=x; i<x+pcm bits; i++) {
                 weighted er mppm data[k*(x+pcm_bits)+i] =
                 data[(l*pcm_bits)-x+i]; }
                 k++; l++;
                }
               }//end of if-else weighted_er_mppm_data null statement

//-------------------------------------------------------------------------------------//

        mppm_er = new int [(x+pcm_bits+y)*pow(2,pcm_bits)];

        if (mppm er==NULL) {gotoxy(19,50);
        cout<<"NOT ENOUGHT MEMORY FOR MPPM-ER ARRAY  "<<endl;}
        else { for (i=0; i<(x+pcm bits+y)*pow(2,pcm bits); i++)
        mppm er[i] = 0; //(x+pcm_bits+y)*pow(2,pcm_bits)
               flag_er=1; //2T

               k=0;
               for (m=0; m<pow(2,pcm bits); m++)
               { for (i=0; i<x; i++)
               mppm er[k*(x+pcm bits+y)+i]=mppm data[k*(x+pcm bits)+i];
              //pow(2,pcm_bits)*(x+1)
                 k++;
               }

               l=0; k=0; j=0; p=0; row counter=0;
               bit counter=0; w=y-1; //can be also w=0; w=y-1
               for ( m=0; m<pow(2,pcm bits); m++)
               { for (n=0; n<pasc2; n++)
                 { for (i=0; i<x; i++) if ( mppm data[i+l*(x+pcm_bits)]==
                   weighted er mppm data[i+k*(x+pcm bits)] )
                   row counter++;
                   if ( row counter==(x-erasure bits) )
                   { for ( i=x; i<x+pcm bits; i++)
                     { mppm er[i+l*(x+pcm bits)+y*p]=
                       ( mppm data[i+l*(x+pcm bits)]^
                         weighted_er_mppm_data[i+k*(x+pcm_bits)] );

                       if ( mppm er[i+l*(x+pcm bits)+y*p]!=0 )
                                    bit counter++;
                       else bit counter=bit counter;
                     } mppm er[(j+1)*(x+pcm bits)+y*j+w]=
                         mppm er[(j+1)*(x+pcm bits)+y*j+w]+bit counter;
                     for (i=x; i<x+pcm bits; i++) mppm_er[i+l*(x+pcm_bits)+y*p]=0;
                   w--; //instead of w++
                 } row counter=0; bit counter=0; k++;
               } l++; k=0; p++; j++; bit counter=0; w=y-1; //instead of w=0 ; w=y-1
                //(6+2*pasc2+3*pasc2*pcm-bits)*pow(2,pcm_bits)
             }

//-------------------------------------------------------------------------------------//

    press5=0;
        gotoxy(19,50); cout<<"USE 2-PULSE ALGORITHM OR THE DETAILED?  "<<endl;
        gotoxy(10,39);cin>>press5;

        while ( (press5!=1)||(press5!=2)||(press5!=-1) )
        { if (press5==-1)                       {break;}
          else if ( (press5==1)||(press5==2) ) {break;}
               else                            {gotoxy(19,50);
                  cout<<"WRONG SELECTION.TRY AGAIN              "<<endl;
```

```
            gotoxy(10,39);cout<<"  "<<endl; gotoxy(10,39);cin>>press5;}
      }

     if (press5==-1) {break;}

     press4=0;
     gotoxy(19,50); cout<<"DO YOU WANT TO DISPLAY THE SEQUENCES?   "<<endl;
     gotoxy(10,44);cin>>press4;

     while ( (press4!=1)||(press4!=2)||(press4!=-1) )
     { if (press4==-1)                          {break;}
       else if ( (press4==1)||(press4==2) ) {break;}
            else                            {gotoxy(19,50);
             cout<<"WRONG SELECTION.TRY AGAIN              "<<endl;
             gotoxy(10,44);cout<<"  "<<endl; gotoxy(10,44);cin>>press4;}
     }

     if (press4==-1) {break;}

     l1=0; zero counter=0;
     for ( m=0; m<pow(2,pcm_bits); m++)
     {
      for ( i=0; i<2; i++) {

              if (mppm er[i+m*(x+pcm_bits+y)]==0) {zero_counter++;}
                                                  }
      if (zero counter==2) {break;}
      else                 {zero_counter=0; l1++;}
     }

//-------------------------------------------------------------------------------------//


     bit counter=0; row counter=0; row counter2=0; i=0;
     for ( m=0; m<pow(2,pcm_bits); m++)
     {
     if (mppm er[i+m*(x+pcm_bits+y)]==1) {bit_counter++;} //1st bit//
        i=x-2;
        if (mppm_er[i+m*(x+pcm_bits+y)]==1) {row_counter2++;} //Xth-1 bit//
        i=x-1;
        if (mppm_er[i+m*(x+pcm_bits+y)]==1) {row_counter++;} //Xth bit//
        i=0;
     }

     sequence = new float [2*y+3];
     if (sequence==NULL) {gotoxy(19,50);
          cout<<"NOT ENOUGHT MEMORY FOR ER SEQUENCE ARRAY!!!!      "<<endl;}
     else                {for (i=0; i<2*y+3; i++) sequence[i] = 0;}

     zero_counter=0; pulse_counter=0; pulse_counter2=0; l=0; j=0; flag=1; limit=0;

//-------------------------------------------------------------------------------------//


     for (w=0; w<(y-1); w++)
     {
      for (n=0; n<pow(2,pcm_bits); n++)
      {
       for (i=0; i<x; i++)
       {
        if ( (mppm_er[i+l*(x+pcm_bits+y)]==1)&&(i==0)&&(w==0) )
       {
        zero counter=0; pulse counter=0; pulse counter2=0;


        sequence[y+1]=sequence[y+1]+(mppm_er[(l+1)*(x+pcm_bits)+l*y+j]*
        (row counter2/pow(2,pcm bits)));
    sequence[1]=sequence[1]+(mppm er[(l+1)*(x+pcm bits)+l*y+j]*
        (row counter/pow(2,pcm bits)));

    sequence[0]=sequence[0]+(mppm er[(l+1)*(x+pcm bits)+l*y+j]*
        ((pow(2,pcm bits)-row counter2-row counter)/pow(2,pcm bits)));

    j++;

    }
        else if ( (mppm er[i+l*(x+pcm bits+y)]==1)&&(i!=0)&(i!=(x-1)) ){

        zero_counter=0; pulse_counter=0; pulse_counter2=0;


        for (m=i-1; m>=0; m--) {
```

```
            if (mppm er[m+l*(x+pcm bits+y)]==1) {pulse counter++;}
            else                                {zero counter=m; break;}
           }

           if ( (pulse counter==0)&&(i!=1) ) {
           pulse counter=0;
           for (m=zero counter-1; m>=0; m--) {

           if (mppm er[m+l*(x+pcm bits+y)]==1) {pulse counter++;}
           else                                {break;}
           }
           if ( (pulse counter==0)&&(W==0) ) {sequence[0]=sequence[0]+m
           ppm er[(l+1)*(x+pcm bits)+l*y+j]; j++; flag=0;}
           else if (pulse counter==limit+1)  {
           if (flag==0) {sequence[y+pulse counter]=sequence[y+pulse_counter]+
              mppm er[(l+1)*(x+pcm_bits)+l*y+j]; j++;}
           else         {
           if ( (i==2)&&(pulse counter==1) ) {
           sequence[2*y+2]=sequence[2*y+2]+(mppm er[(l+1)*(x+pcm_bits)+l*y+j]*
                (row counter2/pow(2,pcm bits)));
           sequence[y+2]=sequence[y+2]+(mppm er[(l+1)*(x+pcm bits)+l*y+j]*
                (row counter/pow(2,pcm bits)));

           sequence[y+1]=sequence[y+1]+(mppm er[(l+1)*(x+pcm bits)+l*y+j]*
                ((pow(2,pcm bits)-row counter2-row counter)/pow(2,pcm bits)));

           j++; flag=0;
           }
           else                              {
           sequence[y+1+pulse counter]=sequence[y+1+pulse counter]+
           (mppm er[(l+1)*(x+pcm bits)+l*y+j]*(row counter/pow(2,pcm_bits)));
           sequence[y+pulse counter]=sequence[y+pulse counter]+
         (mppm er[(l+1)*(x+pcm_bits)+l*y+j]*((pow(2,pcm_bits)-row_counter)
         /pow(2,pcm bits)));
           j++; flag=0;
           }
            }
             }
              }
//-------------------------------------------------------------------------------------------//



else if ( (pulse counter==0)&&(i==1)&&(w==0) ) {
sequence[y+1]=sequence[y+1]+(mppm er[(l+1)*(x+pcm bits)+l*y+j]*
        (row counter/pow(2,pcm bits)));

sequence[0]=sequence[0]+(mppm er[(l+1)*(x+pcm bits)+l*y+j]*
        ((pow(2,pcm bits)-row counter)/pow(2,pcm bits)));

j++; flag=0;
}



else if (pulse counter==limit+1) {

if (flag==0) {sequence[pulse counter]=sequence[pulse counter]+
             mppm er[(l+1)*(x+pcm bits)+l*y+j]; j++;}

else         {
sequence[pulse counter+1]=sequence[pulse counter+1]+
            (mppm er[(l+1)*(x+pcm bits)+l*y+j]
            *(row counter/pow(2,pcm bits)));
sequence[pulse counter]=sequence[pulse counter]+(mppm er[(l+1)*(x+pcm_bits)+l*y+j]
     *((pow(2,pcm_bits)-row_counter)/pow(2,pcm_bits)));
     j++;
                    }

                  }

             }
//-------------------------------------------------------------------------------------------//

else if ( (mppm er[i+l*(x+pcm bits+y)]==1)&&(i==(x-1)) )       {
             zero counter=0; pulse counter=0; pulse_counter2=0;

             for (m=i-1; m>=0; m--) {
```

```
            if (mppm er[m+l*(x+pcm bits+y)]==1) {pulse counter++;}
            else                                {zero counter=m; break;}
            }
            if ( x!=(y+1) ) {
            if (pulse counter==0)  {
            pulse counter=0;
            for (m=zero counter-1; m>=0; m--) {

            if (mppm er[m+l*(x+pcm bits+y)]==1) {pulse counter++;}
            else                                {break;}
            }
            if (pulse counter==0) {



            flag=0;

            sequence[2*y+1]=sequence[2*y+1]+(mppm_er[(l+1)*(x+pcm_bits)+l*y+j]*
            (bit counter/pow(2,pcm bits)));
            sequence[0]=sequence[0]+(mppm er[(l+1)*(x+pcm bits)+l*y+j]*
            ((pow(2,pcm_bits)-bit_counter)/pow(2,pcm_bits)));
            j++;
            }
//----------------------------------------------------------------------------------------//
            else if (pulse counter==limit+1) {




            sequence[2*y+1]=sequence[2*y+1]+(mppm_er[(l+1)*(x+pcm_bits)+l*y+j]*
            (bit counter/pow(2,pcm bits)));
            sequence[y+1]=sequence[y+1]+(mppm er[(l+1)*(x+pcm bits)+l*y+j]*
            ((pow(2,pcm_bits)-bit_counter)/pow(2,pcm_bits)));
            j++;
            }
            }
            else if (pulse counter==limit+1) {
            sequence[2*y+1]=sequence[2*y+1]+(mppm_er[(l+1)*(x+pcm_bits)+l*y+j]*
            (bit counter/pow(2,pcm bits)));
            sequence[pulse counter]=sequence[pulse_counter]+(mppm_er[(l+1)*
           (x+pcm bits)+l*y+j]*
           ((pow(2,pcm_bits)-bit_counter)/pow(2,pcm_bits)));
            j++;
            }
            }
//----------------------------------------------------------------------------------------//
            else           {
            if (pulse counter==0)  {
            pulse counter=y;

            if (pulse counter==limit+1) {




            sequence[2*y]=sequence[2*y]+(mppm_er[(l+1)*(x+pcm_bits)+l*y+j]*
            (row counter/pow(2,pcm bits)));
            sequence[2*y-1]=sequence[2*y-1]+(mppm er[(l+1)*(x+pcm bits)+l*y+j]*
            ((pow(2,pcm bits)-bit counter-row counter)/pow(2,pcm bits)));
            sequence[2*y+1]=sequence[2*y+1]+(mppm_er[(l+1)*(x+pcm_bits)+l*y+j]*
            (bit_counter/pow(2,pcm_bits)));
            j++;
            }
            }
//----------------------------------------------------------------------------------------//
            else if (pulse counter==limit+1) {


            sequence[2*y+1]=sequence[2*y+1]+(mppm_er[(l+1)*(x+pcm_bits)+l*y+j]*
           (bit counter/pow(2,pcm bits)));
            sequence[pulse counter]=sequence[pulse_counter]+
           (mppm er[(l+1)*(x+pcm bits)+l*y+j]*
           ((pow(2,pcm_bits)-bit_counter)/pow(2,pcm_bits)));
            j++;
            }


            }
```

```
            }
            if (n==pasc)                        {break;}
           }l++; j=0; if (l<=l1) {flag=1;}
                      else       {flag=0;}
            if (n==pasc)         {break;}
         }l=0; j=0; flag=1; limit++;
        }

//-------------------------------------------------------------------------------------//

       for (i=0; i<2*y+3; i++) {sequence[i]=
                                (sequence[i]/(pcm_bits*pow(2,pcm_bits)));}

       if ( (press4==1)&&(press5==2) ) {
                           gotoxy(0,57);
                                       cout<<"Pe:"<<endl;
                                       gotoxy(3,57);cout<<sequence[0]<<endl;
                    for (i=0; i<y; i++) {
                                       gotoxy(0,58+i); cout<<"Pe.11:"<<endl;
                                       gotoxy(6,58+i);cout<<sequence[i+1]<<endl;}
                    for (i=0; i<y; i++) { gotoxy(0,58+y+i);
                                       cout<<"Pe.101:"<<endl; gotoxy(7,58+y+i);
                                       cout<<sequence[i+y+1]<<endl;}

                                       gotoxy(0,58+2*y);
                                       cout<<"Pe1.1:"<<endl;
                                       gotoxy(6,58+2*y);
                                       cout<<sequence[2*y+1]<<endl;
                                       gotoxy(0,58+2*y+1);
                                       cout<<"Pe10101:"<<endl;
                                       gotoxy(8,58+2*y+1);
                                       cout<<sequence[2*(y+1)]<<endl;


                                       k=1; n=0; w=0; z=0;
      for (i=0; i<2*y+3; i++) {
                  gotoxy(0+w,87+n); cout<<sequence[i]<<endl;

                                 w=w+13;
            if (i==k*y+z) {k++; n++; w=0; z++;}
            }
            delete [] sequence; sequence=0;
            }

//-------------------------------------------------------------------------------------//


       else if (press5==1)
            {

             sequence2 = new float [7];
             if (sequence2==NULL) {gotoxy(19,50);
             cout<<"NOT ENOUGHT MEMORY FOR ER SEQUENCE ARRAY!!!!      "<<endl;}
             else                 {for (i=0; i<7; i++) sequence2[i] = 0;}

             for (i=0; i<2; i++) {sequence2[i]=sequence[i];}
             sequence2[5]=sequence[2*y+1];
             sequence2[6]=sequence[2*y+2];
             sequence2[3]=sequence[y+1];

        for (i=2; i<y+1; i++)     {sequence2[2]=sequence2[2]+sequence[i];}
             for (i=y+2; i<2*y+1; i++)
             {sequence2[4]=sequence2[4]+sequence[i];}

             if (press4==1) {
                             if (sequence2[0]<0) {sequence2[0]=sequence2[0]*(-1);}
                             if (sequence2[1]<0) {sequence2[1]=sequence2[1]*(-1);}
                        if (sequence2[2]<0) {sequence2[2]=sequence2[2]*(-1);}
                             if (sequence2[3]<0) {sequence2[3]=sequence2[3]*(-1);}
                   if (sequence2[4]<0) {sequence2[4]=sequence2[4]*(-1);}
                             if (sequence2[5]<0) {sequence2[5]=sequence2[5]*(-1);}
                             if (sequence2[6]<0) {sequence2[6]=sequence2[6]*(-1);}

                 gotoxy(0,57); cout<<"Pe:"<<endl;
                             gotoxy(3,57);cout<<sequence2[0]<<endl;
                 gotoxy(0,58); cout<<"Pe.11:"<<endl;
                             gotoxy(6,58);cout<<sequence2[1]<<endl;
                 gotoxy(0,59); cout<<"Pe.111:"<<endl;
                             gotoxy(7,59);cout<<sequence2[2]<<endl;
                 gotoxy(0,60); cout<<"Pe.101:"<<endl;
                             gotoxy(7,60);cout<<sequence2[3]<<endl;
```

```
                        gotoxy(0,61); cout<<"Pe.1101:"<<endl;
                        gotoxy(8,61);cout<<sequence2[4]<<endl;

                gotoxy(0,62); cout<<"Pe1.1:"<<endl;
                        gotoxy(6,62);cout<<sequence2[5]<<endl;

                gotoxy(0,63); cout<<"Pe10101:"<<endl;
                        gotoxy(8,63);cout<<sequence2[6]<<endl;

//-------------------------------------------------------------------------------------//

    /*if (sequence[6]<=0.0001)       {sequence[6]=0.12*sequence[6];}
    else if (sequence[6]<=0.001)   {sequence[6]=0.27*sequence[6];}
    else if (sequence[6]<=0.01)    {sequence[6]=0.31*sequence[6];}
    else if (sequence[6]<=0.1)     {sequence[6]=0.09*sequence[6];}
    else if (sequence[6]<=0.2)     {sequence[6]=0.05*sequence[6];}
    else if (sequence[6]<=0.3)     {sequence[6]=0.04*sequence[6];}
    else if (sequence[6]<=0.4)     {sequence[6]=0.03*sequence[6];}
    else if (sequence[6]<=0.5)     {sequence[6]=0.02*sequence[6];}
    else if (sequence[6]<=0.6)     {sequence[6]=0.02*sequence[6];}
    else if (sequence[6]<=0.7)     {sequence[6]=0.01*sequence[6];}
    else if (sequence[6]<=0.8)     {sequence[6]=0.02*sequence[6];}
    else if (sequence[6]<=0.9)     {sequence[6]=0.01*sequence[6];}
    else if (sequence[6]<=1)       {sequence[6]=0.16*sequence[6];}
     else                      {sequence[6]=0*sequence[6];}

    if (sequence[5]<=0.0001)       {sequence[5]=0.12*sequence[5];}
    else if (sequence[5]<=0.001)   {sequence[5]=0.27*sequence[5];}
    else if (sequence[5]<=0.01)    {sequence[5]=0.31*sequence[5];}
    else if (sequence[5]<=0.1)     {sequence[5]=0.09*sequence[5];}
    else if (sequence[5]<=0.2)     {sequence[5]=0.05*sequence[5];}
    else if (sequence[5]<=0.3)     {sequence[5]=0.04*sequence[5];}
    else if (sequence[5]<=0.4)     {sequence[5]=0.03*sequence[5];}
    else if (sequence[5]<=0.5)     {sequence[5]=0.02*sequence[5];}
    else if (sequence[5]<=0.6)     {sequence[5]=0.02*sequence[5];}
    else if (sequence[5]<=0.7)     {sequence[5]=0.01*sequence[5];}
    else if (sequence[5]<=0.8)     {sequence[5]=0.02*sequence[5];}
    else if (sequence[5]<=0.9)     {sequence[5]=0.01*sequence[5];}
    else if (sequence[5]<=1)       {sequence[5]=0.16*sequence[5];}
         else                      {sequence[5]=0*sequence[5];}

    if (sequence[4]<=0.0001)       {sequence[4]=0.12*sequence[4];}
    else if (sequence[4]<=0.001)   {sequence[4]=0.27*sequence[4];}
    else if (sequence[4]<=0.01)    {sequence[4]=0.31*sequence[4];}
    else if (sequence[4]<=0.1)     {sequence[4]=0.09*sequence[4];}
    else if (sequence[4]<=0.2)     {sequence[4]=0.05*sequence[4];}
    else if (sequence[4]<=0.3)     {sequence[4]=0.04*sequence[4];}
    else if (sequence[4]<=0.4)     {sequence[4]=0.03*sequence[4];}
    else if (sequence[4]<=0.5)     {sequence[4]=0.02*sequence[4];}
    else if (sequence[4]<=0.6)     {sequence[4]=0.02*sequence[4];}
    else if (sequence[4]<=0.7)     {sequence[4]=0.01*sequence[4];}
    else if (sequence[4]<=0.8)     {sequence[4]=0.02*sequence[4];}
    else if (sequence[4]<=0.9)     {sequence[4]=0.01*sequence[4];}
    else if (sequence[4]<=1)       {sequence[4]=0.16*sequence[4];}
         else                      {sequence[4]=0*sequence[4];}


    if (sequence[3]<=0.0001)       {sequence[3]=0.12*sequence[3];}
    else if (sequence[3]<=0.001)   {sequence[3]=0.27*sequence[3];}
    else if (sequence[3]<=0.01)    {sequence[3]=0.31*sequence[3];}
    else if (sequence[3]<=0.1)     {sequence[3]=0.09*sequence[3];}
    else if (sequence[3]<=0.2)     {sequence[3]=0.05*sequence[3];}
    else if (sequence[3]<=0.3)     {sequence[3]=0.04*sequence[3];}
    else if (sequence[3]<=0.4)     {sequence[3]=0.03*sequence[3];}
    else if (sequence[3]<=0.5)     {sequence[3]=0.02*sequence[3];}
    else if (sequence[3]<=0.6)     {sequence[3]=0.02*sequence[3];}
    else if (sequence[3]<=0.7)     {sequence[3]=0.01*sequence[3];}
    else if (sequence[3]<=0.8)     {sequence[3]=0.02*sequence[3];}
    else if (sequence[3]<=0.9)     {sequence[3]=0.01*sequence[3];}
    else if (sequence[3]<=1)       {sequence[3]=0.16*sequence[3];}
     else                      {sequence[3]=0*sequence[3];}

    if (sequence[2]<=0.0001)       {sequence[2]=0.12*sequence[2];}
    else if (sequence[2]<=0.001)   {sequence[2]=0.27*sequence[2];}
    else if (sequence[2]<=0.01)    {sequence[2]=0.31*sequence[2];}
    else if (sequence[2]<=0.1)     {sequence[2]=0.09*sequence[2];}
    else if (sequence[2]<=0.2)     {sequence[2]=0.05*sequence[2];}
    else if (sequence[2]<=0.3)     {sequence[2]=0.04*sequence[2];}
    else if (sequence[2]<=0.4)     {sequence[2]=0.03*sequence[2];}
    else if (sequence[2]<=0.5)     {sequence[2]=0.02*sequence[2];}
```

```
      else if (sequence[2]<=0.6)     {sequence[2]=0.02*sequence[2];}
      else if (sequence[2]<=0.7)     {sequence[2]=0.01*sequence[2];}
      else if (sequence[2]<=0.8)     {sequence[2]=0.02*sequence[2];}
      else if (sequence[2]<=0.9)     {sequence[2]=0.01*sequence[2];}
      else if (sequence[2]<=1)       {sequence[2]=0.16*sequence[2];}
           else                      {sequence[2]=0*sequence[2];}

      if (sequence[1]<=0.0001)       {sequence[1]=0.12*sequence[1];}
      else if (sequence[1]<=0.001)   {sequence[1]=0.27*sequence[1];}
      else if (sequence[1]<=0.01)    {sequence[1]=0.31*sequence[1];}
      else if (sequence[1]<=0.1)     {sequence[1]=0.09*sequence[1];}
      else if (sequence[1]<=0.2)     {sequence[1]=0.05*sequence[1];}
      else if (sequence[1]<=0.3)     {sequence[1]=0.04*sequence[1];}
      else if (sequence[1]<=0.4)     {sequence[1]=0.03*sequence[1];}
      else if (sequence[1]<=0.5)     {sequence[1]=0.02*sequence[1];}
      else if (sequence[1]<=0.6)     {sequence[1]=0.02*sequence[1];}
      else if (sequence[1]<=0.7)     {sequence[1]=0.01*sequence[1];}
      else if (sequence[1]<=0.8)     {sequence[1]=0.02*sequence[1];}
      else if (sequence[1]<=0.9)     {sequence[1]=0.01*sequence[1];}
      else if (sequence[1]<=1)       {sequence[1]=0.16*sequence[1];}
           else                      {sequence[1]=0*sequence[1];}

      if (sequence[0]<=0.0001)       {sequence[0]=0.12*sequence[0];}
      else if (sequence[0]<=0.001)   {sequence[0]=0.27*sequence[0];}
      else if (sequence[0]<=0.01)    {sequence[0]=0.31*sequence[0];}
      else if (sequence[0]<=0.1)     {sequence[0]=0.09*sequence[0];}
      else if (sequence[0]<=0.2)     {sequence[0]=0.05*sequence[0];}
      else if (sequence[0]<=0.3)     {sequence[0]=0.04*sequence[0];}
      else if (sequence[0]<=0.4)     {sequence[0]=0.03*sequence[0];}
      else if (sequence[0]<=0.5)     {sequence[0]=0.02*sequence[0];}
      else if (sequence[0]<=0.6)     {sequence[0]=0.02*sequence[0];}
      else if (sequence[0]<=0.7)     {sequence[0]=0.01*sequence[0];}
      else if (sequence[0]<=0.8)     {sequence[0]=0.02*sequence[0];}
      else if (sequence[0]<=0.9)     {sequence[0]=0.01*sequence[0];}
      else if (sequence[0]<=1)       {sequence[0]=0.16*sequence[0];}
          else                      {sequence[0]=0*sequence[0];}

      //for (i=0; i<7; i++) {gotoxy(0,87+i); cout<<sequence2[i]<<endl;}

//-------------------------------------------------------------------------------------------//

         for (i=0; i<24; i++) {standard er[i]=(standard_er[i]*sequence2[0]);}

                 for (i=0; i<24; i++) {pe 11[i]=(pe 11[i]*sequence2[1]);}
                 for (i=0; i<24; i++) {pe 111[i]=(pe 111[i]*sequence2[2]);}
                 for (i=0; i<24; i++) {pe 101[i]=(pe 101[i]*sequence2[3]);}
                 for (i=0; i<24; i++) {pe 1101[i]=(pe 1101[i]*sequence2[4]);}
      for (i=0; i<24; i++) {pe1 1[i]=(pe1 1[i]*sequence2[5]);}
                 for (i=0; i<24; i++) {pe_10101[i]=(pe_10101[i]*sequence2[6]);}

                 for (i=0; i<24; i++) {standard er[i] = standard er[i] -
                 off standard er[i] + pe 11[i] - off pe 11 [24] + pe 111[i] -
                 off pe 111[i] + pe_101[i] - off_pe_101[i] + pe_1101[i] -
                 off pe 1101[i]
                 + pe1_1[i] - off_pe1_1[i] + pe_10101[i] - off_pe_10101[i];}

                 for (i=0; i<24; i++) {
      if (standard er[i]<0) {standard er[i]=standard er[i]*(-1);}
       gotoxy(0,67+i); cout<<"\n"<<standard_er[i]<<endl;
      }

                 for (i=0; i<24; i++) {
                       if (standard er[i]<0) {minimum=0;}
                       else                  {minimum=standard_er[i];}

                                             pulse energy = minimum * photon_energy;
                       energy in frame = pulse energy *
                                             mark space correction;
                                             energy_per_pcm_bit = (energy_in_frame/pcm_bits);

                 dBm = 10 * (log((energy per pcm_bit*B)/pow(10,-3))/log(10));
      gotoxy(0,95+i); cout<<"\n"<<dBm<<endl;
                                                  }*/
                            }
                delete [] sequence; sequence=0;
                delete [] sequence2; sequence2=0;
               }

//-------------------------------------------------------------------------------------------//


      gotoxy(19,50);cout<<"DISPLAY INPUT ARRAYS? PRESS1,2 or 3 ELSE 8 "<<endl;
```

```
    gotoxy(64,47);cin>>press3;

  while ( (press3!=1)||(press3!=2)||(press3!=8)||(press3!=-1) )
  {
   if (press3==1)                          {
   clrscr2(); gotoxy(0,56);
           cout<<"ERASURE MPPM-DATA ARRAY                              "<<endl;
           print input arrays(mppm data,pow(2,pcm bits),
           x+pcm bits,x,y,aut pasc,aut pasc2,aut pasc3,aut pcm bits,
           pcm bits,pasc,pasc2,pasc3,start_number,end_number,press1,
           press2,press3,press4,press5);
   press6=0;
           do
   {
   gotoxy(19,50);
           cout<<
           "DO YOU WANT TO SAVE THE DATA IN A FILE?                "<<endl;
           gotoxy(33,47);cin>>press6;
   if (press6==-1) {break;}
   if (press6==2)  {break;}
   if (press6==3)  {break;}
           }while(press6!=1);


          if (press6==3) {file_op1.close(); break;}
          if (press6==1) {
          k=1;
          for (i=0; i<(x+pcm_bits)*pow(2,pcm_bits); i++)
          {
           if (i==k*(x+pcm bits)) {file op1<<"\n"; k++;}
           file_op1<<mppm_data[i]<<"  ";
          }
          file op1.close();
         }


          break;
         }
         else if (press3==2)                    {
         clrscr2(); gotoxy(0,56);
         cout<<"WEIGTHED ERASURE ARRAY                             "<<endl;
         print input arrays(weighted er mppm data,pasc2,x+pcm bits,x,y,
         aut pasc,aut pasc2,aut pasc3,aut pcm bits,pcm bits,pasc,pasc2,
         pasc3,start_number,end_number,press1,press2,press3,press4,press5);
         press6=0;

//-------------------------------------------------------------------------------------//

          do
          {
          gotoxy(19,50);
              cout<<"DO YOU WANT TO SAVE THE DATA IN A FILE?                "<<endl;
              gotoxy(33,47);cin>>press6;
          if (press6==-1) {break;}

              if (press6==2)  {break;}
          if (press6==3)  {break;}
              }while(press6!=1);
              if (press6==3) {file_op1.close(); break;}
          if (press6==1) {
         k=1;
         for (i=0; i<(x+pcm_bits)*pasc2; i++)
         {
          if (i==k*(x+pcm bits)) {file op1<<"\n"; k++;}
          file_op1<<weighted_er_mppm_data[i]<<"  ";
         }
         file op1.close();
          }


           break;
          }
          else if ( (press3==8)||(press3==-1) ) {break;}
          else {gotoxy(19,50);
              cout<<"WRONG SELECTION.TRY AGAIN                        "<<endl;
              gotoxy(64,47);cin>>press3;}
  }

  if (press3==-1) {break;}
  if (press6==-1) {break;}

  clrscr2();
```

```
       delete [] mppm data; mppm data=0;
       delete [] weighted_er_mppm_data; weighted_er_mppm_data=0;

       delete [] mppm; mppm=0;
       delete [] data; data=0;

//-------------------------------------------------------------------------------------//

        }//end of if-else mppm er null statement
  }//end of if-press1(1) for y!=1 or y!=(x-1)

  //weighted fa array calculation
  else if (press1==2)
  { mppm = new int [x*pasc3]; data = new int [pcm bits*pasc3];
    weighted_fa_mppm_data = new int [(x+pcm_bits)*pasc3];

    if (mppm==NULL) {gotoxy(19,50);
         cout<<"NOT ENOUGHT MEMORY FOR MPPM3 ARRAY!!!!        "<<endl;}
    else { gotoxy(56,11); cout<<erasure weight<<endl;
           gotoxy(60,12); cout<<false_alarm_weight<<endl;

           for (i=0; i<x*pasc3; i++) mppm[i] = 0;
           for (i=0; i<y+false_alarm_bits; i++) mppm[i] = 1;

           i=0; j=0; l=0; m=0; extra pulses=0; k=1;
           for (m=0; m<2*pasc3; m++)
           { for (i=0; i<x-1; i++) if ( mppm[i+(k-1)*x]==1 ) j=i;

             for (i=1; i<=y+false alarm bits; i++)
             { if (mppm[k*x-i]==0) break;
               else if (mppm[k*x-i]==1) extra pulses++;
             } if (extra_pulses==y+false_alarm_bits) break;

             if ( j!=(x-1) )
             { for (i=0; i<j; i++) mppm[i+k*x]=mppm[i+(k-1)*x];
                mppm[j+k*x+1]=1;

               for (i=1; i<=extra pulses; i++)
                mppm[j+k*x+1+i]=1;

               l=0; extra pulses=0;
             }
             else if ( (j==(x-1) ) && (extra pulses!=0) )
                    {l=extra_pulses; k--; extra_pulses=0;}

             k++;
           }

         }//end of if-else mppm null statement

//-------------------------------------------------------------------------------------//

    false alarm weight2=0;
    if (data==NULL) {gotoxy(19,50);
    cout<<"NOT ENOUGHT MEMORY FOR DATA3 ARRAY!!!!      "<<endl;}
    else { for (i=0; i<pcm bits*pasc3; i++) data[i] = 0;
           w=0; row counter=0; bit counter=0; k=1;
           for (l=1; l<=pasc3; l++)
           { for (j=x; j<(x+pcm bits); j++)
             { for (m=0; m<pow(2,pcm bits); m++)
               { for (i=0; i<x; i++) { if ( mppm[i+(l-1)*x]==
                 mppm_data[i+(k-1)*(pcm_bits+x)] ) row_counter++; }

                 if ( ( row counter==(x-false alarm bits) ) &&
                 ( mppm data[j+(k-1)*(pcm bits+x)]==1 ) ) bit counter++;
                 if ( row counter==(x-false_alarm_bits) ) false_alarm_weight2++;
                      row_counter=0; k++;
               } k=1;
               if      ( bit counter > (false_alarm_weight2/2.0) )
               data[w+(l-1)*pcm bits]=1;
               else if ( bit counter < (false_alarm_weight2/2.0) )
               data[w+(l-1)*pcm bits]=0;
               else if ( bit counter = (false_alarm_weight2/2.0) )
               data[w+(l-1)*pcm bits]=2;
               bit counter=0; w++; false_alarm_weight2=0;
             } k=1; w=0;
           }

         }//end of if-else data null statement
```

```
    if (weighted fa mppm data==NULL) {
    gotoxy(19,50);cout<<"NOT ENOUGHT MEMORY FOR WEIGHT-FA ALARM     "<<endl;}
    else { for (i=0; i<(x+pcm_bits)*pasc3; i++) weighted_fa_mppm_data[i] = 0;

           k=0;
           for (m=0; m<pasc3; m++)
           { for (i=0; i<x; i++)
             { weighted_fa_mppm_data[k*(x+pcm_bits)+i] = mppm[k*x+i]; }
             k++; l++;
           }

           k=0; l=0;
           for (m=0; m<pasc3; m++)
           { for (i=x; i<x+pcm bits; i++)
             { weighted fa mppm data[k*(x+pcm bits)+i] =
                      data[(l*pcm_bits)-x+i]; }
             k++; l++;
           }

         }//end of if-else weighted_fa_mppm_data null statement

//-------------------------------------------------------------------------------------//


    mppm_fa = new int [(2*x+pcm_bits-y)*pow(2,pcm_bits)];

    if (mppm fa==NULL) {
          gotoxy(19,50);cout<<"NOT ENOUGHT MEMORY FOR MPPM-FA ARRAY    "<<endl;}
    else { for (i=0; i<(2*x+pcm_bits-y)*pow(2,pcm_bits); i++) mppm_fa[i] = 0;

           flag_fa=1;

           k=0;
           for (m=0; m<pow(2,pcm bits); m++)
           { for (i=0; i<x; i++) mppm fa[k*(2*x+pcm bits-y)+i]=
                    mppm data[k*(x+pcm_bits)+i];
                              k++;
           }

           l=0; k=0; j=0; row counter=0; bit counter=0; w=0;
           for ( m=0; m<pow(2,pcm bits); m++)
           { for (n=0; n<pasc3; n++)
             { for (i=0; i<x; i++) if ( mppm data[i+l*(x+pcm bits)]==
               weighted_fa_mppm_data[i+k*(x+pcm_bits)] ) row_counter++;

                 if ( row counter==(x-false alarm bits) )
                 { for ( i=x; i<x+pcm bits; i++)
                   { mppm fa[i+l*(2*x+pcm bits-y)]=(
                       mppm data[i+l*(x+pcm bits)]^
                       weighted_fa_mppm_data[i+k*(x+pcm_bits)] );

                if ( mppm fa[i+l*(2*x+pcm bits-y)]!=0 ) bit counter++;
                   else bit counter=bit counter;
                 } mppm fa[(j+1)*(x+pcm bits)+j*(x-y)+w]=
                 mppm fa[(j+1)*(x+pcm bits)+j*(x-y)+w]+bit counter;
                 for (i=x; i<x+pcm_bits; i++) mppm_fa[i+l*(2*x+pcm_bits-y)]=0;
                 w++;
               }    row counter=0; bit counter=0; k++;
             } l++; k=0; j++; bit_counter=0; w=0;
           }

           press5=0;
           gotoxy(19,50); cout<<"USE 2-PULSE ALGORITHM OR THE DETAILED?  "<<endl;
           gotoxy(10,39);cin>>press5;

           while ( (press5!=1)||(press5!=2)||(press5!=-1) )
           { if (press5==-1)                          {break;}
             else if ( (press5==1)||(press5==2) ) {break;}
             else                              {gotoxy(19,50);
                     cout<<"WRONG SELECTION.TRY AGAIN              "<<endl;
                     gotoxy(10,39);cout<<"  "<<endl;
           gotoxy(10,39);cin>>press5;}
           }

           if (press5==-1) {break;}

//-------------------------------------------------------------------------------------//


           press4=0;
           gotoxy(19,50);
```

```
            cout<<"DO YOU WANT TO DISPLAY THE SEQUENCE RESULTS?      "<<endl;
            gotoxy(10,44);cin>>press4;

            while ( (press4!=1)||(press4!=2)||(press4!=-1) )
            { if (press4==-1)                          {break;}
              else if ( (press4==1)||(press4==2) ) {break;}
              else
                    {gotoxy(19,50);
                    cout<<"WRONG SELECTION.TRY AGAIN                 "<<endl;
                    gotoxy(10,44);cout<<"          "<<endl;

                    gotoxy(10,44);cin>>press4;}
            }

            if (press4==-1) {break;}

            l1=0; zero counter=0;
            for ( m=0; m<pow(2,pcm_bits); m++)
            {
             for ( i=0; i<2; i++) {

            if (mppm fa[i+m*(2*x+pcm bits-y)]==0) {zero_counter++;}
                                  }
             if (zero counter==2) {break;}
             else                 {zero_counter=0; l1++;}
            }

//-------------------------------------------------------------------------------------------//


            bit counter=0; row counter=0; i=0;
            for ( m=0; m<pow(2,pcm bits); m++)
            {  if (mppm fa[i+m*(2*x+pcm bits-y)]==1)
               {bit counter++;}//1-bit_counter=11//
               i=x-1;
               if (mppm fa[i+m*(2*x+pcm_bits-y)]==1) {row_counter++;}
               //1-row_counter=9//
               i=0;
            }

             sequence = new float [2*(y+1)];
             if (sequence==NULL) {gotoxy(19,50);
             cout<<"NOT ENOUGHT MEMORY FOR FA SEQUENCE ARRAY!!!!      "<<endl;}
             else          {for (i=0; i<2*(y+1); i++) sequence[i] = 0;}

             zero_counter=0; pulse_counter=0; l=0; j=0; flag=1; limit=0;

             for (w=0; w<y+1; w++)
             {
              for (n=0; n<pow(2,pcm_bits); n++)
              {
               for (i=0; i<x; i++)
               {
                if ( (mppm_fa[i+l*(2*x+pcm_bits-y)]==0)&&(i==0)&&(w==0) )
               {
                zero counter=0; pulse counter=0;
            if (mppm_fa[i+l*(2*x+pcm_bits-y)+1]==0)
                {

//-------------------------------------------------------------------------------------------//

          sequence[1]=sequence[1]+(mppm fa[l*(2*x+pcm bits-y)+x+pcm_bits+j]*
                              (row counter/pow(2,pcm bits)));
          sequence[0]=sequence[0]+(mppm fa[l*(2*x+pcm bits-y)+x+pcm bits+j]*
                              ((pow(2,pcm bits)-row counter)/pow(2,pcm bits)));
             }

             else                                      {
          sequence[y+2]=sequence[y+2]+(mppm fa[l*(2*x+pcm bits-y)+x+pcm_bits+j]*
                              (row counter/pow(2,pcm bits)));
          sequence[0]=sequence[0]+(mppm fa[l*(2*x+pcm bits-y)+x+pcm bits+j]*
                              ((pow(2,pcm_bits)-row_counter)/pow(2,pcm_bits)));
           }
           j++; flag=0;
          }
                 else if ( (mppm fa[i+l*(2*x+pcm bits-y)]==0)
                           &&(i!=0)&(i!=(x-1)) ){
           zero counter=0; pulse_counter=0;

          for (m=i-1; m>=0; m--)    {
          if (mppm fa[m+l*(2*x+pcm bits-y)]==1) {pulse counter++;}
          else                                  {flag=0; break;}
```

```
        }
        if ( (pulse counter==limit)&&(flag==0) ) {
        if (pulse counter==0)
                {sequence[0]=sequence[0]+mppm_fa[l*(2*x+pcm_bits-y)+x+pcm_bits+j];}
        else {
        if (mppm fa[i+l*(2*x+pcm bits-y)+1]==0) {sequence[limit]=
                sequence[limit]+mppm fa[l*(2*x+pcm bits-y)+x+pcm bits+j];}
        else                                    {sequence[limit+y+1]=
                sequence[limit+y+1]+mppm_fa[l*(2*x+pcm_bits-y)+x+pcm_bits+j];}
         }
          }

//-------------------------------------------------------------------------------------//



        else if ( (pulse counter==limit)&&(flag==1) ) {
        if (mppm fa[i+l*(2*x+pcm bits-y)+1]==0) {
        sequence[limit+1]=sequence[limit+1]+
                (mppm fa[l*(2*x+pcm bits-y)+x+pcm_bits+j]*
                (row counter/pow(2,pcm bits)));
        sequence[limit]=sequence[limit]+
                 (mppm fa[l*(2*x+pcm bits-y)+x+pcm bits+j]*
                ((pow(2,pcm_bits)-row_counter)/pow(2,pcm_bits)));
        }
        else                                    {
        sequence[limit+y+2]=sequence[limit+y+2]+
                (mppm fa[l*(2*x+pcm bits-y)+x+pcm_bits+j]*
                (row counter/pow(2,pcm bits)));
        sequence[limit+y+1]=sequence[limit+y+1]+
                (mppm fa[l*(2*x+pcm bits-y)+x+pcm bits+j]*
                ((pow(2,pcm_bits)-row_counter)/pow(2,pcm_bits)));
        }
        }

        j++; flag=0;

        }
        else if ( (mppm fa[i+l*(2*x+pcm bits-y)]==0)&&(i==(x-1)) ){
        zero counter=0; pulse_counter=0;

        for (m=i-1; m>=0; m--)    {
        if (mppm fa[m+l*(2*x+pcm bits-y)]==1) {pulse counter++;}
        else                                  {break;}
        }
                if ( (pulse counter==limit)&&(x!=(y+1)) )       {
        if (pulse counter==0) {sequence[0]=sequence[0]+
                mppm fa[l*(2*x+pcm bits-y)+x+pcm bits+j]; flag=0;}
        else {

        sequence[limit+y+1]=sequence[limit+y+1]+
                (mppm fa[l*(2*x+pcm bits-y)+x+pcm_bits+j]*
                (bit counter/pow(2,pcm bits)));
        sequence[limit]=sequence[limit]+(mppm fa[l*
                 (2*x+pcm bits-y)+x+pcm_bits+j]*((pow(2,pcm_bits)-bit_counter)
                 /pow(2,pcm_bits)));
         }
          }
        else if ( (pulse counter==limit)&&(x==(y+1)) ) {
        sequence[limit+1]=sequence[limit+1]+(mppm fa[l*
                (2*x+pcm bits-y)+x+pcm bits+j]*(row counter/pow(2,pcm_bits)));
                sequence[limit+y+1]=sequence[limit+y+1]+
                (mppm fa[l*(2*x+pcm bits-y)+x+pcm_bits+j]*
                (bit counter/pow(2,pcm bits)));
                sequence[limit]=sequence[limit]+
                (mppm fa[l*(2*x+pcm bits-y)+x+pcm bits+j]
               *((pow(2,pcm_bits)-bit_counter-row_counter)/pow(2,pcm_bits)));
         }
                  j++; flag=0;
                                                                                }
                if (n==pasc)    {break;}
               }l++; j=0; if (l<=l1) {flag=1;}
                          else       {flag=0;}
                if (n==pasc)         {break;}
             }l=0; j=0; flag=1; limit++;
            }

//-------------------------------------------------------------------------------------//

            for (i=0; i<2*(y+1); i++)
```

```
            {sequence[i]=(sequence[i]/((x-y)*pcm_bits*pow(2,pcm_bits)));}

            if ( (press4==1)&&(press5==2) ) {
                                gotoxy(0,57); cout<<"Pf:"<<endl;
                                              gotoxy(3,57);cout<<sequence[0]<<endl;
                             for (i=0; i<(y+1); i++) { gotoxy(0,58+i);
                                              cout<<"Pf.10:"<<endl; gotoxy(6,58+i);
                                              cout<<sequence[i+1]<<endl;}
                        for (i=0; i<y; i++)
            { gotoxy(0,58+y+1+i); cout<<"Pf.101:"<<endl;
              gotoxy(7,58+y+1+i);cout<<sequence[i+y+2]<<endl;}

             k=1; n=0; w=0; z=0;
             for (i=0; i<2*(y+1); i++) {
          gotoxy(0+w,87+n); cout<<sequence[i]<<endl;

                if (sequence[i]!=0) {w=w+13;}
          else                  {w=w+5;}
          if (i==k*y+z) {k++; n++; w=0; z++;}
          }
          delete [] sequence; sequence=0;
          }
             if (press5==1)
             {
              sequence2 = new float [5];
              if (sequence2==NULL) {gotoxy(19,50);
              cout<<"NOT ENOUGHT MEMORY FOR FA SEQUENCE ARRAY!!!!      "<<endl;}
              else                 {for (i=0; i<5; i++) sequence2[i] = 0;}

//-------------------------------------------------------------------------------------//

              for (i=0; i<2; i++)       {sequence2[i]=sequence[i];}
              sequence2[3]=sequence[y+2];

          for (i=2; i<y+2; i++)       {sequence2[2]=sequence2[2]+sequence[i];}
              for (i=y+3; i<2*(y+1); i++) {sequence2[4]=sequence2[4]+sequence[i];}

              if (press4==1) {
                    if (sequence2[0]<0) {sequence2[0]=sequence2[0]*(-1);}
                          if (sequence2[1]<0) {sequence2[1]=sequence2[1]*(-1);}
                        if (sequence2[2]<0) {sequence2[2]=sequence2[2]*(-1);}
                          if (sequence2[3]<0) {sequence2[3]=sequence2[3]*(-1);}
                    if (sequence2[4]<0) {sequence2[4]=sequence2[4]*(-1);}

                  gotoxy(0,57); cout<<"Pf:"<<endl;
                          gotoxy(3,57);cout<<sequence2[0]<<endl;
                  gotoxy(0,58); cout<<"Pf.10:"<<endl;
                          gotoxy(6,58);cout<<sequence2[1]<<endl;
                          gotoxy(0,59); cout<<"Pf.110:"<<endl;
                          gotoxy(7,59);cout<<sequence2[2]<<endl;

                  gotoxy(0,60); cout<<"Pf.101:"<<endl;
                          gotoxy(7,60);cout<<sequence2[3]<<endl;

                  gotoxy(0,61); cout<<"Pf.1101:"<<endl;
                          gotoxy(8,61);cout<<sequence2[4]<<endl;


      /*if (sequence[4]<=0.0001)      {sequence[4]=0.12*sequence[4];}
      else if (sequence[4]<=0.001)  {sequence[4]=0.27*sequence[4];}
      else if (sequence[4]<=0.01)   {sequence[4]=0.31*sequence[4];}
          else if (sequence[4]<=0.1)    {sequence[4]=0.09*sequence[4];}
          else if (sequence[4]<=0.2)    {sequence[4]=0.05*sequence[4];}
      else if (sequence[4]<=0.3)    {sequence[4]=0.04*sequence[4];}
      else if (sequence[4]<=0.4)    {sequence[4]=0.03*sequence[4];}
      else if (sequence[4]<=0.5)    {sequence[4]=0.02*sequence[4];}
      else if (sequence[4]<=0.6)    {sequence[4]=0.02*sequence[4];}
      else if (sequence[4]<=0.7)    {sequence[4]=0.01*sequence[4];}
          else if (sequence[4]<=0.8)    {sequence[4]=0.02*sequence[4];}
          else if (sequence[4]<=0.9)    {sequence[4]=0.01*sequence[4];}
          else if (sequence[4]<=1)      {sequence[4]=0.16*sequence[4];}
           else                     {sequence[4]=0*sequence[4];}


      if (sequence[3]<=0.0001)      {sequence[3]=0.12*sequence[3];}
          else if (sequence[3]<=0.001)  {sequence[3]=0.27*sequence[3];}
      else if (sequence[3]<=0.01)   {sequence[3]=0.31*sequence[3];}
      else if (sequence[3]<=0.1)    {sequence[3]=0.09*sequence[3];}
          else if (sequence[3]<=0.2)    {sequence[3]=0.05*sequence[3];}
          else if (sequence[3]<=0.3)    {sequence[3]=0.04*sequence[3];}
```

```
    else if (sequence[3]<=0.4)     {sequence[3]=0.03*sequence[3];}
    else if (sequence[3]<=0.5)     {sequence[3]=0.02*sequence[3];}
else if (sequence[3]<=0.6)     {sequence[3]=0.02*sequence[3];}
else if (sequence[3]<=0.7)     {sequence[3]=0.01*sequence[3];}
else if (sequence[3]<=0.8)     {sequence[3]=0.02*sequence[3];}
else if (sequence[3]<=0.9)     {sequence[3]=0.01*sequence[3];}
else if (sequence[3]<=1)       {sequence[3]=0.16*sequence[3];}
     else                      {sequence[3]=0*sequence[3];}

if (sequence[2]<=0.0001)       {sequence[2]=0.12*sequence[2];}
else if (sequence[2]<=0.001)   {sequence[2]=0.27*sequence[2];}
    else if (sequence[2]<=0.01)    {sequence[2]=0.31*sequence[2];}
    else if (sequence[2]<=0.1)     {sequence[2]=0.09*sequence[2];}
    else if (sequence[2]<=0.2)     {sequence[2]=0.05*sequence[2];}
    else if (sequence[2]<=0.3)     {sequence[2]=0.04*sequence[2];}
    else if (sequence[2]<=0.4)     {sequence[2]=0.03*sequence[2];}
    else if (sequence[2]<=0.5)     {sequence[2]=0.02*sequence[2];}
    else if (sequence[2]<=0.6)     {sequence[2]=0.02*sequence[2];}
    else if (sequence[2]<=0.7)     {sequence[2]=0.01*sequence[2];}
    else if (sequence[2]<=0.8)     {sequence[2]=0.02*sequence[2];}
else if (sequence[2]<=0.9)     {sequence[2]=0.01*sequence[2];}
else if (sequence[2]<=1)       {sequence[2]=0.16*sequence[2];}
     else                      {sequence[2]=0*sequence[2];}

if (sequence[1]<=0.0001)       {sequence[1]=0.12*sequence[1];}
else if (sequence[1]<=0.001)   {sequence[1]=0.27*sequence[1];}
else if (sequence[1]<=0.01)    {sequence[1]=0.31*sequence[1];}
else if (sequence[1]<=0.1)     {sequence[1]=0.09*sequence[1];}
else if (sequence[1]<=0.2)     {sequence[1]=0.05*sequence[1];}
else if (sequence[1]<=0.3)     {sequence[1]=0.04*sequence[1];}
else if (sequence[1]<=0.4)     {sequence[1]=0.03*sequence[1];}
    else if (sequence[1]<=0.5)     {sequence[1]=0.02*sequence[1];}
    else if (sequence[1]<=0.6)     {sequence[1]=0.02*sequence[1];}
else if (sequence[1]<=0.7)     {sequence[1]=0.01*sequence[1];}
else if (sequence[1]<=0.8)     {sequence[1]=0.02*sequence[1];}
else if (sequence[1]<=0.9)     {sequence[1]=0.01*sequence[1];}
else if (sequence[1]<=1)       {sequence[1]=0.16*sequence[1];}
     else                      {sequence[1]=0*sequence[1];}

if (sequence[0]<=0.0001)       {sequence[0]=0.12*sequence[0];}
else if (sequence[0]<=0.001)   {sequence[0]=0.27*sequence[0];}
else if (sequence[0]<=0.01)    {sequence[0]=0.31*sequence[0];}
else if (sequence[0]<=0.1)     {sequence[0]=0.09*sequence[0];}
else if (sequence[0]<=0.2)     {sequence[0]=0.05*sequence[0];}
else if (sequence[0]<=0.3)     {sequence[0]=0.04*sequence[0];}
else if (sequence[0]<=0.4)     {sequence[0]=0.03*sequence[0];}
else if (sequence[0]<=0.5)     {sequence[0]=0.02*sequence[0];}
else if (sequence[0]<=0.6)     {sequence[0]=0.02*sequence[0];}
else if (sequence[0]<=0.7)     {sequence[0]=0.01*sequence[0];}
    else if (sequence[0]<=0.8)     {sequence[0]=0.02*sequence[0];}
    else if (sequence[0]<=0.9)     {sequence[0]=0.01*sequence[0];}
    else if (sequence[0]<=1)       {sequence[0]=0.16*sequence[0];}
     else                      {sequence[0]=0*sequence[0];}

   //for (i=0; i<5; i++) {gotoxy(0,87+i); cout<<sequence2[i]<<endl;}


for (i=0; i<24; i++) {standard fa[i]=(standard fa[i]*sequence2[0]);}
for (i=0; i<24; i++) {pf 10[i]=(pf 10[i]*sequence2[1]);}
for (i=0; i<24; i++) {pf 110[i]=(pf 110[i]*sequence2[2]);}
for (i=0; i<24; i++) {pf_101[i]=(pf_101[i]*sequence2[3]);}

for (i=0; i<24; i++) {pf_1101[i]=(pf_1101[i]*sequence2[4]);}

for (i=0; i<24; i++) {standard fa[i] = standard fa[i] -
    off standard fa [i] + pf 10[i] - off pf 10 [i] + pf 110[i] -
    off_pf_110 [i] + pf_101[i] - off_pf_101 [i] + pf_1101[i] - off_pf_1101[i];}

for (i=0; i<24; i++) {
             if (standard fa[i]<0) {standard fa[i]=
                          standard fa[i]*(-1);}
             gotoxy(0,67+i); cout<<"\n"<<standard_fa[i]<<endl;
            }
for (i=0; i<24; i++) {
             if (standard fa[i]<0) {minimum=0;}
             else                  {minimum=standard_fa[i];}
             pulse energy = minimum * photon energy;
             energy in frame = pulse energy * mark space correction;
                        energy per pcm bit = (energy in frame/pcm bits);
                        dBm = 10 * (log((energy_per_pcm_bit*B)/pow(10,-3))
                        /log(10));
             gotoxy(0,95+i); cout<<"\n"<<dBm<<endl;
```

```
                                                    }*/
                                }

                     delete [] sequence; sequence=0;
                     delete [] sequence2; sequence2=0;
                    }
//-------------------------------------------------------------------------------------------//


   gotoxy(19,50);cout<<"DISPLAY INPUT ARRAYS? PRESS1,3 ELSE 8   "<<endl;
   gotoxy(64,47);cin>>press3;

   while ( (press3!=1)||(press3!=3)||(press3!=8)||(press3!=-1) )
         {
   if (press3==1) {
   clrscr2(); gotoxy(0,56); cout<<"FALSE ALARM MPPM-DATA ARRAY            "<<endl;
   print input arrays(mppm data,pow(2,pcm bits),x+pcm bits,x,y,aut pasc,aut pasc2,
   aut pasc3,aut pcm bits,pcm bits,pasc,pasc2,pasc3,start_number,end_number,press1,
   press2,press3,press4,press5);

   press6=0;
   do
   {
   gotoxy(19,50);
   cout<<"DO YOU WANT TO SAVE THE DATA IN A FILE?               "<<endl;
   gotoxy(33,47);cin>>press6;
   if (press6==-1) {break;}
   if (press6==2)  {break;}
   if (press6==3)  {break;}
   }while(press6!=1);

   if (press6==3) {file_op1.close(); break;}
   if (press6==1) {
   k=1;
   for (i=0; i<(x+pcm_bits)*pow(2,pcm_bits); i++)
   {
   if (i==k*(x+pcm bits)) {file op1<<"\n"; k++;}
   file_op1<<mppm_data[i]<<"  ";
   }
   file op1.close();
                                        }


   break;
   }
   else if (press3==3) {
   clrscr2(); gotoxy(0,56); cout<<"WEIGTHED FALSE ALARM ARRAY             "<<endl;

//-------------------------------------------------------------------------------------------//

   print input arrays(weighted fa mppm data,pasc3,x+pcm bits,x,y,aut pasc,aut pasc2,
   aut pasc3,aut pcm bits,pcm bits,pasc,pasc2,pasc3,start_number,end_number,press1,
   press2,press3,press4,press5);
   press6=0;
   do
   {
   gotoxy(19,50);
   cout<<"DO YOU WANT TO SAVE THE DATA IN A FILE?               "<<endl;
   gotoxy(33,47);cin>>press6;
   if (press6==-1) {break;}
   if (press6==2)  {break;}
   if (press6==3)  {break;}
    }while(press6!=1);

   if (press6==3) {file_op1.close(); break;}
   if (press6==1) {
   k=1;
   for (i=0; i<(x+pcm_bits)*pasc3; i++)
   {
   if (i==k*(x+pcm bits)) {file op1<<"\n"; k++;}
   file_op1<<weighted_fa_mppm_data[i]<<"  ";
   }
   file op1.close();
   }


   break;
   }
   else if ( (press3==8)||(press3==-1) ) {break;}
   else                                  {gotoxy(19,50);
```

65

```
    cout<<"WRONG SELECTION.TRY AGAIN                        "<<endl;
    gotoxy(64,47);cin>>press3;}
    }

    if (press3==-1) {break;}
    if (press6==-1) {break;}

    clrscr2();

            delete [] mppm data; mppm data=0;
            delete [] weighted_fa_mppm_data; weighted_fa_mppm_data=0;

            delete [] mppm; mppm=0;
            delete [] data; data=0;


          }//end of if-else mppm_fa null statement

  }//end of if-press1(2) and (y!=1) and (y!=(x-1))

//-------------------------------------------------------------------------------------------//


  else if (press1==3)
  { mppm = new int [x*pasc2]; data = new int [pcm bits*pasc2];
    weighted_er_mppm_data = new int [(x+pcm_bits)*pasc2];

    if (mppm==NULL) {gotoxy(19,50);
    cout<<"NOT ENOUGHT MEMORY FOR MPPM2 ARRAY!!!!    "<<endl;}
    else { gotoxy(56,11); cout<<erasure weight<<endl;
           gotoxy(60,12); cout<<false_alarm_weight<<endl;

           for (i=0; i<x*pasc2; i++) mppm[i] = 0;
           for (i=0; i<y-erasure_bits; i++) mppm[i] = 1;

           i=0; j=0; l=0; m=0; extra_pulses=0; k=1;

           for (m=0; m<2*pasc2; m++)
           { for (i=0; i<x-1; i++) if ( mppm[i+(k-1)*x]==1 ) j=i;

             for (i=1; i<=y-erasure bits; i++)
             { if (mppm[k*x-i]==0) break;
               else if (mppm[k*x-i]==1) extra pulses++;
             } if (extra_pulses==y-erasure_bits) break;

             if ( j!=(x-1) )
             { for (i=0; i<j; i++) mppm[i+k*x]=mppm[i+(k-1)*x];
               mppm[j+k*x+1]=1;

               for (i=1; i<=extra pulses; i++)
               mppm[j+k*x+1+i]=1;

               l=0; extra pulses=0;
             }
             else if ( (j==(x-1) ) && (extra pulses!=0) )
            {l=extra_pulses; k--; extra_pulses=0;}
             k++;
           }

         }//end of if-else mppm null statement

    erasure_weight2=0;

//-------------------------------------------------------------------------------------------//


    if (data==NULL) {gotoxy(19,50);
    cout<<"NOT ENOUGHT MEMORY FOR DATA2 ARRAY!!!!    "<<endl;}
    else { for (i=0; i<pcm bits*pasc2; i++) data[i] = 0;
           w=0; row counter=0; bit counter=0; k=1;
           for (l=1; l<=pasc2; l++)
           { for (j=x; j<(x+pcm bits); j++)
             { for (m=0; m<pow(2,pcm bits); m++)
               { for (i=0; i<x; i++) {
           if ( mppm[i+(l-1)*x]==mppm_data[i+(k-1)*(pcm_bits+x)] ) row_counter++; }

                    if ( ( row counter==(x-erasure bits) ) &&
            ( mppm data[j+(k-1)*(pcm bits+x)]==1 ) ) bit counter++;
       if (row counter==x-erasure bits) erasure_weight2++;
                    row_counter=0; k++;
             } k=1;
               if ( bit_counter>(erasure_weight2/2.0) )
```

```
                data[w+(l-1)*pcm bits]=1;
                else if ( bit counter<(erasure_weight2/2.0) )
                data[w+(l-1)*pcm bits]=0;
                else if ( bit counter=(erasure_weight2/2.0) )
                data[w+(l-1)*pcm bits]=2;
                bit counter=0; w++; erasure_weight2=0;
              } k=1; w=0;
            }

          }//end of if-else data null statement

      if (weighted er mppm data==NULL) {
      gotoxy(19,50);cout<<"NOT ENOUGHT MEMORY FOR WEIGHT-ERASURE    "<<endl;}
      else { for (i=0; i<(x+pcm_bits)*pasc2; i++) weighted_er_mppm_data[i] = 0;

             k=0;
             for (m=0; m<pasc2; m++)
             {
               for (i=0; i<x; i++)
               { weighted_er_mppm_data[k*(x+pcm_bits)+i] = mppm[k*x+i]; }
               k++; l++;
             }

             k=0; l=0;
             for (m=0; m<pasc2; m++)
             {
               for (i=x; i<x+pcm bits; i++)
               { weighted er mppm data[k*(x+pcm_bits)+i] =
           data[(l*pcm bits)-x+i]; }
               k++; l++;
             }

          }//end of if-else weighted_er_mppm_data null statement

//-------------------------------------------------------------------------------------------//


      mppm = new int [x*pasc3]; data = new int [pcm bits*pasc3];
      weighted_fa_mppm_data = new int [(x+pcm_bits)*pasc3];

      if (mppm==NULL) {gotoxy(19,50);
      cout<<"NOT ENOUGHT MEMORY FOR MPPM3 ARRAY!!!!        "<<endl;}
      else { for (i=0; i<x*pasc3; i++) mppm[i] = 0;
             for (i=0; i<y+false_alarm_bits; i++) mppm[i] = 1;

             i=0; j=0; l=0; m=0; extra pulses=0; k=1;
             for (m=0; m<2*pasc3; m++)
             { for (i=0; i<x-1; i++) if ( mppm[i+(k-1)*x]==1 ) j=i;

               for (i=1; i<=y+false alarm bits; i++)
               { if (mppm[k*x-i]==0) break;
                 else if (mppm[k*x-i]==1) extra pulses++;
               } if (extra_pulses==y+false_alarm_bits) break;

               if ( j!=(x-1) )
               { for (i=0; i<j; i++) mppm[i+k*x]=mppm[i+(k-1)*x];
                  mppm[j+k*x+1]=1;

                 for (i=1; i<=extra pulses; i++)
                  mppm[j+k*x+1+i]=1;

                 l=0; extra pulses=0;
               }
               else if ( (j==(x-1) ) && (extra pulses!=0) )
              {l=extra_pulses; k--; extra_pulses=0;}

               k++;
             }

          }//end of if-else mppm null statement

      false alarm weight2=0;
      if (data==NULL) {gotoxy(19,50);
      cout<<"NOT ENOUGHT MEMORY FOR DATA3 ARRAY!!!!      "<<endl;}
      else { for (i=0; i<pcm bits*pasc3; i++) data[i] = 0;
             w=0; row counter=0; bit counter=0; k=1;
             for (l=1; l<=pasc3; l++)
             { for (j=x; j<(x+pcm bits); j++)
               { for (m=0; m<pow(2,pcm bits); m++)
                 { for (i=0; i<x; i++) {
              if ( mppm[i+(l-1)*x]==mppm_data[i+(k-1)*(pcm_bits+x)] ) row_counter++; }
```

```
                if ( ( row counter==(x-false alarm bits) ) && ( mppm_data[j+(k-1)*
                (pcm bits+x)]==1 ) ) bit counter++;
                if ( row counter==(x-false_alarm_bits) ) false_alarm_weight2++;
                     row_counter=0; k++;
              } k=1;
              if      ( bit counter > (false_alarm_weight2/2.0) )
              data[w+(l-1)*pcm bits]=1;
              else if ( bit counter < (false_alarm_weight2/2.0) )
              data[W+(l-1)*pcm bits]=0;
              else if ( bit counter = (false_alarm_weight2/2.0) )
              data[w+(l-1)*pcm bits]=2;
              bit counter=0; w++; false_alarm_weight2=0;
            } k=1; w=0;
          }

        }//end of if-else data null statement

//-------------------------------------------------------------------------------------//

    if (weighted fa mppm data==NULL) {gotoxy(19,50);
    cout<<"NOT ENOUGHT MEMORY FOR WEIGHT-FA ALARM     "<<endl;}
    else { for (i=0; i<(x+pcm_bits)*pasc3; i++) weighted_fa_mppm_data[i] = 0;

           k=0;
           for (m=0; m<pasc3; m++)
           { for (i=0; i<x; i++)
             { weighted_fa_mppm_data[k*(x+pcm_bits)+i] = mppm[k*x+i]; }
             k++; l++;
           }

           k=0; l=0;
           for (m=0; m<pasc3; m++)
           { for (i=x; i<x+pcm bits; i++)
             { weighted fa mppm data[k*(x+pcm_bits)+i] =
           data[(l*pcm_bits)-x+i]; }
             k++; l++;
           }

         }//end of if-else weighted_fa_mppm_data null statement

    mppm_ws = new int [(x+pcm_bits+2*y)*pow(2,pcm_bits)];

//-------------------------------------------------------------------------------------//

    if (mppm ws == NULL) {gotoxy(19,50);
    cout<<"NOT ENOUGHT MEMORY FOR MPPM-WS ARRAY           "<<endl;}
    else { for (i=0; i<(x+pcm_bits+2*y)*pow(2,pcm_bits); i++) mppm_ws[i] = 0;

           flag_ws=1;

           k=0;
           for (m=0; m<pow(2,pcm bits); m++)
           { for (i=0; i<x; i++) mppm ws[k*(x+pcm bits+2*y)+i]=
           mppm data[k*(x+pcm_bits)+i];
                 k++;
           }

           a=0;
           if (y%2==0) a=y/2;
           else        a=(y/2)+1;

           k=0; l=0; n=0; row counter=0; n1=0; counter=0; flag=0; o=0; w=0;
           for (m=0; m<pow(2,pcm bits); m++)
           { for (p=0; p<a; p++)
             { for (j=flag; j<x; j++)
               { if ( (mppm data[k*(x+pcm bits)+j]==1) && (counter!=0) )
                 {n1=j; flag=n1+1; break;}
                  else if ( (mppm data[k*(x+pcm bits)+j]==1) && (counter==0) )
                  {n=j; counter++;}
               }

               if (p==0) {o=0;}

//-------------------------------------------------------------------------------------//

               if ( (n!=0) && (n!=flag) ) { mppm_data[k*(x+pcm_bits)+n-1]=1;
               for (z=0; z<pasc3; z++)
               { for (j=0; j<x; j++) if (mppm data[k*(x+pcm bits)+j]==
               weighted_fa_mppm_data[l*(x+pcm_bits)+j]) row_counter++;
```

```
                if (row counter==x) { for (j=x; j<x+pcm_bits; j++)
                { mppm ws[j+k*(x+pcm bits)+w*2*y]=
                weighted fa mppm data[l*(x+pcm bits)+j]^mppm_data[k*(x+pcm_bits)+j];

                if (mppm ws[j+k*(x+pcm bits)+w*2*y]!=0)
                {mppm ws[(k+1)*(x+pcm bits)+w*2*y+o]++;}
                mppm ws[j+k*(x+pcm bits)+w*2*y]=0;
                } row_counter=0; l=0; mppm_data[k*(x+pcm_bits)+n-1]=0; break;
               }
                else l++; row counter=0;
                                        }
                                      } l=0; o+=2;

              if ( ( n1!=(n+1) ) && (n1!=0) ) { mppm_data[k*(x+pcm_bits)+n1-1]=1;
              for (z=0; z<pasc3; z++)
              { for (j=0; j<x; j++) if (mppm data[k*(x+pcm bits)+j]==
              weighted fa mppm data[l*(x+pcm bits)+j]) row counter++;
           if (row counter==x) { for (j=x; j<x+pcm bits; j++)
              { mppm ws[j+k*(x+pcm bits)+w*2*y]=weighted fa mppm data[l*(x+pcm bits)+j]
              ^mppm data[k*(x+pcm_bits)+j];

              if (mppm ws[j+k*(x+pcm bits)+w*2*y]!=0)
              {mppm ws[(k+1)*(x+pcm bits)+w*2*y+o]++;}
           mppm ws[j+k*(x+pcm bits)+w*2*y]=0;
          } row_counter=0; l=0; mppm_data[k*(x+pcm_bits)+n1-1]=0; break;
          }
          else l++; row_counter=0;
          }
                  } n=0; n1=0; counter=0; row_counter=0; l=0; o+=2;

              } k++; w++; flag=0; n=0; n1=0; row_counter=0; l=0; counter=0;
            }
//-------------------------------------------------------------------------------------//


            k=0; l=0; n=0; row_counter=0; n1=0; counter=0; flag=0; bit_counter=0;
            l1=1; o=1; w=0;
            for (m=0; m<pow(2,pcm_bits); m++)
            { for (p=0; p<a; p++)
              { for (j=flag; j<x; j++)
                { if ( (mppm data[k*(x+pcm_bits)+j]==1)&&(counter!=0) )
                {n1=j; flag=n1+1; break;}
                  else if ( (mppm data[k*(x+pcm_bits)+j]==1)&&(counter==0) )
               {n=j; counter++;}
                }

                if (p==0) {o=1;}

                if (n!=-1) { if (n!=(x-1)) {mppm_data[k*(x+pcm_bits)+n+1]=1;
                mppm data[k*(x+pcm bits)+n]=0;}
                           else           mppm_data[k*(x+pcm_bits)+n]=0;

                for (j=0; j<x; j++)  if (mppm_data[k*(x+pcm_bits)+j]==1)
                bit_counter++;

             if (bit counter==y) { for (z=0; z<pow(2,pcm bits); z++)
            { for (j=0; j<x; j++) if ( mppm data[k*(x+pcm bits)+j]==
                 mppm data[l1*(x+pcm bits)+j]) row counter++;
               if (row counter==x) { for (j=x; j<x+pcm bits; j++)
               { mppm ws[j+k*(x+pcm bits)+w*2*y]=mppm data[l1*(x+pcm bits)+j]
               ^mppm data[k*(x+pcm_bits)+j];

                if (mppm ws[j+k*(x+pcm bits)+w*2*y]!=0)
                {mppm ws[(k+1)*(x+pcm bits)+w*2*y+o]++;}
                mppm ws[j+k*(x+pcm bits)+w*2*y]=0;
                } row counter=0; l1=k+1; bit counter=0; if (n!=(x-1))
               {mppm data[k*(x+pcm bits)+n+1]=0; mppm data[k*(x+pcm_bits)+n]=1;}
               else {mppm_data[k*(x+pcm_bits)+n]=1;} break;
                     }
               else l1++; row_counter=0;
                  }
                        }
//-------------------------------------------------------------------------------------//

              else if ( bit counter==(y-1) ) { for (z=0; z<pasc2; z++)
  { for (j=0; j<x; j++) if ( mppm data[k*(x+pcm bits)+j]==
  weighted er mppm data[l*(x+pcm bits)+j]) row counter++;
  if (row counter==x) { for (j=x; j<x+pcm bits; j++)
  { mppm ws[j+k*(x+pcm bits)+w*2*y]=weighted er mppm data[l*(x+pcm bits)+j]^
   mppm_data[k*(x+pcm_bits)+j];
```

```
  if (mppm ws[j+k*(x+pcm bits)+w*2*y]!=0) {mppm_ws[(k+1)*(x+pcm_bits)+w*2*y+o]++;}
          mppm ws[j+k*(x+pcm bits)+w*2*y]=0;
  } row_counter=0; l=0; mppm_data[k*(x+pcm_bits)+n]=1; bit_counter=0; break;
  }
  else l++; row counter=0;
                                          }
                                                              }
                         } o+=2;

  if (n1!=0) { if ( n1!=(x-1) ) {mppm_data[k*(x+pcm_bits)+n1+1]=1;
  mppm data[k*(x+pcm bits)+n1]=0;}
  else            mppm_data[k*(x+pcm_bits)+n1]=0;

 for (j=0; j<x; j++) if (mppm_data[k*(x+pcm_bits)+j]==1) bit_counter++;

  if (bit counter==y) { for (z=0; z<pow(2,pcm bits); z++)
  { for (j=0; j<x; j++) if ( mppm data[k*(x+pcm_bits)+j]==
  mppm data[l1*(x+pcm bits)+j]) row counter++;
  if (row counter==x) { for (j=x; j<x+pcm bits; j++)
  { mppm ws[j+k*(x+pcm bits)+w*2*y]=mppm data[l1*(x+pcm bits)+j]^
  mppm data[k*(x+pcm_bits)+j];

  if (mppm ws[j+k*(x+pcm bits)+w*2*y]!=0) {mppm_ws[(k+1)*(x+pcm_bits)+w*2*y+o]++;}
  mppm ws[j+k*(x+pcm bits)+w*2*y]=0;
  } row counter=0; l1=k+1; bit counter=0; if (n1!=(x-1)) {
  mppm data[k*(x+pcm bits)+n1+1]=0;
  mppm data[k*(x+pcm bits)+n1]=1;}
  else {mppm_data[k*(x+pcm_bits)+n1]=1;} break;
  }
 else l1++; row_counter=0;
  }
   }

//-------------------------------------------------------------------------------------------//

else if ( bit counter==(y-1) ) { for (z=0; z<pasc2; z++)
{ for (j=0; j<x; j++) if ( mppm data[k*(x+pcm bits)+j]==
 weighted er mppm data[l*(x+pcm bits)+j]) row counter++;
if (row counter==x) { for (j=x; j<x+pcm bits; j++)
{ mppm ws[j+k*(x+pcm bits)+w*2*y]=weighted er mppm data[l*(x+pcm bits)+j]^
 mppm data[k*(x+pcm bits)+j];

if (mppm ws[j+k*(x+pcm bits)+w*2*y]!=0) {mppm ws[(k+1)*(x+pcm bits)+w*2*y+o]++;}
                                                                                  mppm_ws
[j+k*(x+pcm bits)+w*2*y]=0;
} row counter=0; l=0; mppm data[k*(x+pcm_bits)+n1]=1; bit_counter=0; break;
                                        }
 else l++; row_counter=0;
}
                                                              }

                         } n=0; n1=0; counter=0; row_counter=0; l=0; bit_counter=0;
                         l1=k+1; o+=2;

            } k++; w++; counter=0; n=0; n1=0; flag=0; o+=2;
          row counter=0; l=0;
                  bit_counter=0; l1=k+1;
          }

          press5=0;
          gotoxy(19,50); cout<<"USE 2-PULSE ALGORITHM OR THE DETAILED?  "<<endl;
          gotoxy(10,39);cin>>press5;

          while ( (press5!=1)||(press5!=2)||(press5!=-1) )
      { if (press5==-1)                          {break;}
      else if ( (press5==1)||(press5==2) ) {break;}
      else                                       {gotoxy(19,50);
          cout<<"WRONG SELECTION.TRY AGAIN                "<<endl;
          gotoxy(10,39);cout<<"  "<<endl;
      gotoxy(10,39);cin>>press5;}
          }

//-------------------------------------------------------------------------------------------//


          if (press5==-1) {break;}

         press4=0;
          gotoxy(19,50);
         cout<<"DO YOU WANT TO DISPLAY THE SEQUENCE RESULTS?      "<<endl;
         gotoxy(10,44);cin>>press4;
```

```
            while ( (press4!=1)||(press4!=2)||(press4!=-1) )
            { if (press4==-1)                              {break;}
              else if ( (press4==1)||(press4==2) ) {break;}
              else
                {gotoxy(19,50);
                cout<<"WRONG SELECTION.TRY AGAIN                "<<endl;
                gotoxy(10,44);cout<<"          "<<endl;
                gotoxy(10,44);cin>>press4;}
            }

            if (press4==-1) {break;}

            l1=0; zero counter=0;
            for ( m=0; m<pow(2,pcm_bits); m++)
            {
             for ( i=0; i<2; i++) {
           if (mppm ws[i+m*(x+pcm bits+2*y)]==0) {zero_counter++;}
                                  }
             if (zero counter==2) {break;}
             else                 {zero_counter=0; l1++;}
            }

            bit counter=0; row counter=0; i=0;
            for ( m=0; m<pow(2,pcm bits); m++)
            {  if (mppm ws[i+m*(x+pcm bits+2*y)]==1)
               {bit counter++;}//1-bit_counter=11//
               i=x-1;
               if (mppm ws[i+m*(x+pcm_bits+2*y)]==1) {row_counter++;}
               //1-row_counter=9//
               i=0;
            }
//-------------------------------------------------------------------------------------//

             sequence = new float [y*(y+2)+1];
             if (sequence==NULL) {gotoxy(19,50);
             cout<<"NOT ENOUGHT MEMORY FOR WS SEQUENCE ARRAY!!!!      "<<endl;}
             else                {for (i=0; i<y*(y+2)+1; i++) sequence[i] = 0;}

             zero counter=0; pulse counter=0;
             pulse_counter2=0; l=0; j=0; flag=1; limit=0;

             for (w=0; w<(y-1); w++)
             {
              for (n=0; n<pow(2,pcm_bits); n++)
              {
               for (i=0; i<x; i++)
               {
            if ( (mppm ws[i+l*(x+pcm bits+2*y)]==1)&&(i==0)&&(w==0) )        {
            zero counter=0; pulse counter=1; pulse_counter2=0;

            for (m=i+1; m<x; m++) {

            if (mppm ws[m+l*(x+pcm bits+2*y)]==1) {pulse counter2++;}
            else                                  {break;}
                }

//-------------------------------------------------------------------------------------//

        if (pulse counter2==0) {
        for (z=0; z<2; z++) {
        sequence[l]=sequence[l]+(mppm ws[(l+1)*(x+pcm bits)+l*2*y+j]*
        (row counter/pow(2,pcm bits)));

        sequence[0]=sequence[0]+(mppm ws[(l+1)*(x+pcm bits)+l*2*y+j]*
        ((pow(2,pcm_bits)-row_counter)/pow(2,pcm_bits)));
        j++;
         }
          }
        else                    {
        for (z=0; z<2; z++) {
        sequence[pulse counter2*(y+1)+pulse counter]=
        sequence[pulse counter2*(y+1)]+(mppm ws[(l+1)*(x+pcm bits)+l*2*y+j]*
        (row counter/pow(2,pcm bits)));



        sequence[pulse_counter2*(y+1)]=sequence[pulse_counter2*(y+1)]+
```

```
        (mppm ws[(l+1)*(x+pcm_bits)+l*2*y+j]*((pow(2,pcm_bits)-row_counter)/
        pow(2,pcm_bits)));
        j++;
         }
          }
           }
//-------------------------------------------------------------------------------------//

           else if ( (mppm ws[i+l*(x+pcm bits+2*y)]==1)&&(i!=0)&(i!=(x-1)) ){
               zero counter=0; pulse counter=0; pulse_counter2=0;

                   for (m=i-1; m>=0; m--) {

                       if (mppm ws[m+l*(x+pcm_bits+2*y)]==1)
                     {pulse counter++;}
                           else
                     {zero counter=m; break;}
                              }

        if ( (pulse counter==0)&&(i!=1) ) {
            pulse counter=0;
                for (m=zero counter-1; m>=0; m--) {

                    if (mppm_ws[m+l*(x+pcm_bits+2*y)]==1)
        {pulse counter++;}
        else                                {break;}
            }

//-------------------------------------------------------------------------------------//


        if ( (pulse counter==0)&&(w==0) ) {
        flag=0;
        for (m=i+1; m<x; m++)  {

        if (mppm ws[m+l*(x+pcm bits+2*y)]==1) {pulse counter2++;}
//-------------------------------------------------------------------------------------//
        else                                {break;}
         }
        if (pulse counter2==0) { for (z=0; z<2; z++)
        {sequence[0]=sequence[0]+mppm ws[(l+1)*(x+pcm_bits)+l*2*y+j]; j++;} }
        else                    { for (z=0; z<2; z++)
        {sequence[pulse counter2*(y+1)]=sequence[pulse_counter2*(y+1)]+mppm_ws[(l+1)*
        (x+pcm bits)+l*2*y+j]; j++;} }
//-------------------------------------------------------------------------------------//
        }
        else                               { flag=0; for (z=0; z<2; z++) {j++;} }

        }
        else if ( (pulse counter==0)&&(i==1)&&(w==0) ) {
        for (m=i+1; m<x; m++)  {

        if (mppm ws[m+l*(x+pcm bits+2*y)]==1) {pulse counter2++;}
        else                                {break;}
        }
        if (pulse counter2==0) {
        for (z=0; z<2; z++) {


        sequence[0]=sequence[0]+(mppm ws[(l+1)*(x+pcm bits)+l*2*y+j]*
        ((pow(2,pcm_bits)-row_counter)/pow(2,pcm_bits)));
         j++;
         }
          }
        else                    {
        for (z=0; z<2; z++) {
        sequence[pulse counter2*(y+1)]=sequence[pulse counter2*(y+1)]+
        (mppm ws[(l+1)*(x+pcm_bits)+l*2*y+j]*((pow(2,pcm_bits)-row_counter)
        /pow(2,pcm_bits)));
         j++;
         }
          }
                                                                                    }
        else if (pulse counter==limit+1) {

        for (m=i+1; m<x; m++)  {

        if (mppm ws[m+l*(x+pcm bits+2*y)]==1) {pulse counter2++;}
        else                                {break;}
        }

        if (flag==0) {
```

```
        if (pulse counter2==0) {
        for (z=0; z<2; z++) {sequence[pulse counter]=sequence[pulse counter]+
        mppm ws[(l+1)*(x+pcm bits)+l*2*y+j]; j++;}
         }         else
          {
        for (z=0; z<2; z++) {sequence[pulse counter2*(y+1)+pulse counter]=
        sequence[pulse counter2*(y+1)+pulse_counter]+mppm_ws[(l+1)*
        (x+pcm_bits)+l*2*y+j];
        j++;}
         }
          }

        else         {
        if (pulse counter2==0) {
        for (z=0; z<2; z++) {
        sequence[pulse counter+1]=sequence[pulse counter+1]+
        (mppm ws[(l+1)*(x+pcm bits)+l*2*y+j]*(row counter/pow(2,pcm bits)));
        sequence[pulse counter]=sequence[pulse counter]+(mppm ws[(l+1)*
        (x+pcm_bits)+l*2*y+j]*((pow(2,pcm_bits)-row_counter)/pow(2,pcm_bits)));
        j++;
        }
        }
        else                    {
         for (z=0; z<2; z++) {
        sequence[pulse counter2*(y+1)+pulse counter+1]=
        sequence[pulse counter2*(y+1)+pulse counter+1]+(mppm ws[(l+1)*
        (x+pcm bits)+l*2*y+j]*(row counter/pow(2,pcm bits)));
            sequence[pulse counter2*(y+1)+pulse counter]=
        sequence[pulse counter2*(y+1)+pulse counter]+
        (mppm ws[(l+1)*(x+pcm bits)+l*2*y+j]*
        ((pow(2,pcm bits)-row_counter)/pow(2,pcm_bits)));
                j++;
                    }
                        }

                            }

                                }
                                    }
            else if ( (mppm ws[i+l*(x+pcm bits+2*y)]==1)&&(i==(x-1)) ) {
            zero counter=0; pulse counter=0; pulse_counter2=1;

            for (m=i-1; m>=0; m--) {

            if (mppm ws[m+l*(x+pcm bits+2*y)]==1) {pulse counter++;}
                     else        {zero counter=m; break;}
                                }
        if (pulse counter==0) {
        pulse counter=0;
            for (m=zero counter-1; m>=0; m--) {

                if (mppm ws[m+l*(x+pcm bits+2*y)]==1) {pulse counter++;}
                    else                                    {break;}
                        }
                                            if (pulse_counter==0) {
                flag=0;
                                for (z=0; z<2; z++)
                {

       sequence[pulse counter2*(y+1)]=sequence[pulse counter2*(y+1)]+
      (mppm ws[(l+1)*(x+pcm bits)+l*2*y+j]*(bit counter/pow(2,pcm bits)));
      sequence[0]=sequence[0]+(mppm ws[(l+1)*(x+pcm bits)+l*2*y+j]*
      ((pow(2,pcm bits)-bit counter)/pow(2,pcm_bits)));             j++;

       }

  }
else if (pulse counter==limit+1) {



//-------------------------------------------------------------------------------------//

                                    for (z=0; z<2; z++) {
sequence[pulse counter]=sequence[pulse_counter]+(mppm_ws[(l+1)*(x+pcm_bits)+l*2*y+j]*
 (bit counter/pow(2,pcm bits)));
sequence[pulse counter2*(y+1)+pulse counter]=
sequence[pulse counter2*(y+1)+pulse counter]+(mppm ws[(l+1)*
 (x+pcm_bits)+l*2*y+j]*((pow(2,pcm_bits)-bit_counter)/pow(2,pcm_bits)));
j++;
 }
```

```
  }
   }
    }
              if (n==pasc)                    {break;}
             }l++; j=0; if (l<=l1) {flag=1;}
                        else       {flag=0;}
              if (n==pasc)         {break;}
            }l=0; j=0; flag=1; limit++;
           }

for (i=0; i<y*(y+2)+1; i++) {sequence[i]=(sequence[i]/(pcm_bits*pow(2,pcm_bits)));}

            if ( (press4==1)&&(press5==2) ) {
           //gotoxy(0,57); cout<<"Ps:"<<endl; gotoxy(3,57);cout<<sequence[0]<<endl;

           //for (i=0; i<y; i++) { gotoxy(0,58+i);
           //cout<<"Ps.11:"<<endl; gotoxy(6,58+i);cout<<sequence[i+1]<<endl;}

           //for (i=0; i<y; i++) { gotoxy(0,58+y+i); cout<<"Ps1.1:"<<endl;
           //gotoxy(6,58+y+i);cout<<sequence[(i+1)*(y+1)]<<endl;}


           //for (i=0; i<(y-1); i++) { gotoxy(0,58+2*y+i); cout<<"Ps1.1.11:"<<endl;
           //gotoxy(9,58+2*y+i);cout<<sequence[(i+1)*(y+1)+1]<<endl;}

           //for (i=0; i<(y-2); i++) { gotoxy(0,58+3*y-1+i); cout<<"Ps11.1.1:"<<endl;
           //gotoxy(9,58+3*y-2+i);cout<<sequence[(y+1)+2+i]<<endl;}
           //if (y>4) {for (i=1; i<0.5*y; i++) { gotoxy(0,58+4*y-2+i);
           //cout<<"Ps11.1.11:"<<endl;
           //gotoxy(10,58+4*y-2+i);cout<<sequence[(i+1)*(y+2)]<<endl;} }

//-------------------------------------------------------------------------------------------//

           k=1; n=0; w=0; z=0;
           for (i=0; i<y*(y+2)+1; i++) {
           if (w>60) {n++; w=0;}
           gotoxy(0+w,77+n); cout<<sequence[i]<<endl;

            w=w+12;
           if (i==k*y+z) {k++; n++; w=0; z++;}

           }
           delete [] sequence; sequence=0;
           }

           else if (press5==1)
           {
            sequence2 = new float [9];
            if (sequence2==NULL) {gotoxy(19,50);
            cout<<"NOT ENOUGHT MEMORY FOR WS SEQUENCE ARRAY!!!!      "<<endl;}
            else                 {for (i=0; i<9; i++) sequence2[i] = 0;}


            for (i=0; i<2; i++) {sequence2[i]=sequence[i]; sequence[i]=0;}
            sequence2[3]=sequence[y+1]; sequence[y+1]=0;
            sequence2[5]=sequence[y+2]; sequence[y+2]=0;

//-------------------------------------------------------------------------------------------//


                for (i=2; i<y+1; i++)   {sequence2[2]=sequence2[2]+sequence[i];
                    sequence[i]=0;}
            for (i=y+3; i<2*(y+1); i++) {sequence2[6]=sequence2[6]+
                    sequence[i]; sequence[i]=0;}

            for (i=2; i<y; i++)
                    {sequence2[4]=sequence2[4]+sequence[i*(y+1)];
                    sequence[i*(y+1)]=0;}
            for (i=2; i<y; i++)
                    {sequence2[7]=sequence2[7]+sequence[i*(y+1)+1];
                     sequence[i*(y+1)+1]=0;}
            for (i=0; i<y*(y+2)+1; i++)
                    {sequence2[8]=sequence2[8]+sequence[i];          sequence[i]=0;}

//-------------------------------------------------------------------------------------------//

            if (press4==1) {
                           if (sequence[0]<0) {sequence2[0]=sequence2[0]*(-1);}
                           if (sequence[1]<0) {sequence2[1]=sequence2[1]*(-1);}
                           if (sequence[2]<0) {sequence2[2]=sequence2[2]*(-1);}
                           if (sequence[3]<0) {sequence2[3]=sequence2[3]*(-1);}
```

```
                        if (sequence[4]<0) {sequence2[4]=
                        sequence2[4]*(-1);}
                        if (sequence[5]<0) {sequence2[5]=sequence2[5]*(-1);}
                        if (sequence[6]<0) {sequence2[6]=sequence2[6]*(-1);}
                        if (sequence[7]<0) {sequence2[7]=sequence2[7]*(-1);}
                        if (sequence[8]<0) {sequence2[8]=sequence2[8]*(-1);}

          gotoxy(0,57); cout<<"Ps:"<<endl;
                        gotoxy(3,57);cout<<sequence2[0]<<endl;
          gotoxy(0,58); cout<<"Ps.11:"<<endl;
                        gotoxy(6,58);cout<<sequence2[1]<<endl;
          gotoxy(0,59); cout<<"Ps.111:"<<endl;
                        gotoxy(7,59);cout<<sequence2[2]<<endl;
          gotoxy(0,60); cout<<"Ps1.1:"<<endl;
                        gotoxy(6,60);cout<<sequence2[3]<<endl;
          gotoxy(0,61); cout<<"Ps1.11:"<<endl;
                        gotoxy(7,61);cout<<sequence2[4]<<endl;
       gotoxy(0,62); cout<<"Ps1.1.1:"<<endl;
              gotoxy(8,62);cout<<sequence2[5]<<endl;
       gotoxy(0,63); cout<<"Ps11.1.1:"<<endl;
              gotoxy(9,63);cout<<sequence2[6]<<endl;
       gotoxy(0,64); cout<<"Ps1.1.11:"<<endl;
              gotoxy(9,64);cout<<sequence2[7]<<endl;
       gotoxy(0,65); cout<<"Ps11.1.11:"<<endl;
              gotoxy(10,65);cout<<sequence2[8]<<endl;

//-------------------------------------------------------------------------------------------//

      /*if (sequence[8]<=0.0001)       {sequence[8]=0.12*sequence[8];}
      else if (sequence[8]<=0.001)   {sequence[8]=0.27*sequence[8];}
      else if (sequence[8]<=0.01)    {sequence[8]=0.31*sequence[8];}
      else if (sequence[8]<=0.1)     {sequence[8]=0.09*sequence[8];}
      else if (sequence[8]<=0.2)     {sequence[8]=0.05*sequence[8];}
      else if (sequence[8]<=0.3)     {sequence[8]=0.04*sequence[8];}
      else if (sequence[8]<=0.4)     {sequence[8]=0.03*sequence[8];}
      else if (sequence[8]<=0.5)     {sequence[8]=0.02*sequence[8];}
      else if (sequence[8]<=0.6)     {sequence[8]=0.02*sequence[8];}
      else if (sequence[8]<=0.7)     {sequence[8]=0.01*sequence[8];}
      else if (sequence[8]<=0.8)     {sequence[8]=0.02*sequence[8];}
      else if (sequence[8]<=0.9)     {sequence[8]=0.01*sequence[8];}
      else if (sequence[8]<=1)       {sequence[8]=0.16*sequence[8];}
       else                       {sequence[8]=0*sequence[8];}

      if (sequence[7]<=0.0001)       {sequence[7]=0.12*sequence[7];}
      else if (sequence[7]<=0.001)   {sequence[7]=0.27*sequence[7];}
      else if (sequence[7]<=0.01)    {sequence[7]=0.31*sequence[7];}
      else if (sequence[7]<=0.1)     {sequence[7]=0.09*sequence[7];}
      else if (sequence[7]<=0.2)     {sequence[7]=0.05*sequence[7];}
      else if (sequence[7]<=0.3)     {sequence[7]=0.04*sequence[7];}
      else if (sequence[7]<=0.4)     {sequence[7]=0.03*sequence[7];}
      else if (sequence[7]<=0.5)     {sequence[7]=0.02*sequence[7];}
      else if (sequence[7]<=0.6)     {sequence[7]=0.02*sequence[7];}
      else if (sequence[7]<=0.7)     {sequence[7]=0.01*sequence[7];}
      else if (sequence[7]<=0.8)     {sequence[7]=0.02*sequence[7];}
      else if (sequence[7]<=0.9)     {sequence[7]=0.01*sequence[7];}
      else if (sequence[7]<=1)       {sequence[7]=0.16*sequence[7];}
       else                       {sequence[7]=0*sequence[7];}

      if (sequence[6]<=0.0001)       {sequence[6]=0.12*sequence[6];}
      else if (sequence[6]<=0.001)   {sequence[6]=0.27*sequence[6];}
      else if (sequence[6]<=0.01)    {sequence[6]=0.31*sequence[6];}
      else if (sequence[6]<=0.1)     {sequence[6]=0.09*sequence[6];}
      else if (sequence[6]<=0.2)     {sequence[6]=0.05*sequence[6];}
      else if (sequence[6]<=0.3)     {sequence[6]=0.04*sequence[6];}
      else if (sequence[6]<=0.4)     {sequence[6]=0.03*sequence[6];}
      else if (sequence[6]<=0.5)     {sequence[6]=0.02*sequence[6];}
      else if (sequence[6]<=0.6)     {sequence[6]=0.02*sequence[6];}
      else if (sequence[6]<=0.7)     {sequence[6]=0.01*sequence[6];}
      else if (sequence[6]<=0.8)     {sequence[6]=0.02*sequence[6];}
      else if (sequence[6]<=0.9)     {sequence[6]=0.01*sequence[6];}
      else if (sequence[6]<=1)       {sequence[6]=0.16*sequence[6];}
       else                       {sequence[6]=0*sequence[6];}

      if (sequence[5]<=0.0001)       {sequence[5]=0.12*sequence[5];}
      else if (sequence[5]<=0.001)   {sequence[5]=0.27*sequence[5];}
      else if (sequence[5]<=0.01)    {sequence[5]=0.31*sequence[5];}
      else if (sequence[5]<=0.1)     {sequence[5]=0.09*sequence[5];}
      else if (sequence[5]<=0.2)     {sequence[5]=0.05*sequence[5];}
      else if (sequence[5]<=0.3)     {sequence[5]=0.04*sequence[5];}
      else if (sequence[5]<=0.4)     {sequence[5]=0.03*sequence[5];}
      else if (sequence[5]<=0.5)     {sequence[5]=0.02*sequence[5];}
```

```
else if (sequence[5]<=0.6)     {sequence[5]=0.02*sequence[5];}
else if (sequence[5]<=0.7)     {sequence[5]=0.01*sequence[5];}
else if (sequence[5]<=0.8)     {sequence[5]=0.02*sequence[5];}
else if (sequence[5]<=0.9)     {sequence[5]=0.01*sequence[5];}
else if (sequence[5]<=1)       {sequence[5]=0.16*sequence[5];}
 else                       {sequence[5]=0*sequence[5];}

if (sequence[4]<=0.0001)       {sequence[4]=0.12*sequence[4];}
else if (sequence[4]<=0.001)   {sequence[4]=0.27*sequence[4];}
else if (sequence[4]<=0.01)    {sequence[4]=0.31*sequence[4];}
else if (sequence[4]<=0.1)     {sequence[4]=0.09*sequence[4];}
else if (sequence[4]<=0.2)     {sequence[4]=0.05*sequence[4];}
else if (sequence[4]<=0.3)     {sequence[4]=0.04*sequence[4];}
else if (sequence[4]<=0.4)     {sequence[4]=0.03*sequence[4];}
else if (sequence[4]<=0.5)     {sequence[4]=0.02*sequence[4];}
else if (sequence[4]<=0.6)     {sequence[4]=0.02*sequence[4];}
else if (sequence[4]<=0.7)     {sequence[4]=0.01*sequence[4];}
else if (sequence[4]<=0.8)     {sequence[4]=0.02*sequence[4];}
else if (sequence[4]<=0.9)     {sequence[4]=0.01*sequence[4];}
else if (sequence[4]<=1)       {sequence[4]=0.16*sequence[4];}
     else                      {sequence[4]=0*sequence[4];}


if (sequence[3]<=0.0001)       {sequence[3]=0.12*sequence[3];}
else if (sequence[3]<=0.001)   {sequence[3]=0.27*sequence[3];}
else if (sequence[3]<=0.01)    {sequence[3]=0.31*sequence[3];}
else if (sequence[3]<=0.1)     {sequence[3]=0.09*sequence[3];}
else if (sequence[3]<=0.2)     {sequence[3]=0.05*sequence[3];}
else if (sequence[3]<=0.3)     {sequence[3]=0.04*sequence[3];}
else if (sequence[3]<=0.4)     {sequence[3]=0.03*sequence[3];}
else if (sequence[3]<=0.5)     {sequence[3]=0.02*sequence[3];}
else if (sequence[3]<=0.6)     {sequence[3]=0.02*sequence[3];}
else if (sequence[3]<=0.7)     {sequence[3]=0.01*sequence[3];}
else if (sequence[3]<=0.8)     {sequence[3]=0.02*sequence[3];}
else if (sequence[3]<=0.9)     {sequence[3]=0.01*sequence[3];}
else if (sequence[3]<=1)       {sequence[3]=0.16*sequence[3];}
 else                       {sequence[3]=0*sequence[3];}

if (sequence[2]<=0.0001)       {sequence[2]=0.12*sequence[2];}
else if (sequence[2]<=0.001)   {sequence[2]=0.27*sequence[2];}
else if (sequence[2]<=0.01)    {sequence[2]=0.31*sequence[2];}
else if (sequence[2]<=0.1)     {sequence[2]=0.09*sequence[2];}
else if (sequence[2]<=0.2)     {sequence[2]=0.05*sequence[2];}
else if (sequence[2]<=0.3)     {sequence[2]=0.04*sequence[2];}
else if (sequence[2]<=0.4)     {sequence[2]=0.03*sequence[2];}
else if (sequence[2]<=0.5)     {sequence[2]=0.02*sequence[2];}
else if (sequence[2]<=0.6)     {sequence[2]=0.02*sequence[2];}
else if (sequence[2]<=0.7)     {sequence[2]=0.01*sequence[2];}
else if (sequence[2]<=0.8)     {sequence[2]=0.02*sequence[2];}
else if (sequence[2]<=0.9)     {sequence[2]=0.01*sequence[2];}
else if (sequence[2]<=1)       {sequence[2]=0.16*sequence[2];}
     else                      {sequence[2]=0*sequence[2];}

if (sequence[1]<=0.0001)       {sequence[1]=0.12*sequence[1];}
else if (sequence[1]<=0.001)   {sequence[1]=0.27*sequence[1];}
else if (sequence[1]<=0.01)    {sequence[1]=0.31*sequence[1];}
else if (sequence[1]<=0.1)     {sequence[1]=0.09*sequence[1];}
else if (sequence[1]<=0.2)     {sequence[1]=0.05*sequence[1];}
else if (sequence[1]<=0.3)     {sequence[1]=0.04*sequence[1];}
else if (sequence[1]<=0.4)     {sequence[1]=0.03*sequence[1];}
else if (sequence[1]<=0.5)     {sequence[1]=0.02*sequence[1];}
else if (sequence[1]<=0.6)     {sequence[1]=0.02*sequence[1];}
else if (sequence[1]<=0.7)     {sequence[1]=0.01*sequence[1];}
else if (sequence[1]<=0.8)     {sequence[1]=0.02*sequence[1];}
else if (sequence[1]<=0.9)     {sequence[1]=0.01*sequence[1];}
else if (sequence[1]<=1)       {sequence[1]=0.16*sequence[1];}
 else                       {sequence[1]=0*sequence[1];}

if (sequence[0]<=0.0001)       {sequence[0]=0.12*sequence[0];}
else if (sequence[0]<=0.001)   {sequence[0]=0.27*sequence[0];}
else if (sequence[0]<=0.01)    {sequence[0]=0.31*sequence[0];}
else if (sequence[0]<=0.1)     {sequence[0]=0.09*sequence[0];}
else if (sequence[0]<=0.2)     {sequence[0]=0.05*sequence[0];}
else if (sequence[0]<=0.3)     {sequence[0]=0.04*sequence[0];}
else if (sequence[0]<=0.4)     {sequence[0]=0.03*sequence[0];}
else if (sequence[0]<=0.5)     {sequence[0]=0.02*sequence[0];}
else if (sequence[0]<=0.6)     {sequence[0]=0.02*sequence[0];}
else if (sequence[0]<=0.7)     {sequence[0]=0.01*sequence[0];}
else if (sequence[0]<=0.8)     {sequence[0]=0.02*sequence[0];}
else if (sequence[0]<=0.9)     {sequence[0]=0.01*sequence[0];}
else if (sequence[0]<=1)       {sequence[0]=0.16*sequence[0];}
     else                      {sequence[0]=0*sequence[0];}
```

```
      //for (i=0; i<9; i++) {gotoxy(0,77+i); cout<<sequence2[i]<<endl;}

      for (i=0; i<24; i++) {standard ws[i]=(standard ws[i]*sequence2[0]);}
      for (i=0; i<24; i++) {ps 11[i]=(ps 11[i]*sequence2[1]);}
      for (i=0; i<24; i++) {ps 111[i]=(ps 111[i]*sequence2[2]);}
      for (i=0; i<24; i++) {ps1 1[i]=(ps1 1[i]*sequence2[3]);}
      for (i=0; i<24; i++) {ps 1 1 1[i]=(ps 1 1 1[i]*sequence2[5]);}
      for (i=0; i<24; i++) {ps 11 1 1[i]=(ps 11 1 1[i]*sequence2[6]);}
      for (i=0; i<24; i++) {ps 1 1 11[i]=(ps 1 1 11[i]*sequence2[7]);}
      for (i=0; i<24; i++) {ps_11_1_11[i]=(ps_11_1_11[i]*sequence2[8]);}

      for (i=0; i<24; i++) {standard_ws[i] = standard_ws[i] - off_standard_ws [i] +
      ps 11[i] -
      off ps 11[i] + ps_111[i] - off_ps_111[i] + ps1_1[i] - off_ps1_1[i] +
      ps 1 1 1[i] -
      off ps 1 1 1[i] + ps_11_1_1[i] - off_ps_11_1_1 [i] + ps_1_1_11[i] -
      off ps 1 1 11[i] +
      ps_11_1_11[i] - off_ps_11_1_11[i];}

      for (i=0; i<24; i++) {
      if (standard ws[i]<0) {standard ws[i]=standard ws[i]*(-1);}
      gotoxy(0,77+i); cout<<"\n"<<standard_ws[i]<<endl;
      }
      for (i=0; i<24; i++) {
      if (standard ws[i]<0) {minimum=0;}
      else                  {minimum=standard_ws[i];}
      pulse energy = minimum * photon energy;
      energy in frame = pulse energy * mark space correction;
      energy per pcm bit = (energy in frame/pcm bits);
      dBm = 10 * (log((energy per pcm bit*B)/pow(10,-3))/log(10));
      gotoxy(0,105+i); cout<<"\n"<<dBm<<endl;
      }*/
      }

//-------------------------------------------------------------------------------------//

            delete [] sequence; sequence=0;
            delete [] sequence2; sequence2=0;
           }

            sequence = new float [y*y+y-1];
            if (sequence==NULL) {gotoxy(19,50);
            cout<<"NOT ENOUGHT MEMORY FOR WS SEQUENCE ARRAY!!!!      "<<endl;}
            else                {for (i=0; i<(y*y+y-1); i++) sequence[i] = 0;}

            zero counter=0; pulse_counter=0; pulse_counter2=0; l=0; j=0;
            flag=1; limit=0;

            for (w=0; w<(y-1); w++)
            {
             for (n=0; n<pow(2,pcm_bits); n++)
             {
              for (i=0; i<x; i++)
            {
            if ( (mppm ws[i+l*(x+pcm bits+2*y)]==1)&&(i==0)&&(w==0) )   {
            zero counter=0; pulse counter=0; pulse_counter2=0;

            for (z=0; z<2; z++) {j++;}

            }
//-------------------------------------------------------------------------------------//
            else if ( (mppm ws[i+l*(x+pcm bits+2*y)]==1)&&(i!=0)&&(i!=(x-1)) ){
            zero counter=0; pulse counter=0; pulse_counter2=0;

            for (m=i-1; m>=0; m--) {

                  if (mppm ws[m+l*(x+pcm bits+2*y)]==1) {pulse counter++;}
                        else            {zero counter=m; break;}
            }

                  if ( (pulse counter==0)&&(i!=1) ) {
                        pulse counter=0;
                  for (m=zero counter-1; m>=0; m--) {

            if (mppm ws[m+l*(x+pcm bits+2*y)]==1) {pulse counter++;}
            else                                  {break;}
            }
            if ( (pulse counter==0)&&(w==0) ) {
            flag=0;
            for (z=0; z<2; z++) {j++;}
```

```
            }

            else if (pulse counter==limit+1)  {
                   for (m=i+1; m<x; m++) {

            if (mppm ws[m+l*(x+pcm bits+2*y)]==1) {pulse counter2++;}
            else                                  {break;}
                   }
//-------------------------------------------------------------------------------------//

            if (flag==0) {
            if (pulse counter2==0) { for (z=0; z<2; z++)
            {sequence[pulse counter]=sequence[pulse counter]+
            mppm ws[(l+1)*(x+pcm bits)+l*2*y+j]; j++;} flag=0;}
            else
            { for (z=0; z<2; z++) {sequence[pulse counter2*(y+1)+
              pulse counter]=sequence[pulse counter2*(y+1)+pulse counter]+
              mppm_ws[(l+1)*(x+pcm_bits)+l*2*y+j]; j++;} flag=0;}
            }
            else         {
            if (pulse counter2==0) {
            for (z=0; z<2; z++) {
            sequence[pulse counter+1]=sequence[pulse counter+1]+
           (mppm ws[(l+1)*(x+pcm bits)+l*2*y+j]*(row counter/pow(2,pcm_bits)));
            sequence[pulse counter]=sequence[pulse counter]+
           (mppm ws[(l+1)*(x+pcm_bits)+l*2*y+j]*((pow(2,pcm_bits)-row_counter)
           /pow(2,pcm_bits)));
            j++;
            }
            flag=0;
//-------------------------------------------------------------------------------------//
            }
            else                  {
            for (z=0; z<2; z++) {
            sequence[pulse counter2*(y+1)+pulse counter+1]=
            sequence[pulse counter2*(y+1)+pulse counter+1]+
           (mppm ws[(l+1)*(x+pcm bits)+l*2*y+j]*(row counter/pow(2,pcm_bits)));
            sequence[pulse counter2*(y+1)+pulse counter]=
            sequence[pulse counter2*(y+1)+pulse counter]+
           (mppm ws[(l+1)*(x+pcm_bits)+l*2*y+j]*((pow(2,pcm_bits)-row_counter)
           /pow(2,pcm_bits)));
            j++;
            }
            flag=0;
            }

            }
            }
            }
//-------------------------------------------------------------------------------------//
            else if ( (pulse counter==0)&&(i==1)&&(w==0) ) {
            for (m=i+1; m<x; m++) {

            if (mppm ws[m+l*(x+pcm bits+2*y)]==1) {pulse counter2++;}
            else                                  {break;}
            }
            if (pulse counter2==0) {
            for (z=0; z<2; z++) {
            sequence[1]=sequence[1]+(mppm ws[(l+1)*(x+pcm bits)+l*2*y+j]*
           (row counter/pow(2,pcm bits)));


            j++;
             }
              }
            else {
            pulse counter=1;
            for (z=0; z<2; z++) {
            sequence[pulse counter2*(y+1)+
            pulse counter]=sequence[pulse counter2*(y+1)+pulse counter]+
           (mppm ws[(l+1)*(x+pcm bits)+l*2*y+j]*(row counter/pow(2,pcm bits)));


            j++;
            }
            }
//-------------------------------------------------------------------------------------//
            flag=0;
            }
```

```
            else if (pulse counter==limit+1)         { for (z=0; z<2; z++) {j++;} }









            }
                else if ( (mppm ws[i+l*(x+pcm bits+2*y)]==1)&&(i==(x-1)) ) {
            zero counter=0; pulse counter=0; pulse_counter2=1;

            for (m=i-1; m>=0; m--) {

            if (mppm ws[m+l*(x+pcm bits+2*y)]==1) {pulse counter++;}
            else                                  {zero_counter=m; break;}
                }
            if (pulse counter==0)  {
            pulse counter=0;
            for (m=zero counter-1; m>=0; m--) {

            if (mppm ws[m+l*(x+pcm bits+2*y)]==1) {pulse counter++;}
            else                                  {break;}
            }
            if ( (pulse counter==0)&&(w==0) ) {
            flag=0;
            for (z=0; z<2; z++) {j++;}
            }
//-------------------------------------------------------------------------------------//
            else if (pulse counter==limit+1)  {


            if (flag==0) {
            for (z=0; z<2; z++) {
            sequence[pulse counter2*(y+1)+pulse counter]=
            sequence[pulse counter2*(y+1)+pulse counter]+(mppm ws[(l+1)*
            (x+pcm bits)+l*2*y+j]*(bit counter/pow(2,pcm bits)));
            sequence[pulse counter]=sequence[pulse counter]+
            (mppm ws[(l+1)*(x+pcm_bits)+l*2*y+j]*((pow(2,pcm_bits)-bit_counter)
            /pow(2,pcm_bits)));
            j++;
             }
            }
            else         {

            sequence[pulse counter+1]=sequence[pulse counter+1]+
             (mppm ws[(l+1)*(x+pcm bits)+l*2*y+j]*(row counter/pow(2,pcm_bits)));
            sequence[pulse counter2*(y+1)+pulse counter]=
            sequence[pulse counter2*(y+1)+pulse counter]+(mppm ws[(l+1)*
            (x+pcm bits)+l*2*y+j]*(bit counter/pow(2,pcm bits)));
            sequence[pulse counter]=sequence[pulse counter]+
            (mppm ws[(l+1)*(x+pcm bits)+l*2*y+j]*((pow(2,pcm_bits)-
             bit counter-row counter)/pow(2,pcm bits)));
            j++;

             }
              }
               }
//-------------------------------------------------------------------------------------//
            else if (pulse counter==limit+1) { for (z=0; z<2; z++) {j++;} }









            }
            if (n==pasc)                          {break;}
              }l++; j=0; if (l<=l1) {flag=1;}
                         else       {flag=0;}
               if (n==pasc)         {break;}
             }l=0; j=0; flag=1; limit++;
            }
```

```
//-------------------------------------------------------------------------------------//

     for (i=0; i<(y*y+y-1); i++)
     {sequence[i]=(sequence[i]/(pcm_bits*pow(2,pcm_bits)));}

     if ( (press4==1)&&(press5==2) ) {
     //for (i=0; i<y; i++) { gotoxy(0,58+i); cout<<"Ps.101:"<<endl; gotoxy(7,58+i);
     //cout<<sequence[i+1]<<endl;}

     //for (i=0; i<y-1; i++) { gotoxy(0,58+y+i); cout<<"Ps1.1011:"<<endl;
     //gotoxy(9,58+y+i);cout<<sequence[i+(y+2)]<<endl;}

     //for (i=0; i<y-2; i++) { gotoxy(0,58+y+1+i); cout<<"Ps1011.1:"<<endl;
     //gotoxy(0,58+y+1+i);cout<<sequence[(i+1)*(y+1)+1]<<endl;}


     k=1; n=0; w=0; z=0;
     for (i=0; i<y*y+y-1; i++) {
      if (w>60) {n++; w=0;}
       gotoxy(0+w,105+n); cout<<sequence[i]<<endl;

       w=w+12;
       if (i==k*y+z) {k++; n++; w=0; z++;}
      }
      delete [] sequence; sequence=0;

       }

     else if (press5==1)
          {
      sequence2 = new float [6];
      if (sequence2==NULL) {gotoxy(19,50);
      cout<<"NOT ENOUGHT MEMORY FOR WS SEQUENCE ARRAY!!!!      "<<endl;}
      else                 {for (i=0; i<6; i++) sequence2[i] = 0;}


     sequence2[0]=sequence[1];  sequence[1]=0;
     sequence2[2]=sequence[y+2];    sequence[y+2]=0;

     for (i=2; i<y+1; i++)     {sequence2[1]=sequence2[1]+sequence[i];
     sequence[i]=0;}
     for (i=2; i<y; i++)       {sequence2[3]=sequence2[3]+sequence[i*(y+1)+1];
     sequence[i*(y+1)+1]=0;}
     if (y>2) { for (i=y+3; i<y+6; i++)   {sequence2[4]=sequence2[4]+sequence[i];
     sequence[i]=0;} }
     else                                 {sequence2[4]=0;}
     for (i=0; i<y*y+y-1; i++) {sequence2[5]=sequence2[5]+sequence[i]; sequence[i]=0;}


//-------------------------------------------------------------------------------------//

          if (press4==1) {
        if (sequence2[0]<0) {sequence2[0]=sequence2[0]*(-1);}
                            if (sequence2[1]<0) {sequence2[1]=sequence2[1]*(-1);}
                            if (sequence2[2]<0) {sequence2[2]=sequence2[2]*(-1);}
                            if (sequence2[3]<0) {sequence2[3]=sequence2[3]*(-1);}
          if (sequence2[4]<0) {sequence2[4]=sequence2[4]*(-1);}
                            if (sequence2[5]<0) {sequence2[5]=sequence2[5]*(-1);}

           gotoxy(0,68); cout<<"Ps.101:"<<endl;
                       gotoxy(7,68);cout<<sequence2[0]<<endl;
                       gotoxy(0,69); cout<<"Ps.1101:"<<endl;
                       gotoxy(8,69);cout<<sequence2[1]<<endl;
               gotoxy(0,70); cout<<"Ps.1011:"<<endl;
                       gotoxy(8,70);cout<<sequence2[2]<<endl;
           gotoxy(0,71); cout<<"Ps.10111:"<<endl;
                       gotoxy(9,71);cout<<sequence2[3]<<endl;
           gotoxy(0,72); cout<<"Ps.11011:"<<endl;
                       gotoxy(9,72);cout<<sequence2[4]<<endl;
           gotoxy(0,73); cout<<"Ps.110111:"<<endl;
                       gotoxy(10,73);cout<<sequence2[5]<<endl;
          //for (i=0; i<6; i++) {gotoxy(0,87+i); cout<<sequence2[i]<<endl;}


//-------------------------------------------------------------------------------------//


         /*for (i=0; i<24; i++) {ps_101[i]=(ps_101[i]*sequence2[0]);}
         for (i=0; i<24; i++) {ps_1101[i]=(ps_1101[i]*sequence2[1]);}

         for (i=0; i<24; i++) {ps_1011[i]=(ps_1011[i]*sequence2[2]);}
```

```
            for (i=0; i<24; i++) {ps_10111[i]=(ps_10111[i]*sequence2[3]);}

            for (i=0; i<24; i++) {ps_11011[i]=(ps_11011[i]*sequence2[4]);}
            for (i=0; i<24; i++) {ps_110111[i]=(ps_110111[i]*sequence2[5]);}

            for (i=0; i<24; i++) {standard_ws[i] = standard_ws[i] -
            off_standard_ws[i] + ps_101[i] - off_ps_101[i] + ps_1101[i] -
            off_ps_1101[i] + ps_1011[i] - off_ps_1011[i] + ps_10111[i] -
            off_ps_10111[i] + ps_11011[i] - off_ps_11011[i] + ps_110111[i] -
            off_ps_110111[i];}

            for (i=0; i<24; i++) {
            if (standard_ws[i]<0) {standard_ws[i]=standard_ws[i]*(-1);}
            gotoxy(0,77+i); cout<<"\n"<<standard_ws[i]<<endl;
            }

            for (i=0; i<24; i++) {
            if (standard_ws[i]<0) {minimum=0;}
            else                  {minimum=standard_ws[i];}

            pulse_energy = minimum * photon_energy;
            energy_in_frame = pulse_energy * mark_space_correction;
            energy_per_pcm_bit = (energy_in_frame/pcm_bits);

            dBm = 10 * (log((energy_per_pcm_bit*B)/pow(10,-3))/log(10));
            gotoxy(0,105+i); cout<<"\n"<<dBm<<endl;
                                                }*/
                        }
             delete [] sequence; sequence=0;
             delete [] sequence2; sequence2=0;
            }

            delete [] mppm_data; mppm_data=0;
            delete [] weighted_er_mppm_data; weighted_er_mppm_data=0;
            delete [] weighted_fa_mppm_data; weighted_fa_mppm_data=0;

            delete [] mppm; mppm=0;
            delete [] data; data=0;


          }//end of if-else mppm_ws null statement

  }//end of if-press1(3) for y!=1 or y!=(x-1)
 }

//-------------------------------------------------------------------------------------------//

          counter=0;
          gotoxy(19,50);
          cout<<"WHICH ERROR RATE ARRAY TO DISPLAY?                "<<endl;
          gotoxy(64,47);cin>>press3;

          while ((press3!=4)||(press3!=5)||(press3!=6))
          {
          if (press3==4)      {
          if ( (y!=1)&&(y!=(x-1)) ) {
          if (flag_er==1) {
          gotoxy(0,56); cout<<"MPPM ERASURE ERROR RATE ARRAY                  "<<endl;
          counter=print_error_rate_arrays (mppm_er,pow(2,pcm_bits),x,y,pcm_bits,
          press3,x,y,aut_pasc,aut_pasc2,aut_pasc3,aut_pcm_bits,pcm_bits,pasc,pasc2,
          pasc3,start_number,end_number,press1,press2,press3,press4,press5);
          gotoxy(14,47); cout<<counter<<endl;
          press6=0;
//-------------------------------------------------------------------------------------------//
          do
          {
           gotoxy(19,50);
           cout<<"DO YOU WANT TO SAVE THE DATA IN A FILE?               "<<endl;
           gotoxy(33,47);cin>>press6;
           if (press6==-1) {break;}
           if (press6==2)  {break;}
           if (press6==3)  {break;}
          }while(press6!=1);
           if (press6==3) {file_op.close(); break;}
           if (press6==1) {
           k=1;
           for (i=0; i<(x+y+pcm_bits)*pow(2,pcm_bits); i++)
          {
           if (i==k*(x+y+pcm_bits)) {file_op<<"\n"; k++;}
           file_op<<mppm_er[i]<<"  ";
          }
```

```
          file op.close();
          }

          clrscr2(); break;
          }
          else
         {gotoxy(19,50);cout<<"NOT CALCULATED YET.PLEASE TRY AGAIN            "<<endl;
          gotoxy(64,47);cout<<"              "<<endl; gotoxy(64,47);cin>>press3;}
         }
         else if (y==1)  {
         if (flag er==1) {
//-------------------------------------------------------------------------------------//
         gotoxy(0,56); cout<<"MPPM ERASURE ERROR RATE ARRAY                  "<<endl;
         counter=print error rate arrays (mppm er,pow(2,pcm bits),x,1,pcm bits,press3,
         x,y,aut pasc,aut pasc2,aut pasc3,aut pcm bits,pcm bits,pasc,pasc2,pasc3,
         start number,end number,press1,press2,press3,press4,press5);
         gotoxy(14,47); cout<<counter<<endl;
         do
         {
         gotoxy(19,50);
         cout<<"DO YOU WANT TO SAVE THE DATA IN A FILE?                "<<endl;
         gotoxy(33,47);cin>>press6;
         if (press6==-1) {break;}
         if (press6==2)  {break;}
         if (press6==3)  {break;}
         }while(press6!=1);
         if (press6==3) {file_op.close(); break;}
         if (press6==1) {
         k=1;
         for (i=0; i<(x+y)*pow(2,pcm_bits); i++)
         {
         if (i==k*(x+y)) {file op<<"\n"; k++;}
         file_op<<mppm_er[i]<<"  ";
         }
         file op.close();
         }

         clrscr2(); break;
         }
         else {gotoxy(19,50);
         cout<<"NOT CALCULATED YET.PLEASE TRY AGAIN            "<<endl;
         gotoxy(64,47);cout<<"              "<<endl; gotoxy(64,47);cin>>press3;}
         }
         else if ( y==(x-1) )    {
         if (flag er==1) {
//-------------------------------------------------------------------------------------//
         gotoxy(0,56); cout<<"MPPM ERASURE ERROR RATE ARRAY                  "<<endl;
         counter=print error rate arrays (mppm er,pow(2,pcm bits),x,y,pcm bits,press3,
         x,y,aut pasc,aut pasc2,aut pasc3,aut pcm bits,pcm bits,pasc,pasc2,pasc3,
         start number,end number,press1,press2,press3,press4,press5);
         gotoxy(14,47); cout<<counter<<endl;
         do
         {
          gotoxy(19,50);
          cout<<"DO YOU WANT TO SAVE THE DATA IN A FILE?                "<<endl;
          gotoxy(33,47);cin>>press6;
          if (press6==-1) {break;}
          if (press6==2)  {break;}
          if (press6==3)  {break;}
         }while(press6!=1);
         if (press6==3) {file_op.close(); break;}
         if (press6==1) {
         k=1;
          for (i=0; i<(x+y+pcm_bits)*pow(2,pcm_bits); i++)
         {
          if (i==k*(x+y+pcm bits)) {file op<<"\n"; k++;}
          file_op<<mppm_er[i]<<"  ";
         }
          file op.close();
         }

          clrscr2(); break;
         }
         else            {gotoxy(19,50);
         cout<<"NOT CALCULATED YET.PLEASE TRY AGAIN            "<<endl;
         gotoxy(64,47);cout<<"              "<<endl; gotoxy(64,47);cin>>press3;}

         }
         }
//-------------------------------------------------------------------------------------//
         else if (press3==5) {
         if ( (y!=1)&&(y!=(x-1)) ) {
```

```
        if (flag fa==1) {
        gotoxy(0,56); cout<<"MPPM FALSE ALARM ERROR RATE ARRAY         "<<endl;
        counter=print error rate arrays (mppm fa,pow(2,pcm bits),x,y,pcm bits,
        press3,x,y,aut pasc,aut pasc2,aut pasc3,aut pcm bits,pcm bits,pasc,pasc2,
        pasc3,start number,end number,press1,press2,press3,press4,press5);
        gotoxy(14,47); cout<<counter<<endl;
        do
        {
         gotoxy(19,50);
         cout<<"DO YOU WANT TO SAVE THE DATA IN A FILE?                "<<endl;
         gotoxy(33,47);cin>>press6;
        if (press6==-1) {break;}
        if (press6==2)  {break;}
        if (press6==3)  {break;}
        }while(press6!=1);
         if (press6==3) {file_op.close(); break;}
         if (press6==1) {
        k=1;
        for (i=0; i<(x+pcm_bits+x-y)*pow(2,pcm_bits); i++)
        {
         if (i==k*(x+pcm bits+x-y)) {file op<<"\n"; k++;}
         file_op<<mppm_fa[i]<<"  ";
        }
         file op.close();
        }
//-------------------------------------------------------------------------------------//

        clrscr2(); break;
        }
        else
        {gotoxy(19,50);cout<<"NOT CALCULATED YET.PLEASE TRY AGAIN       "<<endl;
        gotoxy(64,47);cout<<"              "<<endl; gotoxy(64,47);cin>>press3;}
        }
        else if (y==1)            {
        if (flag fa==1) {
        gotoxy(0,56); cout<<"MPPM FALSE ALARM ERROR RATE ARRAY         "<<endl;
        counter=print error rate arrays (mppm fa,pow(2,pcm bits),x,y,pcm bits,
        press3,x,y,aut pasc,aut pasc2,aut pasc3,aut pcm bits,pcm bits,pasc,pasc2,
        pasc3,start number,end number,press1,press2,press3,press4,press5);
        gotoxy(14,47); cout<<counter<<endl;
        do
        {
        gotoxy(19,50);
        cout<<"DO YOU WANT TO SAVE THE DATA IN A FILE?                "<<endl;
        gotoxy(33,47);cin>>press6;
        if (press6==-1) {break;}
        if (press6==2)  {break;}
        if (press6==3)  {break;}
        }while(press6!=1);
        if (press6==3) {file_op.close(); break;}
        if (press6==1) {
        k=1;
        for (i=0; i<(2*x+pcm_bits-y)*pow(2,pcm_bits); i++)
        {
        if (i==k*(2*x+pcm bits-y)) {file_op<<"\n"; k++;}
        file_op<<mppm_fa[i]<<"  ";
         }
//-------------------------------------------------------------------------------------//
        file op.close();
        }

        clrscr2(); break;
        }
        else
        {gotoxy(19,50);cout<<"NOT CALCULATED YET.PLEASE TRY AGAIN       "<<endl;
        gotoxy(64,47);cout<<"              "<<endl; gotoxy(64,47);cin>>press3;}
        }
        else if ( y==(x-1) )      {
        if (flag fa==1) {
        gotoxy(0,56); cout<<"MPPM FALSE ALARM ERROR RATE ARRAY         "<<endl;
        counter=print error rate arrays (mppm fa,pow(2,pcm bits),x,y,1,press3,
        x,y,aut pasc,aut pasc2,aut pasc3,aut pcm bits,pcm bits,pasc,pasc2,pasc3,
        start number,end number,press1,press2,press3,press4,press5);
        gotoxy(14,47); cout<<counter<<endl;
        do
        {
        gotoxy(19,50);
        cout<<"DO YOU WANT TO SAVE THE DATA IN A FILE?                "<<endl;
        gotoxy(33,47);cin>>press6;
        if (press6==-1) {break;}
        if (press6==2)  {break;}
        if (press6==3)  {break;}
```

```
        }while(press6!=1);
        if (press6==3) {file_op.close(); break;}
        if (press6==1) {
        k=1;
        for (i=0; i<(x+x-y)*pow(2,pcm_bits); i++)
        {
        if (i==k*(x+x-y)) {file op<<"\n"; k++;}
        file_op<<mppm_fa[i]<<"  ";
        }
        file op.close();
        }
//-------------------------------------------------------------------------------------//

        clrscr2(); break;
        }
        else
        {gotoxy(19,50);cout<<"NOT CALCULATED YET.PLEASE TRY AGAIN       "<<endl;
        gotoxy(64,47);cout<<"              "<<endl; gotoxy(64,47);cin>>press3;}

        }
        }
        else if (press3==6) {
        if ( (y!=1)&&(y!=(x-1)) ) {
        if (flag ws==1) {
        gotoxy(0,56); cout<<"MPPM WRONG SLOT ERROR RATE ARRAY          "<<endl;
        counter=print error rate arrays (mppm ws,pow(2,pcm bits),x,y,pcm bits,
        press3,x,y,aut pasc,aut pasc2,aut pasc3,aut pcm bits,pcm bits,pasc,pasc2,
        pasc3,start number,end number,press1,press2,press3,press4,press5);
        gotoxy(14,47); cout<<counter<<endl;
        do
        {
        gotoxy(19,50);
        cout<<"DO YOU WANT TO SAVE THE DATA IN A FILE?                "<<endl;
        gotoxy(33,47);cin>>press6;
        if (press6==-1) {break;}
        if (press6==2)  {break;}
        if (press6==3)  {break;}
        }while(press6!=1);
        if (press6==3) {file_op.close(); break;}
        if (press6==1) {
        k=1;
        for (i=0; i<(x+pcm_bits+y+y)*pow(2,pcm_bits); i++)
        {
        if (i==k*(x+pcm bits+y+y)) {file_op<<"\n"; k++;}
        file_op<<mppm_ws[i]<<"  ";
        }
        file op.close();
        }
//-------------------------------------------------------------------------------------//

        clrscr2(); break;
        }
        else
        {gotoxy(19,50);cout<<"NOT CALCULATED YET.PLEASE TRY AGAIN       "<<endl;
        gotoxy(64,47);cout<<"              "<<endl; gotoxy(64,47);cin>>press3;}
        }
        else if (y==1)            {
        if (flag ws==1) {
        gotoxy(0,56); cout<<"MPPM WRONG SLOT ERROR RATE ARRAY          "<<endl;
        counter=print error rate arrays (mppm ws,pow(2,pcm bits),x,y,pcm bits,
        press3,x,y,aut pasc,aut pasc2,aut pasc3,aut pcm bits,pcm bits,pasc,pasc2,
        pasc3,start number,end number,press1,press2,press3,press4,press5);
        gotoxy(14,47); cout<<counter<<endl;
        do
        {
        gotoxy(19,50);
        cout<<"DO YOU WANT TO SAVE THE DATA IN A FILE?                "<<endl;
        gotoxy(33,47);cin>>press6;
        if (press6==-1) {break;}
        if (press6==2)  {break;}
        if (press6==3)  {break;}
        }while(press6!=1);
        if (press6==3) {file_op.close(); break;}
        if (press6==1) {
        k=1;
        for (i=0; i<(x+pcm_bits+y+y)*pow(2,pcm_bits); i++)
        {
        if (i==k*(x+pcm bits+y+y)) {file_op<<"\n"; k++;}
        file_op<<mppm_ws[i]<<"  ";
        }
//-------------------------------------------------------------------------------------//
        file_op.close();
```

```
        }

        clrscr2(); break;
        }
        else
        {gotoxy(19,50);cout<<"NOT CALCULATED YET.PLEASE TRY AGAIN       "<<endl;
        gotoxy(64,47);cout<<"              "<<endl; gotoxy(64,47);cin>>press3;}
        }
        else if ( y==(x-1) )      {
        if (flag ws==1) {
        clrscr2();
        gotoxy(0,56); cout<<"MPPM WRONG SLOT ERROR RATE ARRAY          "<<endl;
        counter=print error rate arrays (mppm ws,pow(2,pcm bits),x,y,pcm bits,
        press3,x,y,aut pasc,aut pasc2,aut pasc3,aut pcm bits,pcm bits,pasc,
        pasc2,pasc3,start number,end number,press1,press2,press3,press4,press5);
        gotoxy(14,47); cout<<counter<<endl;
        do
        {
        gotoxy(19,50);
        cout<<"DO YOU WANT TO SAVE THE DATA IN A FILE?                "<<endl;
        gotoxy(33,47);cin>>press6;
        if (press6==-1) {break;}
        if (press6==2)  {break;}
        if (press6==3)  {break;}
        }while(press6!=1);
        if (press6==3) {file_op.close(); break;}
        if (press6==1) {
        k=1;
//-------------------------------------------------------------------------------------//
        for (i=0; i<(x+pcm_bits+y+y)*pow(2,pcm_bits); i++)
        {
        if (i==k*(x+pcm bits+y+y)) {file_op<<"\n"; k++;}
        file_op<<mppm_ws[i]<<"  ";
        }
        file op.close();
        }

        clrscr2(); break;
        }
        else
        {gotoxy(19,50);cout<<"NOT CALCULATED YET.PLEASE TRY AGAIN       "<<endl;
         gotoxy(64,47);cout<<"              "<<endl; gotoxy(64,47);cin>>press3;}

        }
        }
        else if ( (press3==8)||(press3==-1) ) {break;}
        else { gotoxy(19,50);
        cout<<"NOT VALID SELECTION.PLEASE TRY AGAIN          "<<endl;
        gotoxy(64,47);cin>>press3;}
        }

        if (press3==-1) {break;}

       gotoxy(19,50);
        cout<<"TO EXIT PRESS -1:                            "<<endl;
        gotoxy(36,50);cin>>question;

 }//end of while-loop

}//end of main

//-------------------------------------------------------------------------------------//




//-------------------------------------------------------------------------------------//
//                                  FUNCTIONS                                          //
//-------------------------------------------------------------------------------------//
//-Control Panel
void control_panel()
{

//-------------------------------------------------------------------------------------//

gotoxy(0,0);
cout<<"||_____________________________________________________________________________
```

```
 ||";
cout<<"||_____________________________________________________________________________
 ||";
cout<<"||                     | MPPM BIT ERROR RATES CALCULATION |
 ||";
cout<<"||_____________________________________________________________________________
 ||";
cout<<"||_____________________________________________________________________________
 ||";
cout<<"||                      |      INPUT CONTROL PANEL    |
 ||";
cout<<"||                      |TO EXIT PRESS: -1 AT ANY TIME|
 ||";
cout<<"|| MANUAL INPUTS        |_____________________________| AUTOMATIC CALCULATIONS
 ||";
cout<<"||                      | ENTER CORRECT INTEGERS ONLY |       MAXIMUM
 ||";
cout<<"||----------------------|_____________________________|-----------------------
-||";
cout<<"|| Number of Slots:                    | Maximum MPPM Number:
 ||";
cout<<"|| Number of Pulses:                   | Erasure Weight:
 ||";
cout<<"|| Pascal's Number:                    | False Alarm Weight:
 ||";
cout<<"|| PCM BITS:                           | Pascal's Number:
 ||";
cout<<"||                                     | PCM BITS:
 ||";
cout<<"|| Erasure Number:                     | Erasure Number:
 ||";
cout<<"|| False Alarm Number:                 | False Alarm Number:
 ||";
cout<<"||_____________________________________|_____________________________________
 ||";
cout<<"||
 ||";
cout<<"||                 PRESS 1, 2 OR 3 TO CHOOSE BIT ERROR RATE
 ||";
cout<<"||
 ||";
cout<<"|| 1 ERASURE ERROR           2 FALSE ALARM ERROR        3 WRONG SLOT ERROR
 ||";
cout<<"||
 ||";
cout<<"||                       Choice:
 ||";
cout<<"||_____________________________________________________________________________
 ||";
cout<<"||                      |          PCM DATA           |
 ||";
cout<<"|| 1 LINEAR INCREMENT   |_____________________________| 6
 ||";
cout<<"|| 2 LINEAR DECREMENT   | PRESS 1, 2, 3, 4, 5, 6, 7, 8| 7
 ||";
cout<<"|| 3 GRAY-CODE NUMBER   |  9 OR 10 FOR THE DATA USING | 8
 ||";
cout<<"|| 4 RANDOM NUMBERS     | LINEAR MAPPING OR ALGORITHMS| 9
 ||";
cout<<"|| 5 READ NUMBERS       |-----------------------------| 10
 ||";
cout<<"||                      |                             |
 ||";
cout<<"|| Number:              | Choice:                     | Range:     -
 ||";
cout<<"||______________________|_____________________________|_____________________
 ||";
cout<<"||                      |           OUTPUTS           |
 ||";
cout<<"||      ALGORITHM       |_____________________________|_____________________
 ||";
cout<<"||----------------------|  PRESS 1, 2, 3 UNTIL 9 TO   |         ARRAYS
 ||";
cout<<"|| 1 2-Pulse 2 Original |  PRINT MAPPING ARRAYS AND   |----------------------
-||";
cout<<"||                      |  MPPM ERROR RATE ARRAYS AND | 1 MPPM-DATA ARRAY
 ||";
cout<<"|| SELECT:              |  THEN PRESS 1, 2 UNTIL 3 TO | 2 ERASURE WEIGHT
 ||";
cout<<"||----------------------|    MANIPULATE THE RESULTS   | 3 FALSE ALARM WEIGHT
 ||";
cout<<"||  DISPLAY SEQUENCES   |-----------------------------| 4 ERASURE PCM ERROR
```

86

```
 ||";
cout<<"||----------------------|                           | 5 FALSE ALARM PCM ERR
.||";
cout<<"|| 1 YES        2 NO    | 1 WRITE ARRAY  2 DW/LD FPGA | 6 WRONG SLOT PCM ERRO
R||";
cout<<"|| Select:              |         3 CONTINUE          |
 ||";
cout<<"||----------------------|                           | 8 NO DISPLAY-EXIT
 ||";
cout<<"||                      |                           | 9 RANDOM DATA NUMBERS
 ||";
cout<<"|| Statistics:          | Select:                   | Chooice:
 ||";
cout<<"||______________________|___________________________|_____________________
 ||";
cout<<"||_____________________________________________________________________
 ||";
cout<<"|| PROGRAM MESSAGE:
 ||";
cout<<"||_____________________________________________________________________
 ||";
cout<<"||  _____________________________________________________________________
 ||"<<endl;
} //-------54*T

//-------------------------------------------------------------------------------------//




//-Pascal's Number Calculation
unsigned long double pascal_calc (int x, int y)
{
  unsigned long int a,b,c,value,value2,value3,pasc; int i; //T

  if (y>=x-y)                                          //T
  { value=y+1; a=y+2; value3=x-y; c=x-y-1;             //T

    for (i=1; i<x-y; i++)                              //(x-y)*T
      {value=value*a; a++;}
    for (i=1; i<x-y;  i++)                             //(x-y)*T
      {value3=value3*c; c--;}
    pasc=value/value3;                                 //T
  } else                                               //-------4*T+(2*(x-y)*T)
       { value=x-y+1; a=x-y+2; value2=y ; b=y-1;       //T

         for (i=1; i<y; i++)                           //y*T
           {value=value*a;a++;}
         for ( i=1; i<y; i++)
         {value2=value2*b;b--;}                        //y*T
         pasc=value/value2;
       } return pasc;                                  //T
}                                                      //-------4*T+2*y*T
                                                       //8variables=7long double+1integer



//-------------------------------------------------------------------------------------//




//-Print Random Data-vertical printing-
void print random array (int *data, unsigned long int pasc 1, int x_1, int x,
                         int y, unsigned long int aut pasc,
                         unsigned long int aut pasc2,
                         unsigned long int aut pasc3, int aut pcm bits, int pcm_bits,
                         unsigned long int pasc, unsigned long int pasc2,
                         unsigned long int pasc3, int start number, int end number,
                         int press1, int press2, int press3, int press4, int press5)
{ int i; int j; int k=0; int number=0; int l=x-1; char dummy=0;   //T

    for (j=0; j<pasc_1; j++)        //pasc
```

```
    { for (i=0; i<x 1; i++)     //x
      { if ( data[k*x 1+i]==1 ) number=number+pow(2,1-i); //T
        else                    number=number;
      } cout<<number<<endl; number=0; k++;   //T
    } cout<<"\n\nPress any button and then enter to continue: "; cin>>dummy;//T
    clrscr();
    control panel();
    fill data(x,y,aut pasc,aut pasc2,aut pasc3,aut pcm bits,pcm bits,pasc,pasc2,
              pasc3,start number,end_number,press1,press2,press3,press4,press5);
} //-----((x*T+T)*pasc)+(2*T)
  //-----6 variables = 5 integer + 1 char



//-------------------------------------------------------------------------------------//



//-Print Random Data2-horizontal printing-
void print random array2 (int *data, unsigned long int pasc 1, int x_1, int x,
                          int y, unsigned long int aut pasc,
                          unsigned long int aut pasc2, unsigned long int aut pasc3,
                          int aut pcm bits, int pcm bits, unsigned long int pasc,
                          unsigned long int pasc2, unsigned long int pasc3,
                          int start number, int end number, int press1, int press2,
                          int press3, int press4, int press5)
{ int i; int j; int k=0; int number=0; int l=x-1; char dummy=0;

    for (j=0; j<pasc 1; j++) //pasc
    { for (i=0; i<x 1; i++) //x
      { if ( data[k*x 1+i]==1 ) number=number+pow(2,1-i); //T
        else                    number=number;
      } cout<<number<<'\t'; number=0; k++; //T
    } cout<<"\n\nPress any button and then enter to continue: "; cin>>dummy; //T
    clrscr();
    control panel();
    fill data(x,y,aut pasc,aut pasc2,aut pasc3,aut pcm bits,pcm bits,pasc,pasc2,pasc3,
              start number,end number,press1,press2,press3,press4,press5);
} //----6 variables = 5 integers + 1 char
  //----(pasc*(x*T+T))+T


//-------------------------------------------------------------------------------------//


//-Print Input Arrays
void print input arrays (int *mppm, unsigned long int pasc1, int x1, int x, int y,
                         unsigned long int aut pasc, unsigned long int aut pasc2,
                         unsigned long int aut pasc3, int aut pcm bits, int pcm_bits,
                         unsigned long int pasc, unsigned long int pasc2,
                         unsigned long int pasc3, int start number, int end number,
                         int press1, int press2, int press3, int press4, int press5)
{ int i=0;int j=0;int k=0;const limit=290;char dummy=0;

    for (i=0; i<x1*pasc1; i++)
    { if (k==x1*limit)
      {
        cout<<"\n\nPress any button and then enter to continue printing: ";
              cin>>dummy;

        clrscr();
        control panel();
        fill data(x,y,aut pasc,aut pasc2,aut pasc3,aut pcm bits,
                      pcm bits,pasc,pasc2,pasc3,start number,end_number,
                      press1,press2,press3,press4,press5);
        gotoxy(0,56); cout<<"\n"; k=0;//3*T
      }
      if (j==x1) { cout<<"\n"; j=0; } //2*T
      cout<<mppm[i]; j++; k++;  //T
    } cout<<"\n\nPress any button and then enter to print the next array: ";
      cin>>dummy; //T
  clrscr();
  control panel();
  fill data(x,y,aut pasc,aut pasc2,aut pasc3,aut pcm bits,pcm bits,pasc,
            pasc2,pasc3,start_number,end_number,press1,press2,press3,
            press4,press5);
} //------5 variables = 4 integers + 1 char
  //------(6*T)*x*pasc+(2*T)
```

```
//-------------------------------------------------------------------------------------//

void fill data (int x, int y, unsigned long int aut pasc, unsigned long int aut_pasc2,
                unsigned long int aut pasc3, int aut pcm bits, int pcm_bits,
                unsigned long int pasc, unsigned long int pasc2,
                unsigned long int pasc3, int start number, int end number,
                int press1, int press2, int press3, int press4, int press5)
{
  gotoxy(19,10); cout<<x<<endl;
  gotoxy(20,11); cout<<y<<endl;
  gotoxy(57,13); cout<<aut pasc<<endl;
  gotoxy(56,15); cout<<aut pasc2<<endl;
  gotoxy(60,16); cout<<aut pasc3<<endl;
  gotoxy(50,14); cout<<aut pcm bits<<endl;
  gotoxy(61,10); cout<<pow(2,aut pcm bits)<<endl;
  gotoxy(72,10); cout<<pow(2,aut_pcm_bits)-aut_pasc<<endl;

  gotoxy(12,13); cout<<pcm bits<<endl;
  gotoxy(19,12); cout<<pasc<<endl;
  gotoxy(18,15); cout<<pasc2<<endl;
  gotoxy(22,16); cout<<pasc3<<endl;
  gotoxy(33,23); cout<<press1<<endl;
  gotoxy(33,32); cout<<press2<<endl;

  if ( (press2==1)||(press2==2)||(press2==3) ) {
                                                gotoxy(10,32);
                                                cout<<start_number<<endl;
                                               }
  else  {
         gotoxy(62,32); cout<<start number<<endl;
         gotoxy(68,32); cout<<end_number<<endl;
        }

  gotoxy(64,47); cout<<press3<<endl;
  gotoxy(10,44); cout<<press4<<endl;
  gotoxy(10,39); cout<<press5<<endl; //20T
}

//-------------------------------------------------------------------------------------//

//-Print Error Rate Arrays
int print error rate arrays (int *mppm, unsigned long int pasc 1, int x_1, int y_1,
                             int pcm bits 1, int press, int x, int y,
                             unsigned long int aut pasc, unsigned long int aut_pasc2,
                             unsigned long int aut pasc3, int aut pcm_bits,
                             int pcm bits, unsigned long int pasc,
                             unsigned long int pasc2, unsigned long int pasc3,
                             int start number, int end number, int press1,
                             int press2, int press3, int press4, int press5)
{
  int i; int j=0; const limit=290; char dummy=0; int p=0; int k=0; int counter=0;
  int x1; int y1;

  if ( (press==4)&&(y 1!=1)&&( y!=(x 1-1) ) )        {x1=x 1+y+pcm_bits_1;
                                                      y1=y 1;}
  else if ( (press==4)&&(y 1==1) )                   {x1=x 1+1;
                                                      y1=y 1;}
  else if ( (press==4)&&( y 1==(x 1-1) ) )           {x1=x 1+y+pcm_bits_1;
                                                      y1=y_1;}

  else if ( (press==5)&&(y 1!=1)&&( y 1!=(x 1-1) ) ) {x1=2*x 1+pcm bits 1-y_1;
                                                      y1=x 1-y 1;}
  else if ( (press==5)&&(y 1==1) )                   {x1=2*x 1+pcm_bits_1-y_1;
                                                      y1=x 1-y 1;}
  else if ( (press==5)&&( y 1==(x 1-1) ) )           {x1=2*x 1-y 1;
                                                      y1=x_1-y_1;}

  else if ( (press==6)&&(y 1!=1)&&( y 1!=(x 1-1) ) ) {x1=x 1+pcm_bits_1+2*y;
                                                      y1=2*y;}
  else if ( (press==6)&&(y 1==1) )                   {x1=x 1+pcm bits 1+2*y;
                                                      y1=2*y;}
  else if ( (press==6)&&(y 1==x 1-1) )               {x1=x 1+pcm bits_1+2*y;
                                                      y1=2*y;} //18*T

//-------------------------------------------------------------------------------------//

  if ( x1<65 ) //x*pasc*T
  {
   for (i=0; i<x1*pasc_1; i++)
```

```
    {
     if (k==x1*limit)
     {
      cout<<"\n\nPress any button and then enter to continue printing: "<<endl;
      cin>>dummy;
      clrscr();
      control panel();
      fill data(x,y,aut_pasc,aut_pasc2,aut_pasc3,aut_pcm_bits,pcm_bits,pasc,pasc2,
                pasc3,
                start number,end number,press1,press2,press3,press4,press5);
      gotoxy(0,56); cout<<"\n"; k=0;
     } if (j==x1-y1+p) {cout<<"  ";p++;}
       if (j==x1)      {cout<<"\n"; j=0;}
       if (p==y1)      {p=0;}
       cout<<mppm[i]; j++; k++;
    } cout<<"\n\nPress any button and then enter to print the next array: "<<endl;
      cin>>dummy;
   }
   else {
         for (i=0; i<x1*pasc_1; i++)
         {
          if (k==x1*limit)
          {
            cout<<"\n\nPress any button and then enter to continue printing: "
                <<endl;
                cin>>dummy;
                clrscr();
           control panel();
           fill data(x,y,aut pasc,aut_pasc2,aut_pasc3,aut_pcm_bits,pcm_bits,
                             pasc,
                             pasc2,pasc3,start number,end_number,press1,press2,
                             press3,press4,press5);
           gotoxy(0,56); cout<<"\n"; k=0;
          } if (j==x1-y1+p) {cout<<mppm[i]<<"  "; p++;}
            if (p==y1) {cout<<"\n";p=0;}
            if (j==x1) {j=0;}
            j++; k++;
         } cout<<"\n\nPress any button and then enter to print the next array: "
           <<endl;
           cin>>dummy;
        }

//-------------------------------------------------------------------------------------//

   counter=0; //T

   if ( (press==4)&&(y_1!=1)&&( y_1!=(x_1-1) ) )
   {
    for (i=0; i<pasc_1; i++)
    {
     for (j=0; j<y1; j++)
     {
      counter=counter+mppm[((i+1)*(x_1+pcm_bits))+(i*y1)+j]; //14*pasc*y*T
     }
    }
   }
   else if ( (press==4)&&(y_1==1) )
        {
         for (i=0; i<pasc_1; i++) {counter=counter+mppm[((i+1)*x_1)+i];}
        }
   else if ( (press==4)&&(y_1==(x_1-1)) )
        {
          for (i=0; i<pasc_1; i++)
          {
           for (j=0; j<y1; j++)
           {
            counter=counter+mppm[((i+1)*(x_1+pcm_bits_1))+(i*y1)+j];
           }
          }
        }
   else if ( (press==5)&&(y_1!=1)&&( y!=(x_1-1) ) )
        {
         for (i=0; i<pasc_1; i++)
         {
          for (j=0; j<y1; j++)
          {
           counter=counter+mppm[((i+1)*(x_1+pcm_bits_1))+(i*y1)+j];
          }
         }
        }
   else if ( (press==5)&&(y_1==1) )
        {
```

```
        for (i=0; i<pasc_1; i++)
        {
         for (j=0; j<y1; j++)
         {
          counter=counter+mppm[((i+1)*x_1)+(i*y1)+j];
         }
        }
       }
//-------------------------------------------------------------------------------------//
  else if ( (press==5)&&( y==(x-1) ) )
       {
        for (i=0; i<pasc_1; i++)
        {
         counter=counter+pcm_bits;
        }
       }
  else if ( (press==6)&&(y_1!=1)&&( y_1!=(x_1-1) ) )
       {
        for (i=0; i<pasc_1; i++)
        {
         for (j=0; j<y1; j++)
         {
          counter=counter+mppm[((i+1)*(x_1+pcm_bits_1))+(i*y1)+j];
         }
        }
       }
  else if ( (press==6)&&(y_1==1) )
       {
        for (i=0; i<pasc_1; i++)
        {
         for (j=0; j<y1; j++)
         {
          counter=counter+mppm[((i+1)*x_1)+(i*y1)+j];
         }
        }
       }
  else if ( (press==6)&&( y==(x-1) ) )
       {
        for (i=0; i<pasc_1; i++)
        {
         for (j=0; j<y1; j++)
         {
          counter=counter+mppm[((i+1)*(x_1+pcm_bits_1))+(i*y1)+j];
         }
        }
       }
  clrscr();
  control panel(); //3T
  fill data(x,y,aut pasc,aut pasc2,aut pasc3,aut pcm bits,pcm bits,pasc,pasc2,pasc3,
            start_number,end_number,press1,press2,press3,press4,press5);
//-------------------------------------------------------------------------------------//
  return counter; //T
}
//----9 variables = 8 int + 1 char
```

```
//--------------------------------------------------------------------------//
//                                  COMMENTS                                //
//--------------------------------------------------------------------------//
//Program:     Create Permutation according to MPPM codewords               //
//Cut:         Final                                                        //
//Date:        13/09/07                                                     //
//Author:      Konstantinos Nikolaidis                                      //
//Inputs:      Keyboard (Pascal's Number)                                   //
//Description: Save all possible permuations of an MPPM mapping             //
//                                                                          //
//Time Analysis:  BEST CASE: 22*T+N*T     WORST CASE: 24*T+2*N*T+size*T    //
//                                                                          //
//Space Analysis: 112 bits + 1 int pointer                                  //
//                                                                          //
//--------------------------------------------------------------------------//
//                                  CLASSES                                 //
//--------------------------------------------------------------------------//


#include <stdio.h>
#include <iostream.h>
#include <fstream.h>


unsigned long int counter; //global variable

//--------------------------------------------------------------------------//

void print(const int *v, const int size) //print function
{
 fstream file_op1("mapping_dec.txt",ios::app|ios::in);
 //write in decimal form all permutaions-possible mappings
 if (!file_op1) {cerr<<"File could not be opened"<<endl;}

 if (v != 0) {
              for (int i = 0; i < size; i++) {
                                               file_op1<<v[i]<<"  ";

                                             }
              file_op1<<endl;
              counter++; //2*T+size*T
             }

 file_op1.close(); //4*T
} //best-4*T  worst-6*T+size*T

//--------------------------------------------------------------------------//

void visit(int *Value, int N, int k)
{
 static level = -1;

 level = level+1; Value[k] = level;

 if (level == N) {print(Value, N);}
 else {
       for (int i = 0; i < N; i++) {
                                     if (Value[i]==0) {visit(Value, N, i);}
                                    }
      }
 level = level-1; Value[k] = 0; //best-6*T worst-6*T+N*T
}

//--------------------------------------------------------------------------//

void main()
{
 int N;
 int *Value;
 counter=0;

 while (1)
 {
  counter=0;
  cout<<"Enter Pascal's Number Please?"; //2^pcm_bits=Pascal's Number
  cin>>N;

  if (N==-1) {break;} //exit with -1

  Value = new int [N];

  for (int i = 0; i < N; i++) {
```

```
                                Value[i] = 0;
                               }
  visit(Value, N, 0);

  delete [] Value; Value=0;

  cout<<"\nThe number of different mappings is:"<<counter<<endl; //12*T+N*T
 }

}
//---------------------------------------------------------------------------//
```

2

APPENDIX D

Mathematical Analysis
Matched Filtering

12-2 MPPM SYSTEM

NORMALIZED BANDWIDTH OF 30

MPPM investigation using a MATCHED and zero gu with 1Gbit/s data and WHITE noise

Set up the scan limits

$i := 0,1..9$ $\quad n := 10$

$v_i := v_{off} + \frac{i}{range}$

$x := 0..n$ This gives the row of the matrix

$y := 0..n$ This gives the column of the matrix

$v =$

	0
0	0.519
1	0.521
2	0.521
3	0.522
4	0.523
5	0.524
6	0.525
7	0.526
8	0.527
9	0.529

Preamplifier terms

$f_p := 1.2 \cdot 10^9$ Preamp bandwidth

$S_o := 16 \cdot 10^{-24}$ Preamp noise at input - double sided Philips TZA 3043

$B := 1 \cdot 10^9$ Bit rate

$T_b := \frac{1}{B}$ PCM bit time

$T_s := \frac{pcm\_bits \cdot T_b}{X}$ Slot time

$n := 10$ Number of like symbols in PCM

$\eta q := 1.6 \cdot 10^{-19}$ Quantum energy

$\lambda := 1.55 \cdot 10^{-6}$ This is the wavelength of operation

$photon\_energy := \frac{6.63 \cdot 10^{-34} \cdot 3 \cdot 10^8}{\lambda}$

$R_o := \frac{\eta q}{photon\_energy}$

Pulse shape terms

$\alpha := \frac{0.1874 \cdot T_b}{f_n}$

$\alpha_n := \frac{0.1874 \cdot T_b}{f_n \cdot T_s}$

$R_o = 1.247$

$\alpha_n = 0.012$

$\omega_p := 2 \cdot \pi \cdot f_p$

$\omega_{pn} := 2 \cdot \pi \cdot f_p \cdot T_s$

$\tau_R := \alpha$ $\qquad \tau_{Rn} := \frac{\alpha}{T_s}$ $\qquad \tau_{Rn} = 0.012$

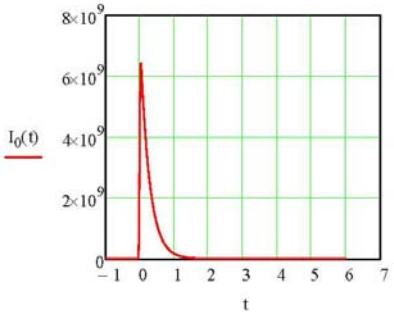
$$Pulse(t) := \frac{1}{\sqrt{2\cdot\pi}\cdot\alpha_n\cdot T_s}\cdot\exp\left(\frac{-t^2}{2\cdot\alpha_n{}^2}\right)$$

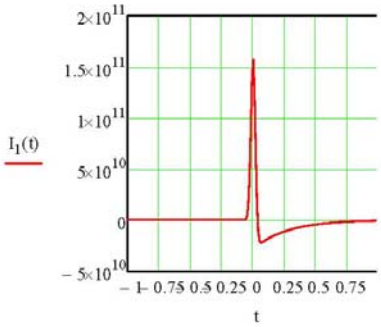
Pulse shape
Matched filter only

$$I_0(t) := \frac{\omega_{pn}}{2\cdot T_s}\cdot\exp\left[\left(\alpha_n\cdot\omega_{pn}\right)^2\right]\cdot\exp\left(-\omega_{pn}\cdot t\right)\cdot\operatorname{erfc}\left(\alpha_n\cdot\omega_{pn}-\frac{t}{2\cdot\alpha_n}\right)$$

$$I_1(t) := \frac{\omega_{pn}^{\,2}}{2\cdot T_s}\cdot\exp\left[\left(\alpha_n\cdot\omega_{pn}\right)^2\right]\cdot\exp\left(-\omega_{pn}\cdot t\right)\cdot\left[\frac{\exp\left[-\left(\alpha_n\cdot\omega_{pn}-\frac{t}{2\cdot\alpha_n}\right)^2\right]}{\alpha_n\cdot\omega_{pn}\cdot\sqrt{\pi}}-\operatorname{erfc}\left(\alpha_n\cdot\omega_{pn}-\frac{t}{2\cdot\alpha_n}\right)\right]$$

$t := -1, -0.99 .. 6$





$$noise := \frac{\omega_{pn}}{2\cdot T_s}\cdot\exp\left[\left(\alpha_n\cdot\omega_{pn}\right)^2\right]\cdot\operatorname{erfc}\left(\alpha_n\cdot\omega_{pn}\right)$$

$noise = 3.578 \times 10^9$

Set up the pulse shapes

$12-2$ and $12-3$ Systems

$v_o(t) := I_0(t)$

$v_{o10}(t) := I_0(t)$

$v_{o110}(t) := I_0(t) + I_0(t-1)$

$slope_o(t) := I_1(t)$

$slope_{10}(t) := I_1(t)$

$slope_{110}(t) := I_1(t) + I_1(t-1)$

$v_{o1110}(t) := I_0(t) + I_0(t-1) + I_0(t-2)$ $\quad slope_{1110}(t) := I_1(t) + I_1(t-1) + I_1(t-2)$

$v_{o101}(t) := I_0(t) + I_0(t-2)$ $\quad slope_{101}(t) := I_1(t) + I_1(t-2)$

$v_{o1101}(t) := I_0(t) + I_0(t-1) + I_0(t-3)$ $\quad slope_{1101}(t) := I_1(t) + I_1(t-1) + I_1(t-3)$

$v_{o1011}(t) := I_0(t) + I_0(t-2) + I_0(t-3)$ $\quad slope_{1011}(t) := I_1(t) + I_1(t-2) + I_1(t-3)$

$v_{o10101}(t) := I_0(t) + I_0(t-2) + I_0(t-4)$ $\quad slope_{10101}(t) := I_1(t) + I_1(t-2) + I_1(t-4)$

up to $12-4$ System

$v_{o11110}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3)$

$v_{o11101}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-4)$

$v_{o11011}(t) := I_0(t) + I_0(t-1) + I_0(t-3) + I_0(t-4)$

$v_{o10111}(t) := I_0(t) + I_0(t-2) + I_0(t-3) + I_0(t-4)$

$slope_{11110}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3)$

$slope_{11101}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-4)$

$slope_{11011}(t) := I_1(t) + I_1(t-1) + I_1(t-3) + I_1(t-4)$

$slope_{10111}(t) := I_1(t) + I_1(t-2) + I_1(t-3) + I_1(t-4)$

up to $12-5$ System

$v_{o111110}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-4)$

$v_{o111101}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-5)$

$v_{o111011}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-4) + I_0(t-5)$

$v_{o110111}(t) := I_0(t) + I_0(t-1) + I_0(t-3) + I_0(t-4) + I_0(t-5)$

$v_{o101111}(t) := I_0(t) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-5)$

$slope_{111110}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-4)$

$slope_{111101}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-5)$

$slope_{111011}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-4) + I_1(t-5)$

$slope_{110111}(t) := I_1(t) + I_1(t-1) + I_1(t-3) + I_1(t-4) + I_1(t-5)$

$slope_{101111}(t) := I_1(t) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-5)$

up to $12-6$ System

$v_{o1111110}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-5)$

$v_{o1111101}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-6)$

$$v_{o1111011}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-5) + I_0(t-6)$$

$$v_{o1110111}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-4) + I_0(t-5) + I_0(t-6)$$

$$v_{o1101111}(t) := I_0(t) + I_0(t-1) + I_0(t-3) + I_0(t-4) + I_0(t-5) + I_0(t-6)$$

$$v_{o1011111}(t) := I_0(t) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-5) + I_0(t-6)$$

$$slope_{1111110}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-5)$$

$$slope_{1111101}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-6)$$

$$slope_{1111011}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-5) + I_1(t-6)$$

$$slope_{1110111}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-4) + I_1(t-5) + I_1(t-6)$$

$$slope_{1101111}(t) := I_1(t) + I_1(t-1) + I_1(t-3) + I_1(t-4) + I_1(t-5) + I_1(t-6)$$

$$slope_{1011111}(t) := I_1(t) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-5) + I_1(t-6)$$

up to $12-7$ System

$$v_{o11111110}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-5) + I_0(t-6)$$

$$v_{o11111101}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-5) + I_0(t-7)$$

$$v_{o11111011}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-6) + I_0(t-7)$$

$$v_{o11110111}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-5) + I_0(t-6) + I_0(t-7)$$

$$v_{o11101111}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-4) + I_0(t-5) + I_0(t-6) + I_0(t-7)$$

$$v_{o11011111}(t) := I_0(t) + I_0(t-1) + I_0(t-3) + I_0(t-4) + I_0(t-5) + I_0(t-6) + I_0(t-7)$$

$$v_{o10111111}(t) := I_0(t) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-5) + I_0(t-6) + I_0(t-7)$$

$$slope_{11111110}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-5) + I_1(t-6)$$

$$slope_{11111101}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-5) + I_1(t-7)$$

$$slope_{11111011}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-6) + I_1(t-7)$$

$$slope_{11110111}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-5) + I_1(t-6) + I_1(t-7)$$

$$slope_{11101111}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-4) + I_1(t-5) + I_1(t-6) + I_1(t-7)$$

$$slope_{11011111}(t) := I_1(t) + I_1(t-1) + I_1(t-3) + I_1(t-4) + I_1(t-5) + I_0(t-6) + I_0(t-7)$$

$$slope_{10111111}(t) := I_1(t) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-5) + I_0(t-6) + I_0(t-7)$$

up to $12-8$ System

$$v_{o11111111_0}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-5) + I_0(t-6) + I_0(t-7)$$

$$v_{o11111110_1}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-5) + I_0(t-6) + I_0(t-8)$$

$$v_{o11111101_1}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-5) + I_0(t-7) + I_0(t-8)$$

$$v_{o11111011_1}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-6) + I_0(t-7) + I_0(t-8)$$

$$v_{o11110111_1}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-5) + I_0(t-6) + I_0(t-7) + I_0(t-8)$$

$$v_{o11101111_1}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-4) + I_0(t-5) + I_0(t-6) + I_0(t-7) + I_0(t-8)$$

$$v_{o11011111_1}(t) := I_0(t) + I_0(t-1) + I_0(t-3) + I_0(t-4) + I_0(t-5) + I_0(t-6) + I_0(t-7) + I_0(t-8)$$

$$v_{o10111111_1}(t) := I_0(t) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-5) + I_0(t-6) + I_0(t-7) + I_0(t-8)$$

$$slope_{11111111_0}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-5) + I_1(t-6) + I_1(t-$$

$$slope_{11111110_1}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-5) + I_1(t-6) + I_1(t-$$

$$slope_{11111101_1}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-5) + I_1(t-7) + I_1(t-$$

$$slope_{11111011_1}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-6) + I_1(t-7) + I_1(t-$$

$$slope_{11110111_1}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-5) + I_1(t-6) + I_1(t-7) + I_1(t-$$

$$slope_{11101111_1}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-4) + I_1(t-5) + I_1(t-6) + I_1(t-7) + I_1(t-$$

$$slope_{11011111_1}(t) := I_1(t) + I_1(t-1) + I_1(t-3) + I_1(t-4) + I_1(t-5) + I_1(t-6) + I_1(t-7) + I_1(t-$$

$$slope_{10111111_1}(t) := I_1(t) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-5) + I_1(t-6) + I_1(t-7) + I_1(t-$$

up to $12-9$ System

$$v_{o111111111_0}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-5) + I_0(t-6) + I_0(t-7)$$

$$v_{o111111110_1}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-5) + I_0(t-6) + I_0(t-7)$$

$$v_{o111111101_1}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-5) + I_0(t-6) + I_0(t-8)$$

$$v_{o111111011_1}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-5) + I_0(t-7) + I_0(t-8)$$

$$v_{o111110111_1}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-6) + I_0(t-7) + I_0(t-8)$$

$$v_{o111101111_1}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-5) + I_0(t-6) + I_0(t-7) + I_0(t-8)$$

$$v_{o111011111_1}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-4) + I_0(t-5) + I_0(t-6) + I_0(t-7) + I_0(t-8)$$

$$v_{o1101111111}(t) := I_0(t) + I_0(t-1) + I_0(t-3) + I_0(t-4) + I_0(t-5) + I_0(t-6) + I_0(t-7) + I_0(t-8)$$
$$v_{o1011111111}(t) := I_0(t) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-5) + I_0(t-6) + I_0(t-7) + I_0(t-8)$$

$$slope_{1111111110}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-5) + I_1(t-6) + I_1(t-$$

$$slope_{1111111101}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-5) + I_1(t-6) + I_1(t-$$

$$slope_{1111111011}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-5) + I_1(t-6) + I_1(t-$$

$$slope_{1111110111}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-5) + I_1(t-7) + I_1(t-$$

$$slope_{1111101111}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-6) + I_1(t-7) + I_1(t-$$

$$slope_{1111011111}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-5) + I_1(t-6) + I_1(t-7) + I_1(t-$$

$$slope_{1110111111}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-4) + I_1(t-5) + I_1(t-6) + I_1(t-7) + I_1(t-$$

$$slope_{1101111111}(t) := I_1(t) + I_1(t-1) + I_1(t-3) + I_1(t-4) + I_1(t-5) + I_1(t-6) + I_1(t-7) + I_1(t-$$
$$slope_{1011111111}(t) := I_1(t) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-5) + I_1(t-6) + I_1(t-7) + I_1(t-$$

up to $12-10$ System

$$v_{o11111111110}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-5) + I_0(t-6) + I_0(t-$$
$$v_{o11111111101}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-5) + I_0(t-6) + I_0(t-$$
$$v_{o11111111011}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-5) + I_0(t-6) + I_0(t-$$
$$v_{o11111110111}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-5) + I_0(t-6) + I_0(t-$$
$$v_{o11111101111}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-5) + I_0(t-7) + I_0(t-$$
$$v_{o11111011111}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-6) + I_0(t-7) + I_0(t-$$
$$v_{o11110111111}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-5) + I_0(t-6) + I_0(t-7) + I_0(t-$$
$$v_{o11101111111}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-4) + I_0(t-5) + I_0(t-6) + I_0(t-7) + I_0(t-$$
$$v_{o11011111111}(t) := I_0(t) + I_0(t-1) + I_0(t-3) + I_0(t-4) + I_0(t-5) + I_0(t-6) + I_0(t-7) + I_0(t-$$
$$v_{o10111111111}(t) := I_0(t) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-5) + I_0(t-6) + I_0(t-7) + I_0(t-$$
$$slope_{11111111110}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-5) + I_1(t-6) + I_1(t-$$
$$slope_{11111111101}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-5) + I_1(t-6) + I_1(t-$$
$$slope_{11111111011}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-5) + I_1(t-6) + I_1(t-$$
$$slope_{11111110111}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-5) + I_1(t-6) + I_1(t-$$
$$slope_{11111101111}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-5) + I_1(t-7) + I_1(t-$$
$$slope_{11111011111}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-6) + I_1(t-7) + I_1(t-$$
$$slope_{11110111111}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-5) + I_1(t-6) + I_1(t-7) + I_1(t-$$

$$slope_{11101111111}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-4) + I_1(t-5) + I_1(t-6) + I_1(t-7) + I_1(t-$$
$$slope_{11011111111}(t) := I_1(t) + I_1(t-1) + I_1(t-3) + I_1(t-4) + I_1(t-5) + I_1(t-6) + I_1(t-7) + I_1(t-$$
$$slope_{10111111111}(t) := I_1(t) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-5) + I_1(t-6) + I_1(t-7) + I_1(t-$$

up to $12-11$ System

$$v_{o111111111110}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-5) + I_0(t-6) + I_0(t-$$
$$v_{o111111111101}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-5) + I_0(t-6) + I_0(t-$$
$$v_{o111111111011}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-5) + I_0(t-6) + I_0(t-$$
$$v_{o111111110111}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-5) + I_0(t-6) + I_0(t-$$
$$v_{o111111101111}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-5) + I_0(t-6) + I_0(t-$$
$$v_{o111111011111}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-5) + I_0(t-7) + I_0(t-$$
$$v_{o111110111111}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-6) + I_0(t-7) + I_0(t-$$
$$v_{o111101111111}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-5) + I_0(t-6) + I_0(t-7) + I_0(t-$$
$$v_{o111011111111}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-4) + I_0(t-5) + I_0(t-6) + I_0(t-7) + I_0(t-$$
$$v_{o110111111111}(t) := I_0(t) + I_0(t-1) + I_0(t-3) + I_0(t-4) + I_0(t-5) + I_0(t-6) + I_0(t-7) + I_0(t-$$
$$v_{o101111111111}(t) := I_0(t) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-5) + I_0(t-6) + I_0(t-7) + I_0(t-$$
$$slope_{11111111110}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-5) + I_1(t-6) + I_1(t$$
$$slope_{11111111101}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-5) + I_1(t-6) + I_1(t$$
$$slope_{11111111011}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-5) + I_1(t-6) + I_1(t$$
$$slope_{11111110111}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-5) + I_1(t-6) + I_1(t$$
$$slope_{11111101111}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-5) + I_1(t-6) + I_1(t$$
$$slope_{11111011111}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-5) + I_1(t-7) + I_1(t$$
$$slope_{11110111111}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-6) + I_1(t-7) + I_1(t$$
$$slope_{11101111111}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-5) + I_1(t-6) + I_1(t-7) + I_1(t$$
$$slope_{11011111111}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-4) + I_1(t-5) + I_1(t-6) + I_1(t-7) + I_1(t$$
$$slope_{10111111111}(t) := I_1(t) + I_1(t-1) + I_1(t-3) + I_1(t-4) + I_1(t-5) + I_1(t-6) + I_1(t-7) + I_1(t$$
$$slope_{01111111111}(t) := I_1(t) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-5) + I_1(t-6) + I_1(t-7) + I_1(t$$

only 12 – 11 System

$$v_{o111111111110}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-5) + I_0(t-6) + I_0(t-$$

$$v_{o111111111101}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-5) + I_0(t-6) + I_0(t-$$

$$v_{o111111111011}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-5) + I_0(t-6) + I_0(t-$$

$$v_{o111111110111}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-5) + I_0(t-6) + I_0(t-$$

$$v_{o111111101111}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-5) + I_0(t-6) + I_0(t-$$

$$v_{o111111011111}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-5) + I_0(t-6) + I_0(t-$$

$$v_{o111110111111}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-5) + I_0(t-7) + I_0(t-$$

$$v_{o111101111111}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-6) + I_0(t-7) + I_0(t-$$

$$v_{o111011111111}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-3) + I_0(t-5) + I_0(t-6) + I_0(t-7) + I_0(t-$$

$$v_{o110111111111}(t) := I_0(t) + I_0(t-1) + I_0(t-2) + I_0(t-4) + I_0(t-5) + I_0(t-6) + I_0(t-7) + I_0(t-$$

$$v_{o101111111111}(t) := I_0(t) + I_0(t-1) + I_0(t-3) + I_0(t-4) + I_0(t-5) + I_0(t-6) + I_0(t-7) + I_0(t-$$

$$v_{o011111111111}(t) := I_0(t) + I_0(t-2) + I_0(t-3) + I_0(t-4) + I_0(t-5) + I_0(t-6) + I_0(t-7) + I_0(t-$$

$$slope_{111111111110}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-5) + I_1(t-6) + I_1(t$$

$$slope_{111111111101}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-5) + I_1(t-6) + I_1(t$$

$$slope_{111111111011}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-5) + I_1(t-6) + I_1(t$$

$$slope_{111111110111}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-5) + I_1(t-6) + I_1(t$$

$$slope_{111111101111}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-5) + I_1(t-6) + I_1(t$$

$$slope_{111111011111}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-5) + I_1(t-6) + I_1(t$$

$$slope_{111110111111}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-5) + I_1(t-7) + I_1(t$$

$$slope_{111101111111}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-6) + I_1(t-7) + I_1(t$$

$$slope_{111011111111}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-3) + I_1(t-5) + I_1(t-6) + I_1(t-7) + I_1(t$$

$$slope_{110111111111}(t) := I_1(t) + I_1(t-1) + I_1(t-2) + I_1(t-4) + I_1(t-5) + I_1(t-6) + I_1(t-7) + I_1(t$$

$$slope_{101111111111}(t) := I_1(t) + I_1(t-1) + I_1(t-3) + I_1(t-4) + I_1(t-5) + I_1(t-6) + I_1(t-7) + I_1(t$$

$$slope_{011111111111}(t) := I_1(t) + I_1(t-2) + I_1(t-3) + I_1(t-4) + I_1(t-5) + I_1(t-6) + I_1(t-7) + I_1(t$$

Sequence isolated pulse

peak value

$t_{po} := root\left(slope_o(t_{pk}) \cdot T_s^2, t_{pk}\right)$

$v_{po} := v_o(t_{po})$ $\quad t_{pk} = 0.042$ $\quad v_o(0.03) = 6.402 \times 10^9$

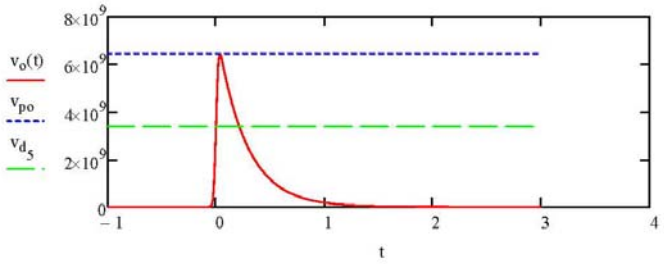
$v_{po} = 6.438 \times 10^9$ $\quad t_{po} = 0.035$ $\quad v_o(0.003) = 4.043 \times 10^9$

decision time

$t_{do_i} := root\left[\left(I_0(t_d) - v_i \cdot I_0(t_{po})\right) \cdot T_s^2, t_d\right]$

$v_{d_i} := I_0\left(t_{do_i}\right)$ $\quad t_{do_5} = -1.278 \times t_d = -0.012$

t_{do_i} =	$v_5 \cdot I_0(t_{po}) = 3.377 \times 10^9$	v_i =	v_{d_i} =	$\frac{v_{d_i}}{v_{po}}$ =
$-1.483 \cdot 10^{-3}$		0.519	$3.345 \cdot 10^9$	0.519
$-1.442 \cdot 10^{-3}$	$v_{d_5} = 3.377 \times 10^9$	0.521	$3.351 \cdot 10^9$	0.52
$-1.401 \cdot 10^{-3}$		0.521	$3.358 \cdot 10^9$	0.521
$-1.36 \cdot 10^{-3}$		0.522	$3.364 \cdot 10^9$	0.522
$-1.319 \cdot 10^{-3}$	$v_{po} = 6.438 \times 10^9$	0.523	$3.371 \cdot 10^9$	0.523
$-1.278 \cdot 10^{-3}$		0.524	$3.377 \cdot 10^9$	0.524
$-1.237 \cdot 10^{-3}$		0.525	$3.383 \cdot 10^9$	0.525
$-1.196 \cdot 10^{-3}$		0.526	$3.39 \cdot 10^9$	0.526
$-1.155 \cdot 10^{-3}$		0.527	$3.396 \cdot 10^9$	0.527
$-1.114 \cdot 10^{-3}$	$t := -1, -0.99 .. 3$	0.529	$3.403 \cdot 10^9$	0.528



Sequence 10

peak value

Same as an isolated pulse

$t_{p10} := \text{root}\left(\text{slope}_{10}(t_{pk})\cdot T_s^{\,2}, t_{pk}\right)$

$t_{p10} = 0.035 \qquad t_{pk} = 0.042$

$v_{p10} := v_{o10}(t_{p10})$

$v_{p10} = 6.438 \times 10^{9}$

decision time

$t_{d10_i} := \text{root}\left[\left(v_{o10}(t_d) - v_{d_i}\right)\cdot T_s^{\,2}, t_d\right] \qquad v_{o10}\left(t_{d10_5}\right) = 3.377 \times 10^{9}$

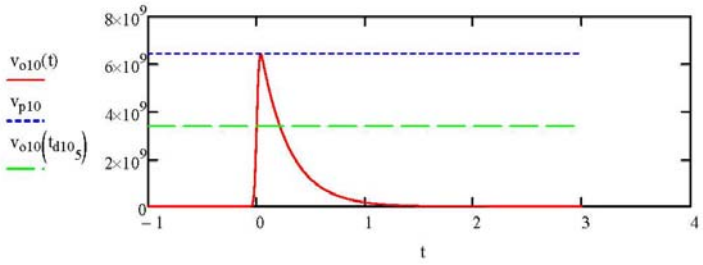
$t_{do_i} =$

$-1.483\cdot10^{-3}$
$-1.442\cdot10^{-3}$
$-1.401\cdot10^{-3}$
$-1.36\cdot10^{-3}$
$-1.319\cdot10^{-3}$
$-1.278\cdot10^{-3}$
$-1.237\cdot10^{-3}$
$-1.196\cdot10^{-3}$
$-1.155\cdot10^{-3}$
$-1.114\cdot10^{-3}$

$\frac{v_{d_i}}{v_{p10}} =$

0.519
0.52
0.521
0.522
0.523
0.524
0.525
0.526
0.527
0.528

$t := -1, -0.99 .. 3$



Sequence 110

CAUTION

peak value

First Pulse

$t_{p11} := \text{root}\left(\text{slope}_{110}(t_{pk})\cdot T_s^{\,2}, t_{pk}\right)$ $t_{pk} = 0.042$

$v_{p11} := v_{o110}(t_{p11})$

$v_{p11} = 6.438 \times 10^9$ $t_{p11} = 0.035$

Second Pulse

$t_{pk1} := t_{pk} + 1$

$t_{p110} := \text{root}\left(\text{slope}_{110}(t_{pk1})\cdot T_s^{\,2}, t_{pk1}\right)$

$v_{p110} := v_{o110}(t_{p110})$ $t_{p110} = 1.035$

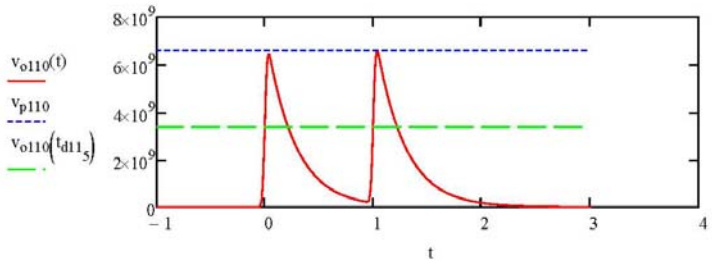
$v_{p110} = 6.591 \times 10^9$

decision time

$$t_{d11_i} := \text{root}\left[\left(v_{o110}(t_d) - v_{d_i}\right)\cdot T_s^{\,2}, t_d\right]$$

t_{d11_i} =	t_{do_i} =	$t_{d11_i} + 1$ =	$\frac{v_{d_i}}{v_{p11}}$ =	$\frac{v_{d_i}}{v_{p110}}$ =
$-1.483\cdot10^{-3}$	$-1.483\cdot10^{-3}$	0.999	0.519	0.507
$-1.442\cdot10^{-3}$	$-1.442\cdot10^{-3}$	0.999	0.52	0.508
$-1.401\cdot10^{-3}$	$-1.401\cdot10^{-3}$	0.999	0.521	0.509
$-1.36\cdot10^{-3}$	$-1.36\cdot10^{-3}$	0.999	0.522	0.51
$-1.319\cdot10^{-3}$	$-1.319\cdot10^{-3}$	0.999	0.523	0.511
$-1.278\cdot10^{-3}$	$-1.278\cdot10^{-3}$	0.999	0.524	0.512
$-1.237\cdot10^{-3}$	$-1.237\cdot10^{-3}$	0.999	0.525	0.513
$-1.196\cdot10^{-3}$	$-1.196\cdot10^{-3}$	0.999	0.526	0.514
$-1.155\cdot10^{-3}$	$-1.155\cdot10^{-3}$	0.999	0.527	0.515
$-1.114\cdot10^{-3}$	$-1.114\cdot10^{-3}$	0.999	0.528	0.516

$t := -1, -0.99 .. 3$



Sequence 1110

CAUTION

peak value

First Pulse

$t_{p111} := \text{root}\left(\text{slope}_{1110}(t_{pk})\cdot T_s^{\,2}, t_{pk}\right)$ $t_{pk} = 0.042$

$v_{p111} := v_{o1110}(t_{p111})$

$v_{p111} = 6.438 \times 10^9$ $t_{p111} = 0.035$

Second Pulse

$t_{pk1} := t_{pk} + 1$

$t_{p1112} := root\left(slope_{1110}(t_{pk1})\cdot T_s^{\,2}, t_{pk1}\right)$

$v_{p1112} := v_{o1110}(t_{p1112})$ $\quad t_{p1112} = 1.035$

$v_{p1112} = 6.591 \times 10^{9}$

Third Pulse

$t_{pk2} := t_{pk} + 2$

$t_{p1110} := root\left(slope_{1110}(t_{pk2})\cdot T_s^{\,2}, t_{pk2}\right)$

$v_{p1110} := v_{o1110}(t_{p1110})$ $\quad t_{p1110} = 2.035$

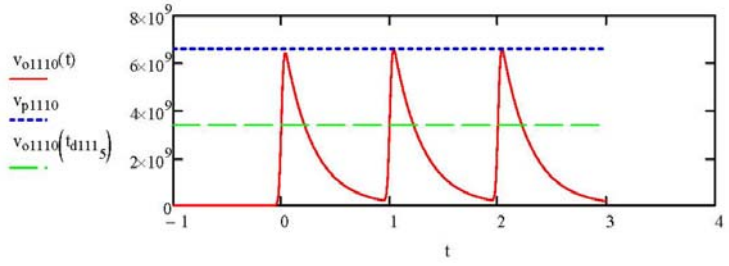
$v_{p1110} = 6.595 \times 10^{9}$

decision time

$t_{d111_i} := root\left[\left(v_{o1110}(t_d) - v_{d_i}\right)\cdot T_s^{\,2}, t_d\right]$

$t_{d111_i} =$	$t_{do_i} =$	$t_{d111_i} + 1 =$	$t_{d111_i} + 2 =$	$\frac{v_{d_i}}{v_{p111}} =$	$\frac{v_{d_i}}{v_{p1110}} =$
$-1.483\cdot10^{-3}$	$-1.483\cdot10^{-3}$	0.999	1.999	0.519	0.507
$-1.442\cdot10^{-3}$	$-1.442\cdot10^{-3}$	0.999	1.999	0.52	0.508
$-1.401\cdot10^{-3}$	$-1.401\cdot10^{-3}$	0.999	1.999	0.521	0.509
$-1.36\cdot10^{-3}$	$-1.36\cdot10^{-3}$	0.999	1.999	0.522	0.51
$-1.319\cdot10^{-3}$	$-1.319\cdot10^{-3}$	0.999	1.999	0.523	0.511
$-1.278\cdot10^{-3}$	$-1.278\cdot10^{-3}$	0.999	1.999	0.524	0.512
$-1.237\cdot10^{-3}$	$-1.237\cdot10^{-3}$	0.999	1.999	0.525	0.513
$-1.196\cdot10^{-3}$	$-1.196\cdot10^{-3}$	0.999	1.999	0.526	0.514
$-1.155\cdot10^{-3}$	$-1.155\cdot10^{-3}$	0.999	1.999	0.527	0.515
$-1.114\cdot10^{-3}$	$-1.114\cdot10^{-3}$	0.999	1.999	0.528	0.516

$t := -1, -0.99 .. 3$



Sequence 101

peak value

Same as isolated pulse and 10 sequence

$t_{p101} := \text{root}\left(\text{slope}_{101}(t_{pk})\cdot T_s^{\,2}, t_{pk}\right)$

$t_{p101} = 0.035$

$v_{p101} := v_{o101}(t_{p101})$

$v_{p101} = 6.438 \times 10^9$

Second Pulse

$t_{pk66} := t_{pk} + 2$

$t_{p1010} := \text{root}\left(\text{slope}_{101}(t_{pk66})\cdot T_s^{\,2}, t_{pk66}\right)$

$v_{p1010} := v_{o101}(t_{p1010})$

$v_{p1010} = 6.442 \times 10^9$

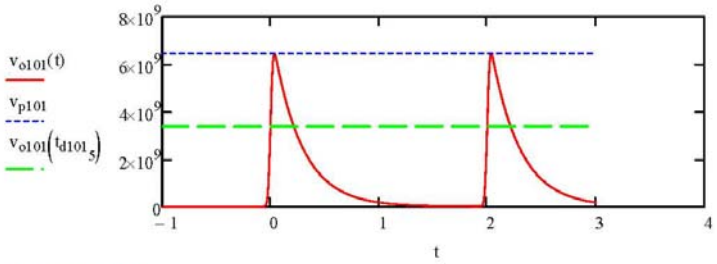
$t_{p1010} = 2.035$

decision time

$$t_{d101_i} := \text{root}\left[\left(v_{o101}(t_d) - v_{d_i}\right)\cdot T_s^{\,2}, t_d\right]$$

$\frac{v_{d_i}}{v_{p101}} =$	$\frac{v_{d_i}}{v_{p1010}} =$
0.519	0.519
0.52	0.52
0.521	0.521
0.522	0.522
0.523	0.523
0.524	0.524
0.525	0.525
0.526	0.526
0.527	0.527
0.528	0.528

$t := -1, -0.99 .. 3$



Sequence 1101

peak value

First Pulse

$$t_{p1101} := \text{root}\left(\text{slope}_{1101}(t_{pk})\cdot T_s^{\,2}, t_{pk}\right)$$

$$v_{p1101} := v_{o1101}(t_{p1101})$$

$$v_{p1101} = 6.438 \times 10^{9}$$

Second Pulse

$t_{pk67} := t_{pk} + 1$

$t_{p1101\_2} := root\left(slope_{1101}(t_{pk67}) \cdot T_s^{\,2}, t_{pk67}\right)$

$v_{p1101\_2} := v_{o1101}(t_{p1101\_2})$

$v_{p1101\_2} = 6.591 \times 10^{9}$

First pulse

Third Pulse

$t_{pk68} := t_{pk} + 3$

$t_{p11010} := root\left(slope_{1101}(t_{pk68}) \cdot T_s^{\,2}, t_{pk68}\right)$

$v_{p11010} := v_{o1101}(t_{p11010})$

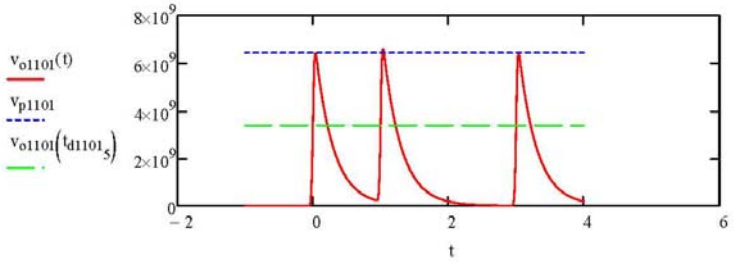
$v_{p11010} = 6.442 \times 10^{9}$

decision time

$t_{d1101_i} := root\left[\left(v_{o1101}(t_d) - v_{d_i}\right) \cdot T_s^{\,2}, t_d\right]$

$t := -1, -0.99 .. 4$

$\frac{v_{d_i}}{v_{p1101}} =$	$\frac{v_{d_i}}{v_{p1101\_2}} =$	$\frac{v_{d_i}}{v_{p11010}} =$
0.519	0.507	0.519
0.52	0.508	0.52
0.521	0.509	0.521
0.522	0.51	0.522
0.523	0.511	0.523
0.524	0.512	0.524
0.525	0.513	0.525
0.526	0.514	0.526
0.527	0.515	0.527
0.528	0.516	0.528



Sequence 1011

peak value

For the first pulse same as 10 sequence

$t_{p1011} := \text{root}\left(\text{slope}_{1011}(t_{pk})\cdot T_s^{\,2}, t_{pk}\right)$

$t_{p1011} = 0.035 \qquad t_{pk} = 0.042$

$v_{o1011}(t_{p1011}) = 6.438 \times 10^{9}$

$v_{p1011} := v_{o1011}(t_{p1011})$

$v_{p1011} = 6.438 \times 10^{9}$

For the second pulse same as 101 sequence

$t_{pk121} := t_{pk} + 2$

$t_{p1011\_2} := \text{root}\left(\text{slope}_{1011}(t_{pk121})\cdot T_s^{\,2}, t_{pk121}\right)$

$t_{p1011\_2} = 2.035$

$v_{o1011}(t_{p1011\_2}) = 6.442 \times 10^{9}$

$v_{p1011\_2} := v_{o1011}(t_{p1011\_2})$

$v_{p1011\_2} = 6.442 \times 10^{9}$

For the third pulse same as 110 sequence

$t_{pk122} := t_{pk} + 3$

$t_{p10110} := \text{root}\left(\text{slope}_{1011}(t_{pk122})\cdot T_s^{\,2}, t_{pk122}\right)$

$t_{p10110} = 3.035$

$v_{o1011}(t_{p10110}) = 6.591 \times 10^{9}$

$v_{p10110} := v_{o1011}(t_{p10110})$

$v_{p10110} = 6.591 \times 10^{9}$

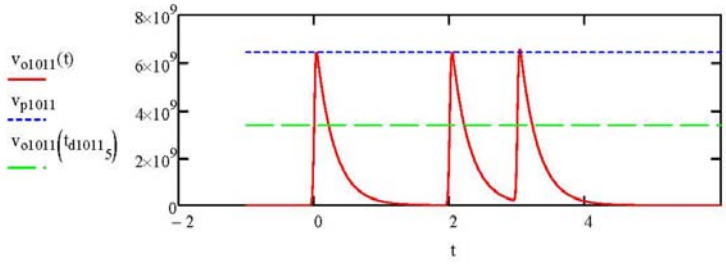
decision time

First Pulse

$$t_{d1011_i} := \text{root}\left[\left(v_{o1011}(t_d) - v_{d_i}\right)\cdot T_s^{\,2}, t_d\right]$$

$\frac{v_{d_i}}{v_{p1011}} =$	$\frac{v_{d_i}}{v_{p1011\_2}} =$	$\frac{v_{d_i}}{v_{p10110}} =$
0.519	0.519	0.507
0.52	0.52	0.508
0.521	0.521	0.509
0.522	0.522	0.51
0.523	0.523	0.511
0.524	0.524	0.512
0.525	0.525	0.513
0.526	0.526	0.514
0.527	0.527	0.515
0.528	0.528	0.516

$t := -1, -0.99 .. 6$



Sequence 11011

peak value

First Pulse

$$t_{p11011} := \text{root}\left(\text{slope}_{11011}(t_{pk})\cdot T_s^{\,2}, t_{pk}\right)$$

$v_{p11011} := v_{o11011}(t_{p11011})$ $\quad t_{p11011} = 0.035$

$v_{p11011} = 6.438 \times 10^{9}$

Second Pulse

$t_{pk123} := t_{pk} + 1$

$t_{p11011\_2} := \text{root}\left(\text{slope}_{11011}(t_{pk123})\cdot T_s^{\,2}, t_{pk123}\right)$

$v_{p11011\_2} := v_{o11011}(t_{p11011\_2})$

$v_{p11011\_2} = 6.591 \times 10^{9} \quad t_{p11011\_2} = 1.035$

Third Pulse

$t_{pk124} := t_{pk} + 3$

$t_{p11011\_3} := \text{root}\left(\text{slope}_{11011}(t_{pk124})\cdot T_s^{\,2}, t_{pk124}\right)$

$v_{p11011\_3} := v_{o11011}(t_{p11011\_3})$

$v_{p11011\_3} = 6.442 \times 10^{9}$

Fourth Pulse

$t_{pk125} := t_{pk} + 4$

$t_{p110110} := \text{root}\left(\text{slope}_{11011}(t_{pk125})\cdot T_s^{\,2}, t_{pk125}\right)$

$v_{p110110} := v_{o11011}(t_{p110110})$

$v_{p110110} = 6.591 \times 10^{9}$

decision time

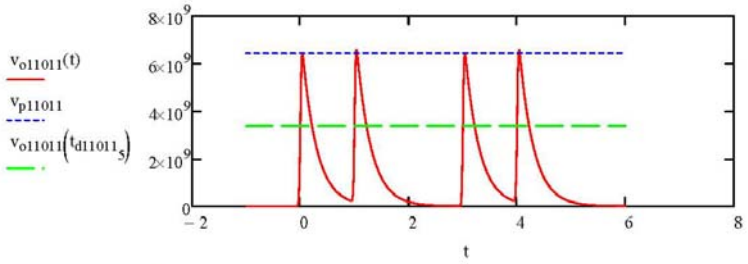
First pulse

$t_{d11011_i} := \text{root}\left[\left(v_{o11011}(t_d) - v_{d_i}\right)\cdot T_s^{\,2}, t_d\right]$

$\frac{v_{d_i}}{v_{p11011}} =$	$\frac{v_{d_i}}{v_{p11011\_2}} =$	$\frac{v_{d_i}}{v_{p11011\_3}} =$	$\frac{v_{d_i}}{v_{p110110}} =$
0.519	0.507	0.519	0.507

0.52	0.508	0.52	0.508
0.521	0.509	0.521	0.509
0.522	0.51	0.522	0.51
0.523	0.511	0.523	0.511
0.524	0.512	0.524	0.512
0.525	0.513	0.525	0.513
0.526	0.514	0.526	0.514
0.527	0.515	0.527	0.515
0.528	0.516	0.528	0.516

$t := -1, -0.99 .. 6$



Sequence 10111

peak value

For the first pulse same as 10 sequence

$t_{p10111} := \text{root}\left(\text{slope}_{10111}(t_{pk})\cdot T_s^{\,2}, t_{pk}\right)$

$t_{p10111} = 0.035 \qquad t_{pk} = 0.042$

$v_{o10111}(t_{p10111}) = 6.438 \times 10^9$

$v_{p10111} := v_{o10111}(t_{p10111})$

For the second pulse same as 101 sequence

$t_{pk186} := t_{pk} + 2$

$t_{p10111\_2} := \text{root}\left(\text{slope}_{10111}(t_{pk186})\cdot T_s^{\,2}, t_{pk186}\right)$

$t_{p10111\_2} = 2.035$

$v_{o10111}(t_{p10111\_2}) = 6.442 \times 10^9$

$v_{p10111\_2} := v_{o10111}(t_{p10111\_2})$

$v_{p10111\_2} = 6.442 \times 10^9$

For the third pulse same as 110 sequence

$t_{pk187} := t_{pk} + 3$

$t_{p10111\_3} := \text{root}\left(\text{slope}_{10111}(t_{pk187}) \cdot T_s^{\,2}, t_{pk187}\right)$

$t_{p10111\_3} = 3.035$

$v_{o10111}(t_{p10111\_3}) = 6.591 \times 10^9$

$v_{p10111\_3} := v_{o10111}(t_{p10111\_3})$

$v_{p10111\_3} = 6.591 \times 10^9$

For the third pulse same as 110 sequence

$t_{pk188} := t_{pk} + 4$

$t_{p101110} := \text{root}\left(\text{slope}_{10111}(t_{pk188}) \cdot T_s^{\,2}, t_{pk188}\right)$

$t_{p101110} = 4.035$

$v_{o10111}(t_{p101110}) = 6.595 \times 10^9$

$v_{p101110} := v_{o10111}(t_{p101110})$

$v_{p101110} = 6.595 \times 10^9$

$\frac{v_{d_i}}{v_{p10111}} =$	$\frac{v_{d_i}}{v_{p10111\_2}} =$	$\frac{v_{d_i}}{v_{p10111\_3}}$
0.519	0.519	0.507
0.52	0.52	0.508
0.521	0.521	0.509
0.522	0.522	0.51
0.523	0.523	0.511
0.524	0.524	0.512
0.525	0.525	0.513
0.526	0.526	0.514
0.527	0.527	0.515
0.528	0.528	0.516

decision time

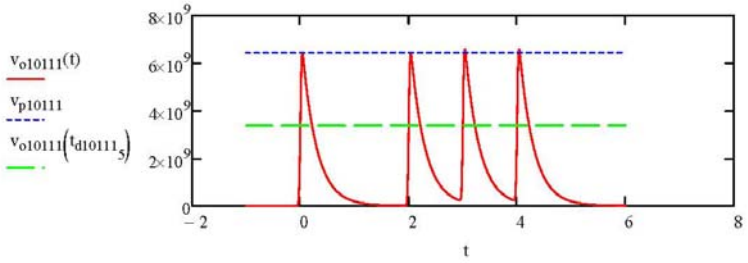
First Pulse

$t_{d10111_i} := \text{root}\left[\left(v_{o10111}(t_d) - v_{d_i}\right) \cdot T_s^{\,2}, t_d\right]$

$\frac{v_{d_i}}{v_{p101110}} =$
0.507
0.508
0.509
0.51
0.511
0.512

$t := -1, -0.99 .. 6$

0.513
0.514
0.515
0.516



Sequence 110111

peak value

For the first pulse same as 10 sequence

$t_{p110111} := \text{root}\left(\text{slope}_{110111}(t_{pk})\cdot T_s^{\,2}, t_{pk}\right)$

$t_{p110111} = 0.035 \quad t_{pk} = 0.042$

$v_{o110111}(t_{p110111}) = 6.438 \times 10^9$

$v_{p110111} := v_{o110111}(t_{p110111}) \quad v_{p110111} = 6.438 \times 10^9$

For the second pulse same as 101 sequence

$t_{pk189} := t_{pk} + 1$

$t_{p110111\_2} := \text{root}\left(\text{slope}_{110111}(t_{pk189})\cdot T_s^{\,2}, t_{pk189}\right)$

$t_{p110111\_2} = 1.035$

$v_{o110111}(t_{p110111\_2}) = 6.591 \times 10^9$

$v_{p110111\_2} := v_{o110111}(t_{p110111\_2})$

$v_{p110111\_2} = 6.591 \times 10^{9}$

For the third pulse same as 110 sequence

$t_{pk190} := t_{pk} + 3$

$t_{p110111\_3} := root\left(slope_{110111}(t_{pk190}) \cdot T_s^{\,2}, t_{pk190}\right)$

$t_{p110111\_3} = 3.035$

$v_{o110111}(t_{p110111\_3}) = 6.442 \times 10^{9}$

$v_{p110111\_3} := v_{o110111}(t_{p110111\_3})$

$v_{p110111\_3} = 6.442 \times 10^{9}$

For the third pulse same as 110 sequence

$t_{pk191} := t_{pk} + 4$

$t_{p110111\_4} := root\left(slope_{110111}(t_{pk191}) \cdot T_s^{\,2}, t_{pk191}\right)$

$t_{p110111\_4} = 4.035$

$v_{o110111}(t_{p110111\_4}) = 6.591 \times 10^{9}$

$v_{p110111\_4} := v_{o110111}(t_{p110111\_4})$

$v_{p110111\_4} = 6.591 \times 10^{9}$

For the third pulse same as 110 sequence

$t_{pk192} := t_{pk} + 5$

$t_{p1101110} := root\left(slope_{110111}(t_{pk192}) \cdot T_s^{\,2}, t_{pk192}\right)$

$t_{p1101110} = 5.035$

$v_{o110111}(t_{p1101110}) = 6.595 \times 10^{9}$

$v_{p1101110} := v_{o110111}(t_{p1101110})$

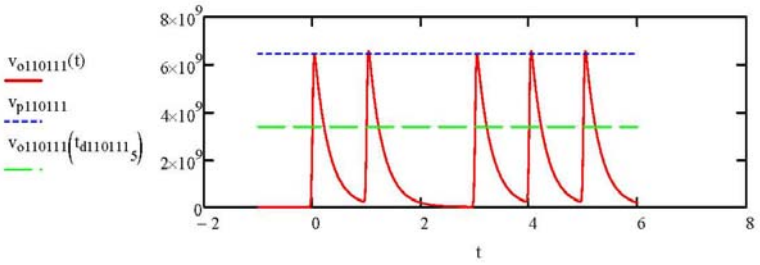
$$v_{p1101110} = 6.595 \times 10^{9}$$

decision time

First Pulse

$$t_{d110111_i} := \text{root}\left[\left(v_{o110111}(t_d) - v_{d_i}\right)\cdot T_s^{\,2}, t_d\right]$$

$$t := -1, -0.99 .. 6$$



Sequence 10101

peak value

For the first pulse same as 10 sequence

$$t_{p10101} := \text{root}\left(\text{slope}_{10101}(t_{pk})\cdot T_s^{\,2}, t_{pk}\right)$$

$$t_{p10101} = 0.035 \qquad t_{pk} = 0.042$$

$$v_{o10101}(t_{p10101}) = 6.438 \times 10^{9}$$

$$v_{p10101} := v_{o10101}(t_{p10101})$$

$$v_{p10101} = 6.438 \times 10^{9}$$

For the second pulse same as 101 sequence

$$t_{pk448} := t_{pk} + 2$$

$$t_{p10101\_2} := \text{root}\left(\text{slope}_{10101}(t_{pk448})\cdot T_s^{\,2}, t_{pk448}\right)$$

$$t_{p10101\_2} = 2.035$$

$v_{o10101}(t_{p10101\_2}) = 6.442 \times 10^9$

$v_{p10101\_2} := v_{o10101}(t_{p10101\_2})$

$v_{p10101\_2} = 6.442 \times 10^9$

For the third pulse same as 110 sequence

$t_{pk449} := t_{pk} + 4$

$t_{p101010} := \text{root}\left(\text{slope}_{1011111111}(t_{pk449}) \cdot T_s^{\,2}, t_{pk449}\right)$

$t_{p101010} = 4.035$

$v_{o10101}(t_{p101010}) = 6.442 \times 10^9$

$v_{p101010} := v_{o10101}(t_{p101010})$

$v_{p101010} = 6.442 \times 10^9$

decision time

First Pulse

$t_{d10101_i} := \text{root}\left[\left(v_{o10101}(t_d) - v_{d_i}\right) \cdot T_s^{\,2}, t_d\right]$

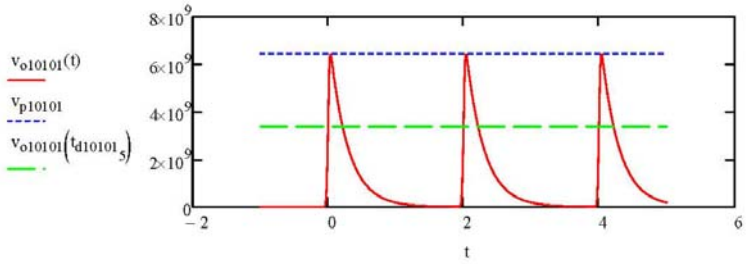
$\frac{v_{d_i}}{v_{p10101}} =$ $\frac{v_{d_i}}{v_{p10101\_2}} =$ $\frac{v_{d_i}}{v_{p101010}} =$

0.519
0.52
0.521
0.522
0.523
0.524
0.525
0.526
0.527
0.528

0.519
0.52
0.521
0.522
0.523
0.524
0.525
0.526
0.527
0.528

0.519
0.52
0.521
0.522
0.523
0.524
0.525
0.526
0.527
0.528

$t := -1, -0.99 .. 5$



ERROR SOURCES

FALSE ALARM ERROR

Equivalent pcm error number is found by error_number/((x-y)*pcm_bits*pcm_words)

Standard false alarm

$$Qf_i := \eta q \cdot \frac{v_{d_i}}{\sqrt{S_o \cdot noise}}$$

$$P_f(b,i) := \frac{T_s}{\tau_R} \cdot \frac{1}{2} \cdot erfc\left(\frac{b \cdot Qf_i}{\sqrt{2}}\right)$$

$$P_{ef}(b,i) := 0.205204 \cdot P_f(b,i)$$

Slot after pulse false alarm - time td+Ts

$$v_{o010_i} := v_{o10}\left(t_{do_i} + 1\right)$$

$$Q_{fl0_i} := \eta q \cdot \frac{v_{d_i} - v_{o010_i}}{\sqrt{S_o \cdot noise}}$$

$$P_{fl0}(b,i) := \frac{T_s}{\tau_R} \cdot \frac{1}{2} \cdot erfc\left(\frac{b \cdot Q_{fl0_i}}{\sqrt{2}}\right)$$

$$P_{efl0}(b,i) := 0.0303914 \cdot P_{fl0}(b,i)$$

Slot after double pulse false alarm - time td+2Ts

$$v_{o0110_i} := v_{o110}\left(t_{do_i} + 2\right)$$

$$Q_{f0110_i} := \eta q \cdot \frac{v_{d_i} - v_{o0110_i}}{\sqrt{S_o \cdot noise}}$$

$$P_{f0110}(b,i) := \frac{T_s}{\tau_R}\cdot\frac{1}{2}\cdot\operatorname{erfc}\left(\frac{b\cdot Q_{f0110_i}}{\sqrt{2}}\right)$$

$$P_{ef0110}(b,i) := 0.00474853\cdot P_{f0110}(b,i)$$

Slot between pulses false alarm - time td+Ts

$$v_{o0101_i} := v_{o101}\left(t_{do_i} + 1\right)$$

$$Q_{f101_i} := \eta q\cdot\frac{v_{d_i} - v_{o0101_i}}{\sqrt{S_o\cdot noise}}$$

$$P_{f101}(b,i) := \frac{T_s}{\tau_R}\cdot\frac{1}{2}\cdot\operatorname{erfc}\left(\frac{b\cdot Q_{f101_i}}{\sqrt{2}}\right)$$

$$P_{ef101}(b,i) := 0.00440267\cdot P_{f101}(b,i)$$

Slot between double pulse and a pulse false alarm - time td+2Ts

$$v_{o01101_i} := v_{o1101}\left(t_{do_i} + 2\right)$$

$$Q_{f1101_i} := \eta q \cdot \frac{v_{d_i} - v_{o0110_i}}{\sqrt{S_o \cdot noise}}$$

$$P_{f1101}(b,i) := \frac{T_s}{\tau_R} \cdot \frac{1}{2} \cdot erfc\left(\frac{b \cdot Q_{f1101_i}}{\sqrt{2}}\right)$$

$$P_{ef1101}(b,i) := 4.47591e\text{-}005 \cdot P_{f1101}(b,i)$$

TOTAL FALSE ALARM ERROR

$$P_{ftot}(b,i) := P_{ef}(b,i) + P_{ef10}(b,i) + P_{ef0110}(b,i) + P_{ef101}(b,i) + P_{ef1101}(b,i)$$

$P_{ftot}(b,i) =$

$6.707 \cdot 10^{-10}$
$6.163 \cdot 10^{-10}$
$5.662 \cdot 10^{-10}$
$5.201 \cdot 10^{-10}$
$4.777 \cdot 10^{-10}$
$4.386 \cdot 10^{-10}$
$4.027 \cdot 10^{-10}$
$3.697 \cdot 10^{-10}$
$3.393 \cdot 10^{-10}$
$3.114 \cdot 10^{-10}$

WRONG SLOT ERROR

Equivalent pcm error number is found by error_number/(pcm_bits*pcm_words)

Standard wrong slot

$slope_i := slope_o\left(t_{do_i}\right)$

$$Q_{s_i} := \eta q \cdot \left(\frac{1}{2}\right) \cdot \frac{slope_i}{\sqrt{S_o \cdot noise}}$$

$$P_s(b,i) := \frac{1}{2} \cdot erfc\left(\frac{Q_{s_i} \cdot b}{\sqrt{2}}\right)$$

$$P_{es}(b,i) := 0.844279 \cdot P_s(b,i)$$

Wrong slot on first pulse of two

$slope_{011_i} := slope_{110}\left(t_{do_i}\right)$

$$Q_{s11_i} := \eta q \cdot \left(\frac{1}{2}\right) \cdot \frac{slope_{011_i}}{\sqrt{S_o \cdot noise}}$$

$$P_{s11}(b,i) := \frac{1}{2} \cdot erfc\left(\frac{Q_{s11_i} \cdot b}{\sqrt{2}}\right)$$

$$P_{es11}(b,i) := 0.0996094 \cdot P_{s11}(b,i)$$

Wrong slot on second pulse of two CAUTION

$$slope_{0110_i} := slope_{110}\left(t_{do_i} + 1\right)$$

$$Q_{s110_i} := \eta q \cdot \left(\frac{1}{2}\right) \cdot \frac{slope_{0110_i}}{\sqrt{S_o \cdot noise}}$$

$$P_{s110}(b,i) := \frac{1}{2} \cdot erfc\left(\frac{Q_{s110_i} \cdot b}{\sqrt{2}}\right)$$

$$P_{es110}(b,i) := 0.0478516 \cdot P_{s110}(b,i)$$

Wrong slot on third pulse of three

$$slope_{01110_i} := slope_{1110}\left(t_{do_i} + 2\right)$$

$$Q_{s1110_i} := \eta q \cdot \left(\frac{1}{2}\right) \cdot \frac{slope_{01110_i}}{\sqrt{S_o \cdot noise}}$$

$$P_{s1110}(b,i) := \frac{1}{2} \cdot erfc\left(\frac{Q_{s1110_i} \cdot b}{\sqrt{2}}\right)$$

$$P_{es1110}(b,i) := 0.00146484 \cdot P_{s1110}(b,i)$$

Wrong slot on second pulse of 101

$$slope_{0101_i} := slope_{101}\left(t_{do_i} + 2\right)$$

$$Q_{s101_i} := \eta q \cdot \left(\frac{1}{2}\right) \cdot \frac{slope_{0101_i}}{\sqrt{S_o \cdot noise}}$$

$$P_{s101}(b,i) := \frac{1}{2} \cdot erfc\left(\frac{Q_{s101_i} \cdot b}{\sqrt{2}}\right)$$

$$P_{es101}(b,i) := 0.0801188 \cdot P_{s101}(b,i)$$

Wrong slot on third pulse of 1101

$$slope_{01101_i} := slope_{1101}\left(t_{do_i} + 3\right)$$

$$Q_{s1101_i} := \eta q \cdot \left(\frac{1}{2}\right) \cdot \frac{slope_{01101_i}}{\sqrt{S_o \cdot noise}}$$

$$P_{s1101}(b,i) := \frac{1}{2} \cdot erfc\left(\frac{Q_{s1101_i} \cdot b}{\sqrt{2}}\right)$$

$$P_{es1101}(b,i) := 0.000732422 \cdot P_{s1101}(b,i)$$

$$slope_{01011_i} := slope_{1011}\left(t_{do_i} + 2\right)$$

$$Q_{s1011_i} := \eta q \cdot \left(\frac{1}{2}\right) \cdot \frac{slope_{01011_i}}{\sqrt{S_o \cdot noise}}$$

$$P_{s1011}(b,i) := \frac{1}{2} \cdot erfc\left(\frac{Q_{s1011_i} \cdot b}{\sqrt{2}}\right)$$

$$P_{es1011}(b,i) := 0.00146484 \cdot P_{s1011}(b,i)$$

$$slope_{010111_i} := slope_{10111}\left(t_{do_i} + 2\right)$$

$$Q_{s10111_i} := \eta q \cdot \left(\frac{1}{2}\right) \cdot \frac{slope_{010111_i}}{\sqrt{S_o \cdot noise}}$$

$$P_{s10111}(b,i) := \frac{1}{2} \cdot erfc\left(\frac{Q_{s10111_i} \cdot b}{\sqrt{2}}\right)$$

$$P_{es10111}(b,i) := 0 \cdot P_{s10111}(b,i)$$

$$slope_{011011_i} := slope_{11011}\left(t_{do_i} + 3\right)$$

$$Q_{s11011_i} := \eta q \cdot \left(\frac{1}{2}\right) \cdot \frac{slope_{011011_i}}{\sqrt{S_o \cdot noise}}$$

$$P_{s11011}(b,i) := \frac{1}{2} \cdot erfc\left(\frac{Q_{s11011_i} \cdot b}{\sqrt{2}}\right)$$

$$P_{es11011}(b,i) := 0 \cdot P_{s11011}(b,i)$$

$$slope_{0110111_i} := slope_{110111}\left(t_{do_i} + 3\right)$$

$$Q_{s110111_i} := \eta q \cdot \left(\frac{1}{2}\right) \cdot \frac{slope_{0110111_i}}{\sqrt{S_o \cdot noise}}$$

$$P_{s110111}(b,i) := \frac{1}{2} \cdot erfc\left(\frac{Q_{s110111_i} \cdot b}{\sqrt{2}}\right)$$

$$P_{es110111}(b,i) := 0 \cdot P_{s110111}(b,i)$$

Wrong slot on second pulse of three 111

$$slope_{0111_i} := slope_{1110}\left(t_{do_i} + 1\right)$$

$$Q_{s1\_1\_1_i} := \eta q \cdot \left(\frac{1}{2}\right) \cdot \frac{slope_{0111_i}}{\sqrt{S_o \cdot noise}}$$

$$P_{s1\_1\_1}(b,i) := \frac{1}{2}\cdot\mathrm{erfc}\left(\frac{Q_{s1\_1\_1_i}\cdot b}{\sqrt{2}}\right)$$

$$P_{es1\_1\_1}(b,i) := 0\cdot P_{s1\_1\_1}(b,i)$$

Wrong slot on second pulse of four 1111

$$slope_{01111_i} := slope_{11110}\left(t_{do_i} + 1\right)$$

$$Q_{s1\_1\_11_i} := \eta q\cdot\left(\frac{1}{2}\right)\cdot\frac{slope_{01111_i}}{\sqrt{S_o\cdot noise}}$$

$$P_{s1\_1\_11}(b,i) := \frac{1}{2}\cdot\mathrm{erfc}\left(\frac{Q_{s1\_1\_11_i}\cdot b}{\sqrt{2}}\right)$$

$$P_{es1\_1\_11}(b,i) := 0\cdot P_{s1\_1\_11}(b,i)$$

Wrong slot on second pulse of four 1111

$$slope_{01111_i} := slope_{11110}\left(t_{do_i} + 2\right)$$

$$Q_{s11\_1\_1_i} := \eta q\cdot\left(\frac{1}{2}\right)\cdot\frac{slope_{01111_i}}{\sqrt{S_o\cdot noise}}$$

$$P_{s11\_1\_1}(b,i) := \frac{1}{2}\cdot\mathrm{erfc}\left(\frac{Q_{s11\_1\_1_i}\cdot b}{\sqrt{2}}\right)$$

$$P_{es11\_1\_1}(b,i) := 0\cdot P_{s11\_1\_1}(b,i)$$

Wrong slot on second pulse of five 11111

$$slope_{011111_i} := slope_{111110}\left(t_{do_i} + 2\right)$$

$$Q_{s11\_1\_11_i} := \eta q\cdot\left(\frac{1}{2}\right)\cdot\frac{slope_{011111_i}}{\sqrt{S_o\cdot noise}}$$

$$P_{s11\_1\_11}(b,i) := \frac{1}{2}\cdot\mathrm{erfc}\left(\frac{Q_{s11\_1\_11_i}\cdot b}{\sqrt{2}}\right)$$

$$P_{es11\_1\_11}(b,i) := 0\cdot P_{s11\_1\_11}(b,i)$$

TOTAL WRONG SLOT ERROR

$P_{cws}(b,i) := P_{cs}(b,i) + P_{cs11}(b,i) + P_{cs110}(b,i) + P_{cs1110}(b,i) + P_{cs101}(b,i) + P_{cs1101}(b,i) + P_{cs1011}(b,i$

$P_{cws}(b,i) =$

0
0
0
0
0
0
0
0
0
0

ERASURES

Equivalent pcm error number is found by error_number/(pcm_bits*pcm_words)

Standard erasure of pulse

modify pulse parameters

$t_{po_i} := t_{do_i} + 0.5$

$t_{po} := t_{po}$

$t_{p11} = 0.035$

$Q_{r_i} := \eta q \cdot \frac{v_o(t_{po_i}) - v_{d_i}}{\sqrt{S_o \cdot noise}}$

$P_r(b,i) := \frac{1}{2} \cdot erfc\left(\frac{Q_{r_i} \cdot b}{\sqrt{2}}\right)$

$P_{er}(b,i) := 0.85201 \cdot P_r(b,i)$

$t_{do_i} + 0.5 > t_{po} =$
1
1
1
1
1
1
1
1
1
1

$t_{do_i} + 0.5 =$
0.499
0.499
0.499
0.499
0.499
0.499
0.499
0.499
0.499
0.499

$t_{po} =$

	0
0	0.035
1	0.035
2	0.035
3	0.035
4	0.035
5	0.035
6	0.035
7	0.035
8	0.035
9	0.035

$t_{do_i} =$
-1.483·10
-1.442·10
-1.401·10
-1.36·10
-1.319·10
-1.278·10
-1.237·10
-1.196·10
-1.155·10
-1.114·10

Erasure of first pulse in 11

$t_{p11_i} := t_{do_i} + 0.5$

$t_{p11} := t_{p11}$

$Q_{r11_i} := \eta q \cdot \frac{v_{o110}(t_{p11}) - v_{d_i}}{\sqrt{S_o \cdot noise}}$

$v_{d_i} =$

$3.345 \cdot 10^9$
$3.351 \cdot 10^9$
$3.358 \cdot 10^9$
$3.364 \cdot 10^9$
$3.371 \cdot 10^9$

$$P_{r11}(b,i) := \frac{1}{2}\cdot\operatorname{erfc}\left(\frac{Q_{r11_i}\cdot b}{\sqrt{2}}\right) \qquad v_{o110}(t_{p11}) = 6.438 \times 10^{10}$$

3.377·10⁹
3.383·10⁹
3.39·10⁹
3.396·10⁹
3.403·10⁹

$$P_{er11}(b,i) := 0.00581869\cdot P_{r11}(b,i)$$

Erasure of second pulse in 11

modify pulse parameters

$$t_{p110_i} := t_{do_i} + 1.5$$

$$t_{o110} := t_{p110}$$

$$Q_{r110_i} := \eta q\cdot\frac{v_{o110}(t_{p110_i}) - v_{d_i}}{\sqrt{S_o\cdot noise}}$$

$$P_{r110}(b,i) := \frac{1}{2}\cdot\operatorname{erfc}\left(\frac{Q_{r110_i}\cdot b}{\sqrt{2}}\right)$$

$$P_{er110}(b,i) := 0.0593262\cdot P_{r110}(b,i)$$

$t_{do_i} + 1.5 > t_{p11}$	$t_{do_i} + 1.5 =$
1	1.499
1	1.499
1	1.499
1	1.499
1	1.499
1	1.499
1	1.499
1	1.499
1	1.499
1	1.499

Erasure of third pulse in 111

modify pulse parameters

$$t_{p1110_i} := t_{do_i} + 2.5$$

$$t_{o1110} := t_{p1110}$$

$t_{do_i} + 2.5 > t_{p1110} =$	$t_{do_i} + 2.5 =$	$t_{p1110_i} =$
1	2.499	2.035

$$Q_{r1110_i} := \eta q \cdot \frac{v_{o1110}\left(t_{p1110_i}\right) - v_{d_i}}{\sqrt{S_o \cdot noise}}$$

$$P_{r1110}(b,i) := \frac{1}{2} \cdot erfc\left(\frac{Q_{r1110_i} \cdot b}{\sqrt{2}}\right)$$

$$P_{er1110}(b,i) := 0 \cdot P_{r1110}(b,i)$$

1	2.499	2.035
1	2.499	2.035
1	2.499	2.035
1	2.499	2.035
1	2.499	2.035
1	2.499	2.035
1	2.499	2.035
1	2.499	2.035
1	2.499	2.035
1	2.499	2.035

Erasure of second pulse in 101

Determine the peak amplitude of last pulse

$t := t_{pk} + 2$

$t_{p1010} := \text{root}\left[\left(I_1(t) + I_1(t-2)\right)\cdot T_s^{\,2}, t\right]$

modify pulse parameters

$t_{p1010_i} := t_{do_i} + 2.5$ $\qquad t_{p1010} = 2.035$

$t_{p1010_i} := t_{p1010}$

$$Q_{r101_i} := \eta q \cdot \frac{v_{o101}\left(t_{p1010_i}\right) - v_{d_i}}{\sqrt{S_o \cdot noise}}$$

$$P_{r101}(b,i) := \frac{1}{2}\cdot \text{erfc}\left(\frac{Q_{r101_i}\cdot b}{\sqrt{2}}\right)$$

$P_{er101}(b,i) := 0.0709229 \cdot P_{r101}(b,i)$

Erasure of third pulse in 1101

modify pulse parameters

$t_{p11010_i} := t_{do_i} + 3.5$

$t_{p11010_i} := t_{p11010}$

$$Q_{r1101_i} := \eta q \cdot \frac{v_{o1101}\left(t_{p11010_i}\right) - v_{d_i}}{\sqrt{S_o \cdot noise}}$$

$$P_{r1101}(b,i) := \frac{1}{2}\cdot \text{erfc}\left(\frac{Q_{r1101_i}\cdot b}{\sqrt{2}}\right)$$

$P_{er1101}(b,i) := 0.000366211 \cdot P_{r1101}(b,i)$

Erasure of last pulse in 10101

modify pulse parameters

$t_{p10101_i} := t_{do_i} + 4.5$

$t_{p10101_i} := t_{p10101}$

$$Q_{r10101_i} := \eta q \cdot \frac{v_{o10101}\left(t_{p10101_i}\right) - v_{d_i}}{\sqrt{S_o \cdot noise}}$$

$$P_{r10101}(b,i) := \frac{1}{2} \cdot erfc\left(\frac{Q_{r10101_i} \cdot b}{\sqrt{2}}\right)$$

$$P_{er10101}(b,i) := 0.000406901 \cdot P_{r10101}(b,i)$$

TOTAL ERASURE ERROR

$$P_{erasure}(b,i) := P_{er}(b,i) + P_{er11}(b,i) + P_{er110}(b,i) + P_{er1110}(b,i) + P_{er101}(b,i) + P_{er1101}(b,i) + P_{er101}$$

TOTAL ERROR RATE

$$P_{eb}(b,i) := P_{erasure}(b,i) + P_{ftot}(b,i) + P_{ews}(b,i)$$

$P_{ftot}(b,i) =$

$6.707 \cdot 10^{-10}$
$6.163 \cdot 10^{-10}$
$5.662 \cdot 10^{-10}$
$5.201 \cdot 10^{-10}$
$4.777 \cdot 10^{-10}$
$4.386 \cdot 10^{-10}$
$4.027 \cdot 10^{-10}$
$3.697 \cdot 10^{-10}$
$3.393 \cdot 10^{-10}$
$3.114 \cdot 10^{-10}$

$P_{ews}(b,i) =$

0
0
0
0
0
0
0
0
0
0

$P_{erasure}(b,i) =$

$3.398 \cdot 10^{-10}$
$3.684 \cdot 10^{-10}$
$3.994 \cdot 10^{-10}$
$4.329 \cdot 10^{-10}$
$4.692 \cdot 10^{-10}$
$5.084 \cdot 10^{-10}$
$5.507 \cdot 10^{-10}$
$5.966 \cdot 10^{-10}$
$6.461 \cdot 10^{-10}$
$6.997 \cdot 10^{-10}$

$P_{eb}(b,i) =$

$1.01 \cdot 10^{-9}$
$9.847 \cdot 10^{-10}$
$9.656 \cdot 10^{-10}$
$9.53 \cdot 10^{-10}$
$9.468 \cdot 10^{-10}$
$9.47 \cdot 10^{-10}$
$9.535 \cdot 10^{-10}$
$9.663 \cdot 10^{-10}$
$9.855 \cdot 10^{-10}$
$1.011 \cdot 10^{-9}$

SENSITIVITY

$pc(b,i) := \left(\log\left(P_{eb}(b,i)\right) + 9\right)$ Set for 1 in 10^9 errors

$pc(b,i) =$

$4.516 \cdot 10^{-3}$
$-6.708 \cdot 10^{-3}$
-0.015
-0.021
-0.024
-0.024
-0.021
-0.015
$-6.364 \cdot 10^{-3}$
$4.781 \cdot 10^{-3}$

$a_i := root(pc(b,i),b)$ Find the root to give 1 in 10^9

$minimum := min(a)$ $P_{ews}(b,0) = 0$ $P_{erasure}(b,0) = 3.398 \times 10^{-10}$ $a_i =$

$maximum := max(a)$

$2.978 \cdot 10^3$
$2.976 \cdot 10^3$
$2.974 \cdot 10^3$
$2.973 \cdot 10^3$
$2.973 \cdot 10^3$
$2.973 \cdot 10^3$
$2.973 \cdot 10^3$
$2.974 \cdot 10^3$
$2.976 \cdot 10^3$
$2.978 \cdot 10^3$

$minimum = 2.973 \times 10^3$

$maximum = 2.978 \times 10^3$

$P_{ftot}(b,0) = 6.707 \times 10^{-10}$

$P_{eb}(b,0) = 1.01 \times 10^{-9}$

SENSITIVITY in dBm

$minimum = 2.973 \times 10^{3}$ This is the minimum number of photons per pulse

$Pulse\_energy := minimum \cdot photon\_energy$ This is the minimum pulse energy - Joules

$Mark\_space\_correction := \frac{X}{Y}$ This is to give the average energy in the fr

$Energy\_in\_frame := Pulse\_energy \cdot Mark\_space\_correction$

$Number\_of\_PCM := pcm\_bits$

$Energy\_per\_PCM\_bit := \frac{Energy\_in\_frame}{Number\_of\_PCM}$ This is the average energy per PCM bit-tim

$Energy\_per\_PCM\_bit \cdot B = 3.815 \times 10^{-7}$ This is the average power in Watt (B is 1/bit tim

$\frac{Energy\_per\_PCM\_bit \cdot B}{10^{-3}} = 3.815 \times 10^{-4}$ This is the average power in milliWatt

$dBm := 10 \cdot \log\left(\frac{Energy\_per\_PCM\_bit \cdot B}{10^{-3}}\right)$

$dBm = -34.185$ This is the sensitivity in dBm - dB referenced to 1

INITIALIZATION OF GLOBAL VARIABLES AND TABULATED RESULTS

$t_{pk} \equiv 0.042$ $v_{off} \equiv 0.5195$ $t_d \equiv -0.012$ $b \equiv 2977$ $f_n \equiv 30$

range $\equiv 1000$ minimum $= 2.973 \times 10^3$

maximum $= 2.978 \times 10^3$ dBm $= -34.185$

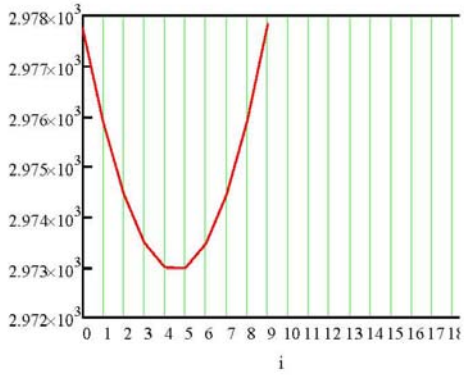
$P_{eb}(b,0) = 1.01 \times 10^{-9}$

$\frac{v_{d_i}}{v_{po}} =$

0.519
0.52
0.521
0.522
0.523
0.524
0.525
0.526
0.527
0.528

$a_i =$

$2.978 \cdot 10^3$
$2.976 \cdot 10^3$
$2.974 \cdot 10^3$
$2.973 \cdot 10^3$
$2.973 \cdot 10^3$
$2.973 \cdot 10^3$
$2.973 \cdot 10^3$
$2.974 \cdot 10^3$
$2.976 \cdot 10^3$
$2.978 \cdot 10^3$



X $\equiv 12$ Y $\equiv 2$ pcm_bits $\equiv 6$

APPENDIX E

Mathematical Analysis
Raised Cosine Filtering

12-1 PPM SYSTEM

NORMALIZED BANDWIDTH OF 30

MPPM investigation using a RAISED COSINE filter with 1Gbit/s data and WHITE noise

Set up the scan limits

range := 1000

i := 0,1 .. 9

n := 10

$v_i := v_{off} + \frac{i}{range}$

x := 0 .. n

This gives the row of the matrix

y := 0 .. n

v =

	0
0	0.87
1	0.871
2	0.872
3	0.873
4	0.874
5	0.875
6	0.876
7	0.877
8	0.878
9	0.879

Preamplifier terms

$f_p \equiv 1.2 \cdot 10^9$ Preamp bandwidth

$S_o \equiv 16 \cdot 10^{-24}$ Preamp noise at input - double sided Philips TZA 3043

$B \equiv 1 \cdot 10^9$ Bit rate

pcm_bits ≡ 3

$T_b \equiv \frac{1}{B}$ PCM bit time

X ≡ 12

Y ≡ 1

$T_s \equiv \frac{pcm\_bits \cdot T_b}{X}$ Slot time

n := 10 Number of like symbols in PCM

$\eta q := 1.6 \cdot 10^{-19}$ Quantum energy

$\lambda := 1.55 \cdot 10^{-6}$ This is the wavelength of operation

$photon\_energy := \frac{6.63 \cdot 10^{-34} \cdot 3 \cdot 10^8}{\lambda}$

$R_o := \frac{\eta q}{photon\_energy}$

Pulse shape terms

$\alpha := \frac{0.1874 \cdot T_b}{f_n}$

$\alpha_n := \frac{0.1874 \cdot T_b}{f_n \cdot T_s}$

Noise corner frequency.

$$\omega_{pn} := 2\cdot\pi\cdot f_p\cdot T_s \qquad B_s := \frac{1}{T_s}$$

Pulse shape

$$Pulse(t) := \frac{1}{\sqrt{2\cdot\pi}\cdot\alpha_n\cdot T_s}\cdot\exp\left(\frac{-t^2}{2\cdot\alpha_n{}^2}\right) \qquad \omega_p := 2\cdot\pi\cdot f_p$$

$$\omega_n := 6\cdot 10^9$$

Pulse shape
Ideal Raised Cosine filter only

$\beta := 0.99$ This is the roll-off factor of the RC filter use lim to find the Io(Ts/2,-Ts/2)

$$I_0(t) := 1\cdot\frac{1}{T_s}\cdot\begin{cases} 1 & \text{if } t = 0 \\ 1 & \text{if } t = \frac{1\cdot T_s}{2} \\ 1 & \text{if } t = \frac{-1\cdot T_s}{2} \\ \left[\frac{\sin\left(\pi\cdot\frac{t}{T_s}\right)\cdot\cos\left(\pi\cdot\beta\cdot\frac{t}{T_s}\right)}{\left(\pi\cdot\frac{t}{T_s}\right)\cdot\left(1-4\cdot\beta^2\frac{t^2}{T_s{}^2}\right)}\right] & \text{otherwise} \end{cases}$$

$$I_3(f_n) := \frac{1}{\pi}\cdot\int_0^{2\pi}\left[\left[\frac{1}{2}\cdot\frac{\left(1+\cos\left(\frac{x}{2}\right)\right)}{e^{\frac{-(\alpha_n\cdot x)^2}{2}}}\right]\right]^2 dx \qquad I_4(f_n) := \frac{1}{\pi}\cdot\int_0^{2\pi} x^2\cdot\left[\left[\frac{1}{2}\cdot\frac{\left(1+\cos\left(\frac{x}{2}\right)\right)}{e^{\frac{-(\alpha_n\cdot x)^2}{2}}}\right]\right]^2 dx$$

$$NEB(f_n) := I_3(f_n)\cdot B_s + I_4(f_n)\cdot\frac{B_s^{\,3}}{\omega_n^{\,2}} \qquad I_4(f_n)\cdot\frac{B_s^{\,3}}{\omega_n^{\,2}}$$

$$noise := S_o\cdot NEB(f_n)$$

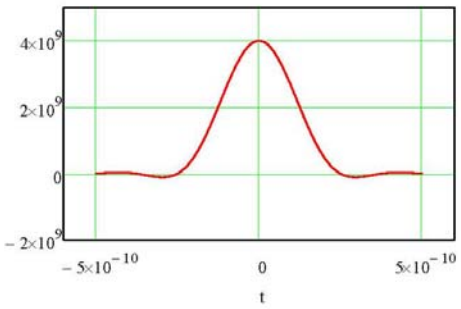
$$noise = 1.158\times 10^{-13}$$

Set up the pulse shapes

12 – 1 PPM System

$I_0(t)$

$$t := -5\cdot 10^{-10}, -4.99\cdot 10^{-10} .. 5\cdot 10^{-10}$$



$$v_o(t) := I_0(t)$$

$$v_{o10}(t) := I_0(t)$$

$$v_{o110}(t) := I_0(t) + I_0(t-1)$$

$$v_{o101}(t) := I_0(t) + I_0(t - 2)$$

Sequence isolated pulse

peak value

$$v_{po} := v_o(t_{pk})$$

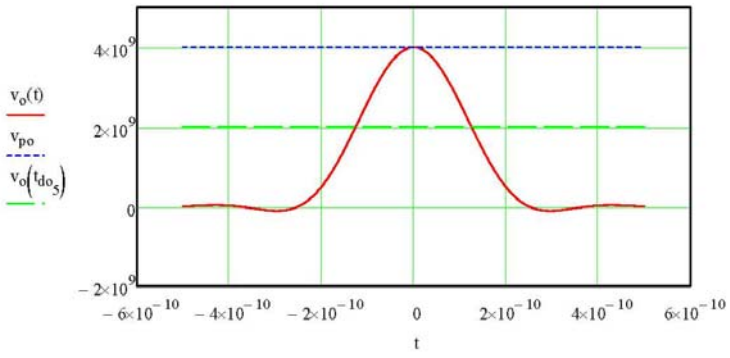
decision time

$$t_{do_i} := \text{root}\left[\left(I_0(t_d) - v_i \cdot I_0(t_{pk})\right) \cdot T_s^{\,2}, t_d\right]$$

$$v_{d_i} := I_0\left(t_{do_i}\right)$$

$$\frac{v_{d_5}}{v_o(t_{pk})} = 0.5$$

$$t := -5 \cdot 10^{-10}, -4.99 \cdot 10^{-10} .. 5 \cdot 10^{-10}$$



Sequence 10

peak value

Same as an isolated pulse

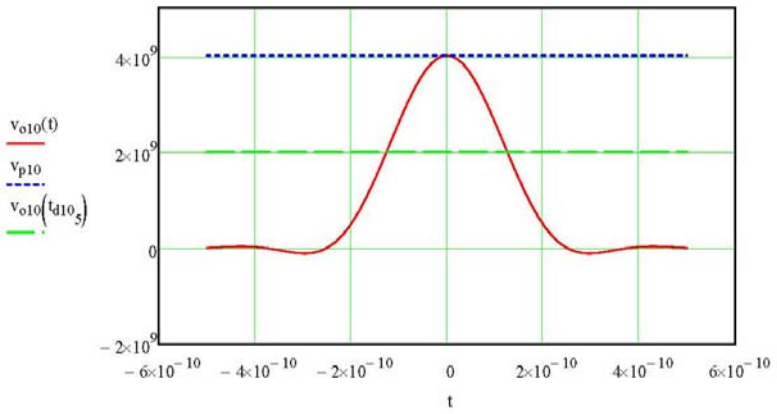
$v_{p10} := v_{o10}(t_{pk})$

decision time

$t_{d10_i} := \text{root}\left[\left(v_{o10}(t_d) - v_{d_i}\right)\cdot T_s^{\,2}, t_d\right]$

$$\frac{v_{d_5}}{v_{o10}(t_{pk})} = 0.5$$

$$\frac{v_{o10}\left(t_{d10_5}\right)}{v_{p10}} = 0.5$$



Sequence 110

peak value

First Pulse

$t_{p11} := t_{pk}$

$v_{p11} := v_{o110}(t_{pk})$

$v_{p11} = 4 \times 10^9$

Second Pulse

$t_{pk1} := t_{pk} + 1$

$t_{p110} := t_{pk1}$

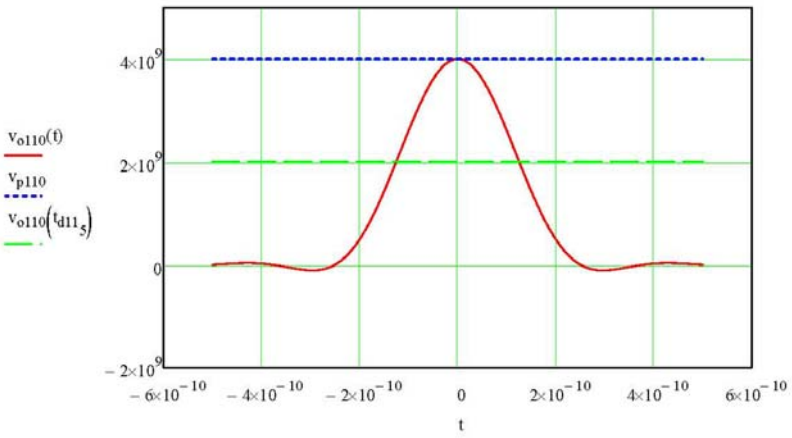
$v_{p110} := v_{o110}(t_{p110})$

$v_{p110} = 4 \times 10^9$

decision time

$$t_{d11_i} := \text{root}\left[\left(v_{o110}(t_d) - v_{d_i}\right)\cdot T_s^{\,2}, t_d\right]$$

$$\frac{v_{d_5}}{v_{o110}(t_{p110})} = 0.5$$



Sequence 101

peak value

Same as isolated pulse and 10 sequence

$$t_{p101} := t_{pk}$$

$$t_{p101} = 0$$

$$v_{p101} := v_{o101}(t_{p101})$$

$v_{p101} = 4 \times 10^9$

Second Pulse

$t_{pk66} := t_{pk} + 2$

$t_{p1010} := t_{pk} + 2$

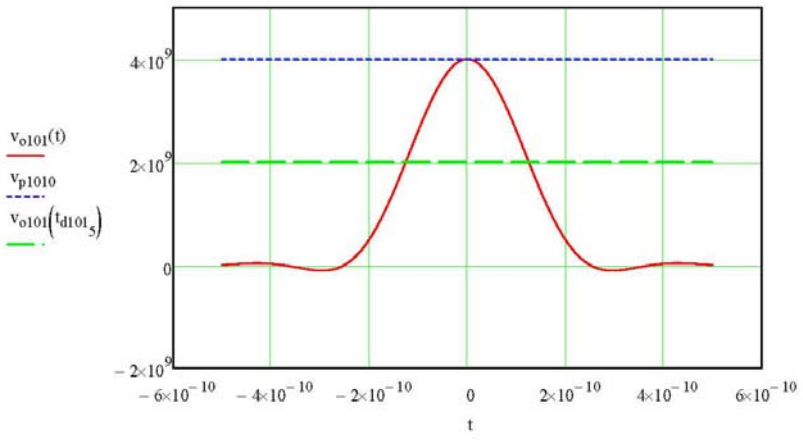
$v_{p1010} := v_{o101}(t_{p1010})$

$v_{p1010} = 4 \times 10^9$

$t_{p1010} = 2$

decision time

$t_{d101_i} := \text{root}\left[\left(v_{o101}(t_d) - v_{d_i}\right) \cdot T_s^{\,2}, t_d\right]$ $\qquad \frac{v_{d_5}}{v_{o101}(t_{p11})} = 0.5$



ERROR SOURCES

FALSE ALARM ERROR

Equivalent pcm error number is found by error_number/((x-y)*pcm_bits*pcm_words)

Standard false alarm

$$Q_f(b,i) := \frac{b \cdot \eta q \cdot v_{d_i}}{\sqrt{noise}}$$

$$P_f(b,i) := \frac{1}{2} \cdot erfc\left(\frac{Q_f(b,i)}{\sqrt{2}}\right)$$

$$P_{ef}(b,i) := 0.32197 \cdot P_f(b,i)$$

$Q_f(b,i) =$

5.862
5.862
5.862
5.862
5.862
5.862
5.862
5.862
5.862
5.862

Slot after pulse false alarm - time td+Ts

$$v_{o010_i} := v_{o10}\left(t_{do_i} + 1\right)$$

$$Q_{f10}(b,i) := \frac{b \cdot \eta q \cdot \left(v_{d_i} - v_{o010_i}\right)}{\sqrt{noise}}$$

$$P_{fl0}(b,i) := \frac{1}{2}\cdot\mathrm{erfc}\left(\frac{Q_{fl0}(b,i)}{\sqrt{2}}\right)$$

$$P_{efl0}(b,i) := 0.0454545\cdot P_{fl0}(b,i)$$

Slot after double pulse false alarm - time td+2Ts

$$v_{o0110_i} := v_{o110}\left(t_{do_i} + 2\right)$$

$$Q_{f0110}(b,i) := \frac{b\cdot\eta q\cdot\left(v_{d_i} - v_{o0110_i}\right)}{\sqrt{noise}}$$

$$P_{f0110}(b,i) := \frac{1}{2}\cdot\mathrm{erfc}\left(\frac{Q_{f0110}(b,i)}{\sqrt{2}}\right)$$

$$P_{ef0110}(b,i) := 0.0364583\cdot P_{f0110}(b,i)$$

Slot between pulses false alarm - time td+Ts

$$v_{o0101_i} := v_{o101}\left(t_{do_i} + 1\right)$$

$$Q_{f101}(b,i) := \frac{b \cdot \eta q \cdot \left(v_{d_i} - v_{o0101_i}\right)}{\sqrt{noise}}$$

$$P_{f101}(b,i) := \frac{1}{2} \cdot erfc\left(\frac{Q_{f101}(b,i)}{\sqrt{2}}\right)$$

$$P_{ef101}(b,i) := 0.0364583 \cdot P_{f101}(b,i)$$

TOTAL FALSE ALARM ERROR

$$P_{ftot}(b,i) := P_{ef}(b,i) + P_{ef10}(b,i) + P_{ef0110}(b,i) + P_{ef101}(b,i)$$

$P_{ftot}(b,i) =$

$1.004 \cdot 10^{-9}$
$1.004 \cdot 10^{-9}$
$1.004 \cdot 10^{-9}$
$1.004 \cdot 10^{-9}$
$1.004 \cdot 10^{-9}$
$1.004 \cdot 10^{-9}$
$1.004 \cdot 10^{-9}$
$1.004 \cdot 10^{-9}$
$1.004 \cdot 10^{-9}$
$1.004 \cdot 10^{-9}$

ERASURES

Equivalent pcm error number is found by error_number/(pcm_bits*pcm_words)

Standard erasure of pulse

modify pulse parameters

$$Q_r(b,i) := \frac{b \cdot \eta q \cdot \left(v_o\left(t_{pk}\right) - v_{d_i}\right)}{\sqrt{noise}}$$

$$P_r(b,i) := \frac{1}{2} \cdot erfc\left(\frac{Q_r(b,i)}{\sqrt{2}}\right)$$

$$P_{er}(b,i) := 1.125 \cdot P_r(b,i)$$

Erasure of second pulse in 11

$$Q_{r110}(b,i) := \frac{b \cdot \eta q \cdot \left(v_{o110}\left(t_{pk} + 1\right) - v_{d_i}\right)}{\sqrt{noise}}$$

$$P_{r110}(b,i) := \frac{1}{2} \cdot erfc\left(\frac{Q_{r110}(b,i)}{\sqrt{2}}\right)$$

$$P_{er110}(b,i) := 0 \cdot P_{r110}(b,i)$$

Erasure of second pulse in 101

$$Q_{r101}(b,i) := \frac{b \cdot \eta q \cdot \left(v_{o101}\left(t_{pk} + 2\right) - v_{d_i}\right)}{\sqrt{noise}}$$

$$P_{r101}(b,i) := \frac{1}{2} \cdot erfc\left(\frac{Q_{r101}(b,i)}{\sqrt{2}}\right)$$

$$P_{er101}(b,i) := 0 \cdot P_{r101}(b,i)$$

TOTAL ERASURE ERROR

$$P_{erasure}(b,i) := P_{er}(b,i) + P_{er110}(b,i) + P_{er101}(b,i)$$

TOTAL ERROR RATE

$$P_{eb}(b,i) := P_{erasure}(b,i) + P_{ftot}(b,i)$$

$P_{ftot}(b,i) =$

$1.004 \cdot 10^{-9}$
$1.004 \cdot 10^{-9}$
$1.004 \cdot 10^{-9}$
$1.004 \cdot 10^{-9}$
$1.004 \cdot 10^{-9}$
$1.004 \cdot 10^{-9}$
$1.004 \cdot 10^{-9}$
$1.004 \cdot 10^{-9}$
$1.004 \cdot 10^{-9}$
$1.004 \cdot 10^{-9}$

$P_{eb}(b,i) =$

$3.498 \cdot 10^{-9}$
$3.498 \cdot 10^{-9}$
$3.498 \cdot 10^{-9}$
$3.498 \cdot 10^{-9}$
$3.498 \cdot 10^{-9}$
$3.498 \cdot 10^{-9}$
$3.498 \cdot 10^{-9}$
$3.498 \cdot 10^{-9}$
$3.498 \cdot 10^{-9}$
$3.498 \cdot 10^{-9}$

$P_{erasure}(b,i) =$

$2.493 \cdot 10^{-9}$
$2.493 \cdot 10^{-9}$
$2.493 \cdot 10^{-9}$
$2.493 \cdot 10^{-9}$
$2.493 \cdot 10^{-9}$
$2.493 \cdot 10^{-9}$
$2.493 \cdot 10^{-9}$
$2.493 \cdot 10^{-9}$
$2.493 \cdot 10^{-9}$
$2.493 \cdot 10^{-9}$

$$dBm := 10 \cdot \log\left(\frac{Energy\_per\_PCM\_bit \cdot B}{10^{-3}}\right)$$

$dBm = -46.381$ This is the sensitivity in dBm - dB referenced to 1 mW

INITIALIZATION OF GLOBAL VARIABLES AND TABULATED RESULTS

$t_{pk} = 0$ $v_{off} = 0.87$ $f_n = 30$ $b = 6238$ $t_d = -1.2545 \cdot 10^{-10}$

$sensitivity := 2966$

APPENDIX F

Publications

Publication 1

Publication 2

Publication 3

Publication 4

Investigation of an Optical Multiple PPM Link over a Highly Dispersive Optical Channel

K. Nikolaidis, M.J.N. Sibley

Department of Engineering

Huddersfield University, Queensgate, Huddersfield, HD1 3DH

Abstract

This paper describes a performance analysis of $\binom{X}{Y}$ multiple PPM systems, in which X denotes the number of data slots and Y the number of pulses, operating over a plastic optical fibre (POF) channel. The effects of receiver noise and channel dispersion are accounted for and the manner in which the erasure, wrong-slot and false-alarm errors affect system performance is examined. The receiver/decoder uses slope detection and a maximum likelihood sequence detector (MLSD). As the analysis of any $\binom{X}{Y}$ multiple PPM system is extremely time-consuming, a novel automated solution was designed to predict the equivalent PCM error rates of specific sequences and simplify the task. We also propose a measure of coding quality that accounts for efficiency of coding and bandwidth expansion. Using this measure, original results show that a $\binom{12}{6}$ system is the most efficient for a wide range of bandwidths.

1. Introduction

Digital Pulse Position Modulation (digital PPM) is a modulation format that codes n bits of PCM into a single pulse that occupies one of 2^n time slots. It is considered to be optimal for implementation over optical fibre channels and is the preferred modulation format for the ideal photon counting channel and optical inter-satellite links. A digital PPM frame usually has several empty time slots which are used as a guard interval to reduce the effects of inter-symbol (ISI) and inter-frame (IFI) interference. Various studies over the last three decades [1-8] have shown that such a scheme can offer an improvement in receiver sensitivity of 5-11 dB when compared to PCM. Such an increase in sensitivity can be exploited beneficially in both long haul applications and the multi-user environment. However, this improvement results in a considerable increase in the final data rate of the original PCM [9], and this makes implementation difficult.

Alternative methods have been proposed, such as multiple PPM [9-16], dicode PPM [17, 18], differential PPM [19, 20] and overlapping PPM [21] that reduce transmission bandwidth while maintaining an increased sensitivity. Dicode and multiple PPM [9] are the most bandwidth efficient of these formats and multiple PPM, the subject of this paper, offers the best sensitivity without the large bandwidth increase. In this scheme, multiple pulses per frame are used, with the pulse positions being determined by the original PCM word. Thus if k pulses are transmitted in a timeframe of n slots, the number of combinations are $\binom{n}{k} = \frac{n!}{k!(n-k)!}$. Ideally, $\binom{n}{k}$ should be a power of two to ease implementation but for $k > 1$ this is rarely the case. For example, a $\binom{12}{2}$ multiple PPM system uses a 12-slot frame with 2 data pulses to code 6 bits of PCM data. This gives 64 valid multiple PPM frames

instead of the available 66. If linear mapping is used, the PCM word 000000 is translated to a code-word with pulses in slots 1 and 2 (referred to as a [1,2] code-word) and the PCM word 111111 is translated to the [10,11] code-word. If higher order codes are used, such as $\binom{12}{6}$, further reductions in bandwidth can be achieved because it becomes possible to encode up to 9 bits.

Sugiyama and Nosu [13] proposed a detailed noise performance of a $\binom{12}{2}$ multiple PPM, optical fibre system in the presence of erasure errors. A Maximum Likelihood Sequence Detector (MLSD) was used as the decoder-detector and this same scheme is also used in this paper. They concluded that multiple PPM is more efficient than digital PPM in terms of power and bandwidth utilization, resulting in a best predicted sensitivity of 0.58 bits/photon compared to the 0.5 bits/photon for digital PPM, both operating at an error rate of 1 in 10^{-9}. The use of error reduction codes such as Reed-Soloman (RS) to further increase receiver sensitivity was proposed by Atkin and Fung [22]. They analyzed the performance of a RS coded multiple PPM system using an avalanche photodiode (APD) and predicted 0.1 nats/photon compared to the 0.03 nats/photon, for an equivalent (RS) PPM system, both operating at an error rate of 1 in 10^{-9}.

Park and Barry [14] examined the performance of multiple PPM and its variants PPM and OPPM on ISI channels with additive white Gaussian noise. The error probability and channel capacity results indicated that, although PPM modulation schemes were extremely power efficient across ISI-free channels, their power efficiency dropped dramatically when ISI was present. The same authors also investigated the effect of dispersion on multiple PPM [15] for indoor wireless infrared communication. They concluded that a partial-response pre-coding at the transmitter reduces the ISI span to two baud periods, which reduced the complexity of the receiver significantly providing a good balance of

performance and complexity. They also concluded and showed that a $\binom{12}{2}$ combination is particularly efficient [16]. Sibley [9] presented an original performance analysis of a $\binom{12}{2}$ multiple PPM system with a slope detection system coupled with a classical matched filter, MLSD scheme to combat inter-symbol interference. The author concluded that this multiple PPM scheme had a 7.36 dB advantage over PCM when operating under wide bandwidth conditions. When all consecutive pulses were replaced by three-pulse sequences to reduce the effects of ISI and IFI at low bandwidths, it was shown that such a hybrid 2/3 pulse system gave a sensitivity of -22.74 dBm at a channel bandwidth of 0.7 times the PCM bit rate. This represented a 3.61 dB improvement over the original two-pulse system.

In this paper, we present an original method of predicting the sensitivity of any multiple PPM system using MLSD and results are presented for a $\binom{12}{Y}$ multiple PPM scheme. Section 2 discusses the types of error, error probability and MLSD scheme used in the multiple PPM system. Section 3 presents the MLSD approach using specific multiple PPM sequences in a $\binom{12}{Y}$ multiple PPM scheme, and a simplified algorithm is presented. Section 4 presents and discusses the results, and section 5 presents the conclusions.

2. Pulse Detection Errors

As with digital PPM, multiple PPM systems suffer from three types of error, erasure, false alarm and wrong-slot. The following three sections present expressions for their respective error probabilities [3, 4].

2.1 Erasure Error

Erasure errors are generated by noise present at the decision (sampling) time causing the amplitude of the pulse to fall below the threshold voltage. The probability of this occurring, P_e, is given by (1):

$$P_e = 0.5\mathrm{erfc}\left(\frac{Q_e}{\sqrt{2}}\right) \tag{1}$$

where Q_e is given by (2):

$$Q_e = \frac{v_{pk} - v_d}{\sqrt{\langle n_o^{\,2} \rangle}} \tag{2}$$

where the symbols have the following meanings:

$v_{pk} = v_o(t_{pk})$	the peak receiver output,
$v_d = v_o(t_d)$	the receiver output at the threshold crossing time td,
$\langle n_o^{\,2} \rangle$	the mean square receiver output noise.

When an Erasure error occurs, one pulse is removed from the multiple PPM frame. Thus, if the frame has only 2 pulses, as with $\binom{12}{2}$ multiple PPM, only one pulse is present. In this case the number of resultant PCM errors is found in the same way as that used by Sibley [9].

2.2 False Alarm Error

For the False-Alarm Error (FA), noise in an empty slot could cause a threshold violation and so a pulse could be detected in an empty slot. The probability of this occurring is given by (3):

$$P_f = \frac{T_s}{\tau_R} 0.5 \mathrm{erf}\left(\frac{Q_f}{\sqrt{2}}\right) \quad (3)$$

where,

$$Q_f = \frac{v_d - v_o(t_d)}{\sqrt{\langle n_o^{\,2} \rangle}} \quad (4)$$

and

$\frac{T_s}{\tau_R}$ = the number of uncorrelated samples/time slot

τ_R = the time at which the autocorrelation function of the filter has become small

$v_o(t_d)$ = the signal voltage in the slot considered, which can be non-zero due to the effects of ISI.

In the case of a FA error, an extra pulse is detected on a frame. The treatment is identical to that used by Sibley [9].

2.3 Wrong-Slot Errors

Noise on the leading or falling edge of a pulse can cause it to appear either before or after the current slot. To minimize this error, detection should occur at the centre of the current slot. Hence the probability of a Wrong-Slot error (WS), P_s, is given by (5):

$$P_s = 0.5\mathrm{erfc}\left(\frac{Q_s}{\sqrt{2}}\right) \tag{5}$$

with Q_s defined by (6):

$$Q_s = \frac{T_s}{2}\frac{\mathrm{slope}(t_d)}{\sqrt{\langle n_o{}^2\rangle}} \tag{6}$$

where,

T_s = the slot width,

$\mathrm{slope}(t_d)$ = the slope of the received pulse at the threshold crossing instant t_d,

When a WS error occurs, a pulse can be detected immediately before or after the correct slot depending on the size of the dispersion and receiver noise as described by Garrett [3, 4]. For example, consider the codeword [3,5] which decodes to the 010110 equivalent PCM word (if a linear mapping is used). If a WS error occurs on the first pulse of the code-word causing it to appear too early, the detected code-word will be [2,3,5]. If the noise and dispersion causes the pulse to appear one slot later, the code word [4,5] results. Similarly, if the WS error occurs on the 2nd pulse, the possible detected code-words might be [3,4,5] or [3,6]. Thus four possible PCM words result as shown in Table 1. The output of the MLSD, as described by Sibley [9], is obtained by summing the logic 1s for each individual bit and averaging over the number of code-words. A PCM bit is assigned to logic one if it's weighting is greater than 0.5, a logic 0 if it is less than 0.5, or undefined 'U' for worst case scenarios when equal to

0.5 (Table 1). The bit-by-bit comparison between the original code-word and the averaged MLSD codeword gives the average error per PCM bit for a specific multiple PPM sequence and the total wrong slot error is found by averaging all possible WS code-words. A problem occurs in characterising an error as a WS error when IFI occurs on the first or last pulse in a multiple PPM frame. If a WS error occurs on a pulse in the first slot of the frame, a false pulse could occur in the last slot of the *preceding* frame. This would appear to give a False-Alarm error in the frame before the one under consideration. If a WS error occurs on a pulse in the last slot of the frame, the pulse could effectively move into the first slot of the *following* frame. Thus the original pulse is lost and the treatment is similar to an Erasure error.

3. Software Description

The error sources just described will generate errors in the decoded PCM word. In order to calculate the error probability and the effects of ISI and IFI, Sibley [9] considered specific pulse sequences such as *0* and *1* (called standard error) and 1*0*, 11*0*, *1*1, 1*1*, 1*0*1, 10*1* and 10*1*1 caused by ISI or IFI of adjacent slots with the symbol in error being represented in *italics*. The error probability was determined by applying the MLSD decoding and then weighting by the probability that the particular sequence occurs.

This process can be very tedious and time consuming (420 combinations need to be considered just for FA errors in a $\binom{12}{2}$ multiple PPM system) especially if the system is large such as $\binom{22}{5}$. Hence, a software solution, using novel algorithms, was developed that simulates a multiple PPM decoder. In this system, registers are created according to the multiple PPM system (using Dynamic Memory Allocation) and every possible code-word of the system is generated. Digital 1s and 0s represent

occupied and empty multiple PPM slots and so the [1,2] code-word in a $\binom{12}{2}$ multiple PPM system is represented as 110000000000. The remaining bits in the registers contain the equivalent PCM data and so the [1,2] register contains 110000000000|000000 if linear increment mapping starting from zero is used. Other code-words are dealt with in a similar fashion.

Errors in received code-words are dealt with in the following manner. All possible corrupted code-words are generated and stored in registers, together with their equivalent PCM data. These corrupted code-words are generated when an Erasure, False Alarm or Wrong Slot error occurs. For example, in a $\binom{12}{2}$ multiple PPM system the [4,7] codeword can be detected as [4,?] if an erasure error occurs on the second pulse of the codeword. The corrupted code-word is [4,?] and its equivalent PCM data can be found by averaging the PCM data of all code-words with one pulse in slot 4 [9]. The [4,7] codeword could also be detected as [3,4,7] due to a False Alarm. Hence, a corrupted codeword is generated and its equivalent PCM data can be found by averaging the PCM bits of [3,4], [4,7] and [3,7] code-words. The wrong slot corrupted codeword is a mixture of false alarm and erasures. Thus, [4,7] can be detected as [3,4,7] or [4,6,7] (false alarm error caused by left dispersion of the first or second pulse) or as [5,7] or [4,8] (right dispersion of the first or second pulse). If the codeword has adjacent pulses such as [3,4] the codeword can be detected as [?,4] (erasure). The bit-by-bit comparison (XOR) of the weighted and original data will give the number of PCM error bits for every error sequence. (An example is given by Sibley [9] in which the second pulse in [4,7] is erased to give [4,?].) The software automatically checks every code-word for every possible error sequence and calculates the number of PCM error bits that every sequence generates. The PCM equivalent error rate of every error sequence (average error per PCM bit in Table 1) can be obtained and used to calculate the total error probability

of a multiple PPM system. Therefore, for a $\binom{12}{2}$ multiple PPM system, 12 erasure, 220 false alarm and 232 wrong slot errors need to be considered.

If this methodology is expanded to larger systems, a very large number of sequences need to be considered so making this analysis again very prolonged. The solution was the development of a new algorithm that only considers the most important and common sequences in any $\binom{X}{Y}$ multiple PPM system. This simplification can be achieved because some particular sequences rarely occur and so have a negligible effect on the equivalent PCM error rate. Some sequences can be further grouped and so only the 26 sequences shown in table 2 need to be considered to predict the performance of any $\binom{X}{Y}$ multiple PPM system, even when the effects of ISI and IFI are accounted for.

4. Discussion and Mathematical Analysis

In a mathematical analysis of a $\binom{12}{2}$ multiple PPM system, Sibley [9] used a 1.2 GHz bandwidth PINBJT receiver, with a frequency independent, input equivalent noise current spectral density of 16 x 10^{-24} A^2/Hz (double-sided), a classical matched filter and a threshold crossing detector. An operating wavelength of 650 nm and a photo-diode quantum efficiency of 100% were taken. Simulations were carried out using a data rate of 1 Gbit/s and the total equivalent PCM error probability obtained, by adding together the individual probabilities, was taken to be 1 error in 10^9 pulses. For a given sets of parameters, the pulse shapes, derivates and the noise were found and the number of photons per bit, b, calculated. A threshold parameter, v, was defined as

$$v = \frac{v_d}{v_{pk}} \qquad (7)$$

v_{pk} = the peak voltage of an isolated pulse

The same parameters and method for predicting sensitivity were used in this investigation to examine a $\binom{12}{Y}$ multiple PPM system. All the experiments considered transmission through plastic optical fibre with a Gaussian impulse response [12]. The channel bandwidth was normalised to the PCM bit-rate and varied between 100 and 1.2. Operation below 1.2 was impossible due to the high levels of ISI and IFI causing sequences such as 110, 1101 and 1110 to be received as 10, 101 and 10 respectively as shown in figure 1.

4.1 Sensitivity Performance of $\binom{12}{Y}$ Multiple PPM

Table 3 details the variation in the number of photons per multiple PPM pulse, for a PCM bit error rate of 1 in 10^9, as the channel normalised bandwidth, f_n, varies from 100 to 1.2. Two sets of data are presented representing the best and worst sensitivities at a particular bandwidth, and each set compares results from the complete and simplified analyses. As can be seen, there is little difference between the two analyses and so the simplified version was used in the simulations. It is apparent from the table that the optimum coding level ranges from 12-2 for high bandwidths to 12-1 for low bandwidths. However, this representation of the data does not take into account the coding efficiency of the systems

– a $\binom{12}{1}$ system can code up to 3 PCM bits while a $\binom{12}{2}$ system can code up to 6 PCM bits. Thus the $\binom{12}{2}$ system could be regarded as more bandwidth efficient. In order to explore this, we define a bandwidth expansion, BE, parameter as

$$BE = \frac{X}{n} \tag{8}$$

where X is the number of multiple PPM slots in a frame. A process of normalisation is now applied to the sensitivities and bandwidth expansion so that all values are expressed between 0 and 1. To achieve this, the number of photons required for each coding level are divided by the largest number of photons per pulse for that particular bandwidth (worst case sensitivity in table 3) and all bandwidth expansions are divided by the largest BE (12/3 = 4 for all f_n). These normalized results are shown in table 4.

A weighted sum approach can now be applied to the multiple PPM systems (at every value of f_n) in terms of sensitivity and bandwidth expansion, with a range of weights between 0 and 100% in steps of 10%. A weighting of 0% means bandwidth expansion is the most important parameter while a weighting of 100% means sensitivity is more important. A 50% weighting is where the two parameters are equally important and is a key point of interest. Thus, using (9), an efficiency factor, η, can be defined as

$$\eta = 1 - \left(ph_{PCM\,norm} \times weight_{sens} + BE_{norm} \times weight_{BE}\right) \tag{9}$$

where,

$ph_{PCM\,norm}$ is the normalized sensitivity in photons per PCM bit

12

BE_{norm} is the normalized bandwidth expansion

$weight_{sens}$ is the sensitivity weighting = 0 to 1 in steps of 0.1, and

$weight_{BE}$ is the bandwidth weighting = 1- $weight_{sens}$

A system is considered to be 100% efficient if it has the best sensitivity and the lowest BE (coding 9 PCM bits). On the other hand, a system is 0% efficient if it has the lowest sensitivity and the bandwidth expansion of the $\binom{12}{11}$ system (only 3 PCM bits can be encoded). Table 5 shows the efficiency map for a 12-Y multiple PPM system across the range of f_n. As can be seen, the $\binom{12}{6}$ multiple PPM system is generally the most efficient system. It is only when the bandwidth is extremely low that the coding level changes to $\binom{12}{5}$. Both systems code 9 bits of PCM data but the normalised bandwidth expansion for the $\binom{12}{5}$ code is slightly lower. When the sensitivity weighting is greater than 90%, the optimum coding level is generally $\binom{12}{2}$. In this situation, bandwidth expansion is not important and so the higher sensitivity, lower codes can be used. The simulation predicts that $\binom{12}{1}$ (i.e. digital PPM) should be used at very low bandwidths. This is because there is a very high level of pulse dispersion leading to an increased level of ISI and IFI and, as digital PPM only has one pulse in a frame, it will not be affected in the same way as a two pulse frame.

Figure 2 shows a surface plot for a 12-Y multiple PPM system operating with a normalized bandwidth of 30. In this figure, the efficiency factor, η, has been plotted on the vertical axis to demonstrate the

optimum. This figure clearly shows that the $\binom{12}{6}$ coding system is the most efficient except at the extreme when sensitivity is more important than operating speed (above 80-20 weighting). In this instance smaller systems are more efficient such as $\binom{12}{2}$.

5. Conclusion

This paper has examined the performance of a $\binom{12}{Y}$ multiple PPM system coding 1 Gbit/s PCM data. The channel chosen was plastic optical fibre with a Gaussian impulse response and the optical receiver had a 1.2 GHz bandwidth, white noise, a classical matched filter and a threshold crossing detector. Slope detection was used together with a maximum likelihood sequence detector, in order to recover the original PCM data. The effects of Wrong-Slot, False-Alarm and Erasure errors were examined in detail. A novel algorithm was described that greatly reduces the amount of time taken to predict the performance of a multiple PPM system. It was shown that the analysis could be simplified further if error sequences are grouped automatically and reduced in number so that more complicated systems can be analyzed.

A novel system parameter, the efficiency factor, has been proposed to account for the coding efficiency of multiple PPM. Using this parameter, it has been shown that when the number of pulses is increased in a $\binom{12}{Y}$ multiple PPM system, leading to a more bandwidth efficient system, sensitivity is degraded. However if bandwidth efficiency is accounted for, results show that $\binom{12}{6}$ multiple PPM system is

generally the most efficient one. For very low bandwidths, lower systems are more efficient such as $\binom{12}{2}$ and $\binom{12}{1}$. This is due to the high level of pulse dispersion causing detection errors.

References

1. KARP, S. and GAGLIARDI, R.M.: "The design of a Pulse-Position Modulated Optical Communication System", IEEE Trans. in Communications, vol. com -17, Number 6, pp 670-676, December 1969.
2. BLACHMAN, N.M.: "The SNR Threshold in PPM Reception", IEEE Trans. in Communications, vol. com-22, Number 8, pp 1094-1098, August 1974.
3. GARRETT, I.: "Digital pulse-Position Modulation over dispersive optical fibre channels", Presented at Int. Workshop on Digital communications, Tirrenia, Italy, 15-19 August 1983.
4. GARRETT, I.: 'Digital pulse-position modulation over slightly dispersive optical fibre channels', International symposium on *Information theory*, St. Jovite, 1983, pp. 78-79
5. CALVERT, N.M., SIBLEY, M.J.N., and UNWIN, R.T.: "Experimental optical fibre digital pulse-position modulation system", Electron.Lett., 1988, 24, pp.129-131.
6. GARRETT, I., CALVERT, N.M., SIBLEY, M.J.N., UNWIN, R.T. and CRYAN, R.A.: "Optical Fibre digital pulse position modulation", Br.Telecom. Technol. J., vol.7, no. 3, pp 5-11, 1989.
7. CRYAN, R.A., UNWIN, R.T., GARRETT, I., SIBLEY, M.J.N. and CALVERT, N.M.: "Optical Fibre digital pulse-position modulation assuming a Gaussian received pulse shape", IEE Proc.J., vol. 137, no. 4, pp 89-96, 1990.
8. SIBLEY, M.J.N. and MASSARELLA, A.J.: "Detection of digital pulse position modulation over highly/slightly dispersive optical channels", Presented at SPIE Conf. on Video Communications and Fiber Optic Networks, Berlin, 1993.
9. SIBLEY, M.J.N.: "Analysis of multiple pulse position modulation when operating over graded-index, plastic optical fibre", IEE Proc.-Optoelectron., vol.151, no.6, December 2004.

10. LEE, G.M., SCROEDER, G.W.: "Optical Pulse Position Modulation with Multiple Positions per Pulsewidth", IEEE Trans. Commun., vol. COM-25, pp. 360-365, 1977
11. GOL'DSTEYN, A. and FREZINSKIY, B.: "An Investigation of the Transmission of a Multi-Position PPM Optical Signal Through a Communications Line Containing Repeaters", Radio. Eng. Electron Phys., vol.24, pp.65-71, 1978.
12. HERO, M.A., HU, L.: "Multi-pulse PPM and a new look at coding for direct detection optical channels using APD receivers", Proceedings of the 23rd Conference on Communication, Control and Computing, IL, USA: University of Illinois, pp 401-410, 1985.
13. SUGIYAMA, H. and NOSU, K.: "multiple PPM: A Method for Improving the Band-Utilization Efficiency in Optical PPM", Journal of Lightwave Technology, vol. 7, no.3, pp 465-472, 1989.
14. PARK, H. and BARRY, R.: "Performance analysis and channel capacity for multiple-pulse position modulation on multipath channels", IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, pp 247-251, 1996.
15. PARK, H. and BARRY, R.: "Partial-response precoding scheme for multiple pulse-position modulation", IEEE Proc. Opto., vol. 150, no. 2, pp 133-137, 2003.
16. PARK, H. and BARRY, R.: "Trellis-Coded Multiple-Pulse-Position Modulation for Wireless Infrared Communications", IEEE Trans. Commun., vol. 52, no. 4, pp. 643-651, April 2004.
17. SIBLEY, M.J.N.: "Dicode pulse-position modulation: a novel coding scheme for optical-fibre communications", IEE Proc. Optoelectronics, vol.150, no.2, pp 125-132, April 2003.
18. SIBLEY, M.J.N.: "Analysis of dicode pulse position modulation using a PINFET receiver and a slightly/highly dispersive optical channel", IEE Proc. Optoelectronics, vol.150, no.3, pp 205-209, June 2003.
19. ZWILLINGER, D.: "Differential PPM has a higher throughput than PPM for the bandlimited and average power limited channel", IEEE Trans. of Inform. Theory, vol. IT-34, Issue 5, Pt.2, pp 1269-1273, 1988.

20. D. SHIU and KAHN, J.M.,: "Differential Pulse-Position Modulation for Power-Efficient Optical Communication", IEEE Trans. in Communications, vol. com-47, Number 8, pp 1201-1210, August 1999.

21. SHALABY, H.M.H.: "A performance analysis of optical overlapping PPM-CDMA communication systems", IEEE Journal of Light. Tech., vol. 17, no. 3, pp. 426-434, 1999.

22. ATKIN, G.E. and FUNG, K.S.: "Coded multipulse modulation in optical communication systems", IEEE Trans. in Communications, vol. com-42, Number 3, pp 574-582, March 1994.

23. SHIN, B.-G., PARK, J.-H. and KIM, J.-J.: 'Low-loss, high-bandwidth graded-index plastic optical fiber fabricated by the centrifugal deposition method', Applied Physics Letters, vol. 82, no. 26, pp. 4645-4647, 2003.

PCM bits		Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Errors/PCM Bit
First Pulse Left Error (2,3,5)	[2,3]	0	0	1	0	1	1	
	[2,5]	0	0	1	1	0	1	
	[3,5]	0	1	0	1	1	0	
AVERAGE		0/3	1/3	2/3	2/3	2/3	2/3	
		0	0.33	0.66	0.66	0.66	0.66	
Average PCM	[2,3,5]	0	0	1	1	1	1	
Original Word	[3,5]	0	1	0	1	1	0	
Error Bits	XOR	0	1	1	0	0	1	**3/6**
First Pulse Right Error (4,5)	[4,5]	0	1	1	1	1	0	
Original Word	[3,5]	0	1	0	1	1	0	
Error Bits	XOR	0	0	1	0	0	0	**1/6**
Second Pulse Left Error (3,4,5)	[3,4]	0	1	0	1	0	1	
	[3,5]	0	1	0	1	1	0	
	[4,5]	0	1	1	1	1	0	
AVERAGE		0/3	3/3	1/3	3/3	2/3	1/3	
		0	1	0.33	1	0.66	0.33	
Average PCM	[3,4,5]	0	1	0	1	1	0	
Original Word	[3,5]	0	1	0	1	1	0	
Error Bits	XOR	0	0	0	0	0	0	**0/6**
Second Pulse Right Error (3,6)	[3,6]	0	1	0	1	1	1	
Original Word	[3,5]	0	1	0	1	1	0	
Error Bits	XOR	0	0	0	0	0	1	**1/6**
Av. Error/PCM Bit for [3,5]								**5/24=0.20833**

Table 1: Decoding for Wrong-Slot Error using MLSD

Error	False-Alarm	Wrong-Slot	Erasure	
Sequences	***0***	***1***	***1***	Standard errors
	1***0***	***I***1	***I***1	
	11***0***	1***1***	1***1***	
	1***0***1	11***1***	11***1***	
	11***0***1	10***1***	10***1***	
		110***1***	110***1***	
		10***I***1	1010***1***	
		10***I***11		
		110***I***1		
		110***I***11		
		1***I***1		
		1***I***11		
		11***I***1		
		11***I***11		

Table 2: Sequences considered from the simplified algorithm with symbol in error being represented in *italics* (*0* and *1* alone are called standard errors)

fn	Best sensitivity (photons per pulse)		Optimum coding level	Worst sensitivity (photons per pulse)		Optimum coding level
	Simplified	Complete		Simplified	Complete	
100	2763	2763	12-2	2845	2853	12-10
90	2770	2770	12-2	2854	2855	12-10
80	2782	2782	12-2	2868	2871	12-10
70	2798	2798	12-2	2884	2886	12-10
60	2821	2821	12-2	2933	2935	12-10
50	2853	2853	12-2	2951	2953	12-10
40	2903	2903	12-2	2989	2993	12-10
30	2973	2973	12-2	3077	3086	12-10
20	3103	3103	12-2	3204	3205	12-10
18	3149	3149	12-2	3252	3253	12-10
15	3240	3240	12-2	3348	3348	12-10
12	3373	3373	12-2	3487	3488	12-10
10	3501	3501	12-2	3621	3622	12-10
9	3585	3585	12-2	3710	3710	12-10
8	3689	3689	12-2	3819	3820	12-10
7	3823	3823	12-2	3960	3961	12-10
6	4004	4004	12-2	4148	4149	12-10
5	4257	4257	12-2	4411	4413	12-10
4	4642	4642	12-2	4810	4812	12-10
3	5312	5312	12-2	5500	5504	12-10
2	6994	6994	12-2	7232	7237	12-10
1.8	7744	7744	12-2	8006	8010	12-10
1.5	10030	10030	12-1	11910	11910	12-10
1.2	42380	42380	12-1	101000	101000	12-10

Table 3 Best and worst case sensitivity (using the simplified and complete models) and corresponding coding level for varying normalised channel bandwidth

fn	Normalised sensitivity	Normalised BE	Coding level
100	0.971	0.5	12-2
90	0.971	0.5	12-2
80	0.970	0.5	12-2
70	0.970	0.5	12-2
60	0.962	0.5	12-2
50	0.967	0.5	12-2
40	0.971	0.5	12-2
30	0.966	0.5	12-2
20	0.969	0.5	12-2
18	0.968	0.5	12-2
15	0.968	0.5	12-2
12	0.967	0.5	12-2
10	0.967	0.5	12-2
9	0.966	0.5	12-2
8	0.966	0.5	12-2
7	0.965	0.5	12-2
6	0.965	0.5	12-2
5	0.965	0.5	12-2
4	0.965	0.5	12-2
3	0.966	0.5	12-2
2	0.967	0.5	12-2
1.8	0.967	0.5	12-2
1.5	0.842	1	12-1
1.2	0.420	1	12-1

Table 4: Normalized sensitivity, BE and optimum coding level of a 12-Y MPPM system

	BE is more important					Equal Weight	Sensitivity is more important				
fn	**0-100**	**10-90**	**20-80**	**30-70**	**40-60**	**50-50**	**60-40**	**70-30**	**80-20**	**90-10**	**100-0**
100	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-2
90	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-4	12-2
80	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-2	12-2
70	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-4	12-2
60	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-2	12-2
50	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-2	12-2
40	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-2	12-2
30	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-2	12-2
20	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-2	12-2
18	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-2	12-2
15	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-2	12-2
12	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-2	12-2
10	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-2	12-2
9	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-2	12-2
8	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-2	12-2
7	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-2	12-2
6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-2	12-2
5	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-2	12-2
4	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-2	12-2
3	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-2	12-2
2	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-2	12-2
1.8	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-6	12-2	12-2
1.5	12-5	12-5	12-5	12-5	12-5	12-5	12-5	12-2	12-2	12-2	12-1
1.2	12-5	12-5	12-5	12-5	12-5	12-5	12-1	12-1	12-1	12-1	12-1

Table 5: Efficiency map (as defined by equation 9) for a 12-Y multiple PPM system

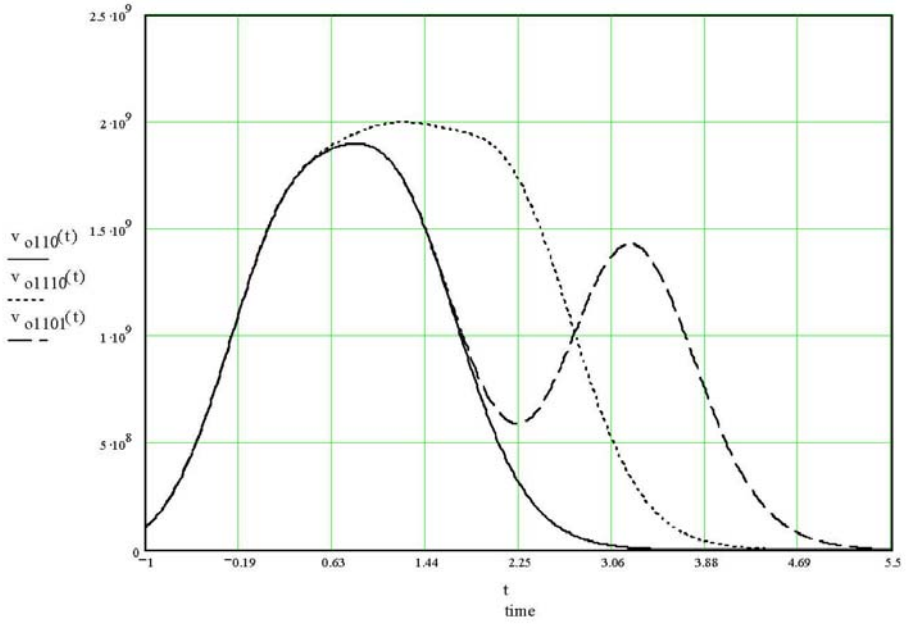


Figure 1: 1110, 110 and 1101 pre-detection multiple PPM sequences with f_n=1.2

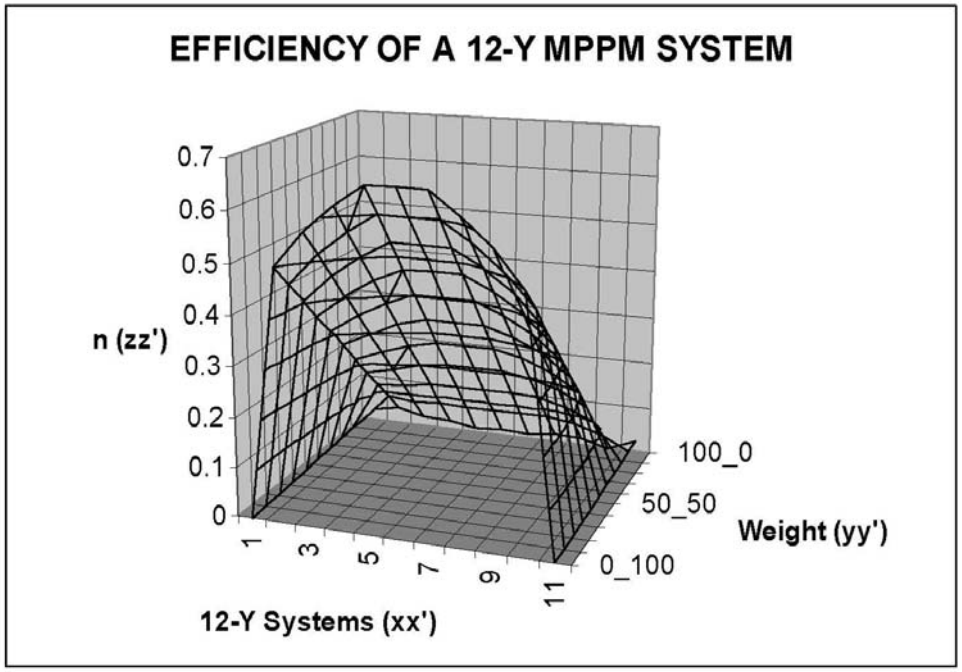


Figure 2: Efficiency (surface weighted sum) plot of a 12-Y multiple PPM system for a normalized bandwidth of 30 (in the yy' axis from 0 to 49, Bandwidth expansion is dominant from 51 to 100 sensitivity is dominant with equality in 50)

Theoretical Investigation into the Effects of Data Mapping in an Optical multiple PPM Link

K. Nikolaidis, M.J.N. Sibley

Department of Engineering and Technology

School of Computing and Engineering

The University of Huddersfield, Queensgate, Huddersfield, HD1 3DH

The effects of linear increment, linear decrement, Gray code and random mapping of data on the performance of a $\binom{12}{Y}$ *multiple PPM system are presented. Simulations using show that the Gray code is the most effective as it minimizes the Hamming distance between adjacent multiple PPM words.*

Introduction: Digital Pulse Position Modulation (PPM) is a modulation format that codes n bits of PCM into a single pulse that occupies one of 2^n time slots. The improvement in receiver sensitivity results in a considerable increase in the final data rate of the original PCM [3], and this makes implementation difficult. Alternative methods have been proposed [1-6] of which multiple PPM [3-6], the subject of this paper, is the most efficient in that it offers the best sensitivity [3] without the large bandwidth increase. Sugiyama and Nosu [5] proposed a detailed noise performance of a $\binom{12}{2}$ multiple PPM, optical fibre system in the presence of erasure errors. A Maximum Likelihood Sequence Detector (MLSD) was used as the decoder-detector, and this same scheme is also used in this paper. They concluded that multiple PPM is more efficient than digital PPM in terms of power and bandwidth

utilization, resulting in a best predicted sensitivity of 0.58 bits/photon compared to the 0.5 bits/photon for digital PPM, both operating at an error rate of 1 in 10^{-9}. They also considered the effects on system sensitivity of randomizing the mapping of the PCM data to multiple PPM words – a technique that is also examined in this paper.

In this paper, we present original detailed results obtained from mapping experiments on a $\binom{12}{Y}$ multiple PPM system using MLSD. The mappings considered are Linear Increment, LI, Linear Decrement, LD, Gray Code, GC, and Random, RD. With the LI code being taken as reference, we show that the other codes give an improvement in sensitivity without introducing redundancy.

Pulse Detection Errors: As with digital PPM, multiple PPM systems suffer from three types of error, erasure, false alarm and wrong-slot [7,8]. In case of an error the number of resultant PCM errors is found using a MLSD in the same way as that used by Sibley [3]. In order to calculate the error probability and the effects of Inter-Symbol Interference, ISI, and Inter-Frame Interference, IFI, Sibley [3] considered specific pulse sequences such as *0* and *1* (called standard error) and 1*0*, 11*0*, *1*1, 1*1*, 1*0*1, 10*1* and 10*1*1 caused by ISI or IFI of adjacent slots with the symbol in error being represented in *italics*. The error probability was determined by applying the MLSD decoding and then weighting by the probability that the particular sequence occurs. This process can be very tedious and time consuming (420 combinations need to be considered just for FA errors in a $\binom{12}{2}$ multiple PPM system) especially if the system is large such as $\binom{22}{5}$. Hence, a software solution, using novel algorithms, was developed as previously described [9]. Use of this software greatly simplified the task of examining the effects of different coding techniques on the PCM error rate.

Mathematical Analysis: The same mathematical models were used in the same way as that used by Sibley [3]. The total equivalent PCM error probability obtained, by adding together the individual probabilities, was taken to be 1 error in 10^9 pulses. For a given sets of parameters, the pulse shapes, derivates and the noise were found and the number of photons per bit, b, calculated. A threshold parameter, v, was defined as

$$v = \frac{v_d}{v_{pk}} \qquad (1)$$

v_{pk} = the peak voltage of an isolated pulse

All the experiments considered transmission through plastic optical fibre with a Gaussian impulse response [4]. The channel bandwidth was normalised to the PCM bit-rate and varied between 100 and 1.2. Operation below 1.2 was impossible due to the high levels of ISI and IFI causing sequences such as 110, 1101 and 1110 to be received as 10, 101 and 10 respectively.

Results and Discussion: Table 1 details the variation in the number of photons per PCM bit as the normalised channel bandwidth, f_n, varies from 100 to 1.2 for $\binom{12}{Y}$ systems using LI. It is apparent from the table that the optimum coding level ranges from 12-2 for high bandwidths to 12-1 for low bandwidths. Moreover, sensitivity is further degraded when the number of pulses is increased. Sugiyama and Nosu [5] only considered random mapping of data to find the optimum coding technique. Here we present results for four different mappings – Linear Increment, Linear Decrement, Gray Code and Random – that operate without redundancy for every version of the $\binom{12}{Y}$ family. The

results are shown in table 2 which details the improvement in sensitivity (photons/PCM bit) obtained over Linear Increment mapping (taken as the reference). Table 3 presents the variation of optimum mapping as the channel normalised bandwidth, f_n, varies. From these results it is evident that eight systems (from $\binom{12}{1}$ t $\binom{12}{6}$ and $\binom{12}{9}$ and $\binom{12}{11}$) enhance their sensitivity if Gray Codes are used. Two systems ($\binom{12}{7}$ and $\binom{12}{8}$) enhance their sensitivity if Linear Increment/Decrement is used (the LI and LD mappings gave almost the same sensitivity) and Random mapping was most efficient only fo $\binom{12}{10}$. Thus it can be seen that Gray Codes are the most efficient mapping for most systems of th $\binom{12}{Y}$ family. This can be explained from the fact that Gray Codes minimize the Hamming distance between adjacent multiple PPM words, and MLSD is used. Hence, if the Hamming distance is kept to minimum, there are minimum errors between the decoded word and the original data.

As an example consider the codeword [2,4] of the $\binom{12}{2}$ MPPM system, which decodes to the 001100 equivalent PCM word (if LI mapping is used). If a WS error occurs on the first or second pulse of the codeword, 7 error bits are generated. This means an average error/PCM bit for [2,4] of 0.29. With Gray coding, [2,4] decodes as 001010 and the number of errors is reduced to 6. The average error/PCM bit is reduced to 0.25. Other results were generated for the rest of the sequences and the other error types in a similar fashion.

Conclusion: This paper has examined the effects of linear increment, linear decrement, Gray code and random coding on the error performance of a $\binom{12}{Y}$ multiple PPM system coding 1 Gbit/s PCM data.

The receiver/decoder uses slope detection and a maximum likelihood sequence detector (MLSD) together with slope detection. By performing a series of mapping experiments, it was shown that the Gray code gives optimum sensitivity for most systems of the 12-Y family. It does this by minimizing the Hamming distance between adjacent multiple PPM words, which reduces the likelihood of a decoding error in the MLSD. None of these codes considered resulted in any additional data being transmitted.

References

1. SIBLEY, M.J.N.: "Dicode pulse-position modulation: a novel coding scheme for optical-fibre communications", IEE Proc. Optoelectronics, vol.150, no.2, pp 125-132, April 2003.
2. ZWILLINGER, D.: "Differential PPM has a higher throughput than PPM for the bandlimited and average power limited channel", IEEE Trans. of Inform. Theory, vol. IT-34, Issue 5, Pt.2, pp 1269-1273, 1988.
3. SIBLEY, M.J.N.: "Analysis of multiple pulse position modulation when operating over graded-index, plastic optical fibre", IEE Proc.-Optoelectron., vol.151, no.6, December 2004.
4. HERO, M.A., HU, L.: "Multi-pulse PPM and a new look at coding for direct detection optical channels using APD receivers", Proceedings of the 23rd Conference on Communication, Control and Computing, IL, USA: University of Illinois, pp 401-410, 1985.
5. SUGIYAMA, H. and NOSU, K.: "multiple PPM: A Method for Improving the Band-Utilization Efficiency in Optical PPM", Journal of Lightwave Technology, vol. 7, no.3, pp 465-472, 1989.
6. ATKIN, G.E. and FUNG, K.S.: "Coded multipulse modulation in optical communication systems", IEEE Trans. in Communications, vol. com-42, Number 3, pp 574-582, March 1994.
7. GARRETT, I.: "Digital pulse-Position Modulation over dispersive optical fibre channels", Presented at Int. Workshop on Digital communications, Tirrenia, Italy, 15-19 August 1983.
8. GARRETT, I.: 'Digital pulse-position modulation over slightly dispersive optical fibre channels', International symposium on *Information theory*, St. Jovite, 1983, pp. 78-79
9. NIKOLAIDIS, K and SIBLEY, M.J.N.: "Investigation of an Optical Multiple PPM Link over a Highly Dispersive Optical Channel", accepted IET Optoelectronics Paper

Linear Increment mapping											
Sensitivity (photons/PCM bit)											
Fn	**12_1**	**12_2**	**12_3**	**12_4**	**12_5**	**12_6**	**12_7**	**12_8**	**12_9**	**12_10**	**12_11**
100	932	921	1199	1401	1567	1877	2190	2802	3610	4740	10234
90	934	923	1202	1405	1572	1883	2197	2811	3622	4755	10267
80	938	927	1207	1411	1579	1891	2207	2823	3637	4777	10314
70	944	933	1214	1419	1588	1901	2219	2839	3658	4803	10373
60	952	940	1224	1431	1601	1917	2238	2863	3689	4843	10457
50	962	951	1239	1448	1619	1941	2263	2896	3731	4898	10575
40	979	968	1260	1473	1648	1973	2303	2947	3797	4985	10762
30	1003	991	1292	1511	1692	2027	2366	3027	3901	5128	11081
20	1047	1034	1348	1576	1764	2113	2466	3155	4065	5340	11528
18	1062	1050	1368	1600	1791	2145	2503	3202	4127	5420	11700
15	1093	1080	1408	1646	1843	2207	2576	3296	4248	5580	12041
12	1137	1124	1466	1714	1919	2299	2683	3432	4424	5812	12540
10	1180	1167	1522	1780	1993	2387	2786	3565	4594	6035	13020
9	1209	1195	1559	1823	2042	2445	2854	3651	4706	6183	13336
8	1244	1230	1605	1877	2102	2517	2938	3759	4845	6365	13728
7	1289	1274	1664	1946	2179	2610	3047	3897	5023	6600	14230
6	1350	1335	1743	2038	2283	2734	3191	4082	5261	6913	14898
5	1435	1419	1854	2168	2429	2909	3395	4342	5595	7352	15833
4	1565	1547	2023	2365	2650	3173	3703	4735	6101	8017	17237
3	1790	1771	2315	2706	3032	3629	4237	5415	6975	9167	19657
2	2356	2331	3047	3561	3993	4775	5575	7119	9162	12053	25663
1.8	2608	2581	3373	3941	4421	5285	6173	7880	10140	13343	28347
1.5	3347	3533	4766	5665	6433	7793	9170	11650	14953	19850	37547
1.2	14127	27390	39450	47555	54461	66327	78167	99250	126900	168333	316653

Table 1: Sensitivity in photons/PCM bit of a 12-Y MPPM system using a Linear Increment mapping

Maximum improvement in sensitivity (photons/PCM bit)											
fn	**12_1**	**12_2**	**12_3**	**12_4**	**12_5**	**12_6**	**12_7**	**12_8**	**12_9**	**12_10**	**12_11**
100	-2	0	-2	-2	-1	-4	0	0	-3	-18	-18
90	-2	0	-1	-1	-1	-5	0	0	-3	-18	-18
80	-2	0	-2	-2	-1	-5	0	0	-3	-18	-22
70	-2	0	-1	-1	-1	-4	0	0	-3	-18	-22
60	-2	0	-1	-2	-1	-5	0	0	-3	-18	-22
50	-2	0	-1	-2	-1	-7	0	0	-3	-18	-18
40	-2	0	-2	-2	-1	-5	0	0	-3	-18	-22
30	-3	0	-2	-2	-1	-5	0	0	-3	-20	-22
20	-3	0	-2	-2	-1	-5	0	0	-4	-20	-22
18	-3	0	-2	-2	-1	-5	0	0	-4	-20	-22
15	-3	0	-2	-2	-1	-5	0	0	-4	-22	-22
12	-4	0	-2	-2	-1	-5	0	0	-4	-23	-26
10	-4	0	-2	-2	-1	-6	0	0	-4	-23	-26
9	-4	0	-2	-2	-1	-6	0	0	-4	-25	-29
8	-5	0	-2	-2	-1	-6	0	0	-4	-25	-33
7	-5	0	-2	-2	-1	-6	0	0	-5	-27	-37
6	-6	0	-2	-2	-2	-7	0	0	-5	-28	-40
5	-7	0	-2	-2	0	-7	0	0	-4	-30	-48
4	-9	0	-3	-3	-2	-8	0	0	-5	-33	-55
3	-10	0	-3	-3	-2	-9	0	0	-5	-38	-88
2	-12	0	-4	-3	-3	-12	0	0	-4	-55	-187
1.8	-13	0	-5	-3	-3	-13	0	0	-5	-62	-220
1.5	0	-30	-26	-95	0	-20	-8	-10	-26	-133	0
1.2	0	-793	-321	-1630	-6	-173	-156	-80	-450	-1500	0

Table 2: Maximum improvement of sensitivity in photons/PCM bit in contrast to Linear Increment mapping. A zero number means that Linear Increment remains the most efficient mapping

12-Y multiple PPM EFFICIENCY MAP											
fn	**12_1**	**12_2**	**12_3**	**12_4**	**12_5**	**12_6**	**12_7**	**12_8**	**12_9**	**12_10**	**12_11**
100	GC	GC/LI	GC	GC	GC	GC	LI	LI	GC	RD	GC
90	GC	GC/LI	GC	GC	GC	GC	LI	LI	GC	RD	GC
80	GC	GC/LI	GC	GC	GC	GC	LI	LI	GC	RD	GC
70	GC	GC/LI	GC	GC	GC	GC	LI	LI	GC	RD	GC
60	GC	GC/LI	GC	GC	GC	GC	LI	LI	GC	RD	GC
50	GC	GC/LI	GC	GC	GC	GC	LI	LI	GC	RD	GC
40	GC	GC/LI	GC	GC	GC	GC	LI	LI	GC	RD	GC
30	GC	GC/LI	GC	GC	GC	GC	LI	LI	GC	RD	GC
20	GC	GC/LI	GC	GC	GC	GC	LI	LI	GC	RD	GC
18	GC	GC/LI	GC	GC	GC	GC	LI	LI	GC	RD	GC
15	GC	GC/LI	GC	GC	GC	GC	LI	LI	GC	RD	GC
12	GC	GC/LI	GC	GC	GC	GC	LI	LI	GC	RD	GC
10	GC	GC/LI	GC	GC	GC	GC	LI	LI	GC	RD	GC
9	GC	GC/LI	GC	GC	GC	GC	LI	LI	GC	RD	GC
8	GC	GC/LI	GC	GC	GC	GC	LI	LI	GC	RD	GC
7	GC	GC/LI	GC	GC	GC	GC	LI	LI	GC	RD	GC
6	GC	GC/LI	GC	GC	GC	GC	LI	LI	GC	RD	GC
5	GC	GC/LI	GC	GC	LI	GC	LI	LI	GC	RD	GC
4	GC	GC/LI	GC	GC	GC	GC	LI	LI	GC	RD	GC
3	GC	GC/LI	GC	GC	GC	GC	LI	LI	GC	RD	GC
2	GC	GC/LI	GC	GC	GC	GC	LI	LI	GC	RD	GC
1.8	GC	GC/LI	GC	GC	GC	GC	LI	LI	GC	RD	GC
1.5	GC/LI	GC	GC	RD	LI	GC	GC	RD	RD	RD	LI
1.2	GC/LI	GC	GC	RD	RD	GC	GC	RD	RD	RD	LI

Table 3: Variation in optimum mapping as the channel normalised bandwidth, f_n, varies from 100 to 1.2 for a 12-Y MPPM system. The mappings considered are Linear Increment, LI, Linear Decrement, LD, Gray Code, GC, and Random, RD

Optimum Mapping in an Optical Multiple PPM link using a Maximum Likelihood Sequence Detection Scheme

K. Nikolaidis, M.J.N. Sibley

Department of Engineering and Technology

Huddersfield University, Queensgate, Huddersfield, HD1 3DH

Abstract

The performance analysis of any multiple PPM system is extremely time-consuming due to the effects of different detection errors on the multiple PPM alphabet. In a small multiple PPM system such as $\binom{12}{2}$, there are 64 codewords and each one can have 10 possible false alarms, 2 possible erasures and 4 possible wrong slot errors. The situation is worse if higher order codes are considered.

This paper presents a novel algorithm that reduces the time taken to predict the sensitivity of a multiple PPM system from almost 2 hours of analysis to under a second. Results obtained using this method agree with those obtained using a full mathematical model. We also present a methodology that obtains a close to optimum mapping for any multiple PPM system. Detailed results show the effectiveness of this mapping routine.

1. Introduction

Digital Pulse Position Modulation (digital PPM) is a modulation format that codes n bits of PCM into a single pulse which occupies one of 2^n time slots [1-7]. Unfortunately this coding scheme results in excessive bandwidth expansion [7] and this has led to many alternative schemes being developed. Of these, multiple PPM has received attention because it reduces the bandwidth expansion effect by using several pulses in a data frame [8-14].

Sugiyama and Nosu [11] presented a detailed noise analysis of $\binom{12}{2}$ multiple PPM operating in the presence of erasure errors with an optical fibre link. A Maximum Likelihood Sequence Detector (MLSD) was used as the decoder-detector. This same scheme is also used in this paper. They also proposed a method of finding the optimum mapping of PCM words to multiple PPM words. By constantly changing the allocation of multiple PPM words to PCM words, they calculated the PCM error probability and, depending on the rate of change of this probability, they changed the mapping until there were no further reductions in error probability. This resulted in the optimum mapping. The analysis did not take into account pulse dispersion and wrong-slot errors.

Sibley [12] analysed the performance of $\binom{12}{2}$ multiple PPM when operating over a dispersive optical fibre channel. The effects of Inter-Symbol Interference (ISI) and Inter-Frame Interference (IFI) were taken into account as well as erasure, false alarm and wrong slot detection errors. Specific data sequences were examined to determine the equivalent PCM error rate. A hybrid 2/3 pulse system was analysed that reduced the effects of pulse dispersion on the error rate.

Nikolaidis and Sibley [13] proposed a novel automated solution that simplified the task of predicting the equivalent PCM error rates of specific sequences. A MLSD was used as the decoder-detector. They also proposed a measure of coding quality that accounts for efficiency of coding and bandwidth expansion. Using this measure, results obtained from a mathematical analysis showed that, over a wide range of channel bandwidths, a $\binom{12}{6}$ system is the most efficient of the $\binom{12}{Y}$ multiple PPM systems. In [14] they showed how the performance of a $\binom{12}{Y}$ multiple PPM system is affected by linear increment, linear decrement, Gray code and random mapping of PCM data to multiple PPM words. Calculations showed that the Gray code is the most effective as it minimizes the Hamming distance between adjacent multiple PPM words.

In this paper, we present a fully automated methodology that further simplifies the original method of predicting the sensitivity of any multiple PPM, MLSD system. It operates by identifying those multiple PPM sequences which, when in error, have the greatest impact on the PCM error rate. These sequences are then used to predict the system error rate and sensitivity. As a result, the new analysis is considerably faster than that previously described [13]. We also present a novel methodology that minimises the effect of these dominant error sequences by changing the mapping of PCM codewords to multiple PPM codewords to generate the optimum mapping.

2. Pulse Detection Errors

As with digital PPM, multiple PPM systems suffer from three types of error – erasure, false alarm and wrong-slot [1-7]. An erasure error results in the detected multiple PPM word having one less pulse in

the frame; false alarm errors result in an extra pulse in the frame; wrong-slot errors cause pulses to appear either before or after the current slot. In all cases, the MLSD must average all possible codewords to generate the most likely sequence [12].

This whole process can be very tedious and time consuming: for the $\binom{12}{2}$ system, 420 combinations need to be considered just for false alarm errors and analysis becomes very difficult with larger systems. In [13] a software solution, using novel algorithms, greatly simplified the analysis of any multiple PPM system and aided in determining the effects of different coding techniques on the PCM error rate. The next section details a new technique for finding those multiple PPM codewords that have a major impact on the PCM error rate. These can then be used to find the system sensitivity.

3. A novel methodology of predicting a close to optimum equivalent PCM mapping

In order to simplify the task of predicting the error rate of a multiple PPM system, the codewords (and hence equivalent PCM words) that are most affected by detection errors can be identified and their effects minimised by coding. The original methodology described here identifies these dominant errors which are then used in another algorithm that determines the optimum coding technique for a particular multiple PPM system. The reduction in processing time that comes from this methodology is significant.

3.1 Dominant error sequences in multiple PPM

Figure 1 is a data flow diagram showing how the dominant error sequences are identified. Each error sequence is isolated in turn by setting its equivalent PCM error rate to one while the other sequences

are set to zero. Thus only one error source contributes to the system performance at any one time. The software previously described in [13] is then used to find the resulting photons/bit for the desired error rate. This is then multiplied by the probability of the particular error sequence occurring to give the "sequence sensitivity". By considering each error sequence in turn and summing the sequence sensitivities, the total number of photons per bit can be found and the sensitivity of the system can be easily calculated. Comparisons with the full mathematical model show that this method is accurate. Tests on other $\binom{12}{Y}$ systems have shown that the dominant error sources and sequence sensitivities are the same in these other systems and so this methodology can be applied to multiple PPM systems in general. This method of identifying the dominant error sequences and determining the subsequent sequence sensitivity means that the simulation time is reduced considerably as shown in Table 1. Thus the effects of different mappings can be determined very rapidly so making analysis much faster.

The most significant error sequences were found to be those involving an erasure with the next being those with a false alarm. The wrong slot sequences are only significant when there is considerable pulse dispersion caused by a normalised fibre bandwidth, f_n, of less than 2. As erasure sequences are the most dominant sequences, the next section describes how to find an optimum, or close to optimum, mapping that minimises their effect.

3.2 Optimum mapping in multiple PPM

In a $\binom{12}{2}$ multiple PPM system, 6 PCM bits can be encoded. In order to minimise the error probability, the Hamming Distance (HD) between all code-words, referred to as the Total Hamming

Distance (THD), should be minimised. Taking the all zero code-word as reference, the PCM codewords have the following properties:

i) 6 codewords with HD of 1
ii) 15 codewords with HD of 2
iii) 20 codewords with HD of 3
iv) 15 codewords with HD of 4
v) 6 codewords with HD of 5
vi) 1 codeword with HD of 6

The mapping of PCM codewords onto the multiple PPM codewords must be chosen to minimise the THD. The three sources of error need to be considered in turn, and their effects on the THD minimised by reducing their individual HD.

Consider the $\binom{12}{2}$ detected sequence [1,x] where $12 \geq x \geq 2$. Erasure of any of the second pulses will cause the MLSD detector to receive only the pulse in slot 1 so giving the [1,?] condition. The received codeword could have been any of the 11 codewords in the [1,x] sequence and so the MLSD will average the equivalent PCM data of these codewords to give a PCM word of 000000. Taking this as the reference, table 2 shows that the Erasure Sequence Hamming Distance, ESHD, is equal to 17 for both linear increment and Gray coded data. To obtain the optimum coding regime for this [1,?] condition, PCM codewords should be chosen and assigned to the multiple PPM codewords so that the ESHD, and hence the THD, are minimised. If [1,2] is assigned to 000000, there are 6 other codewords with HD of 1 and a possible 4 out of 15 other codewords with HD of 2. Table 2 shows the resulting "optimum" mapping for this particular sequence. The ESHD is now reduced to 14 which is an 18% reduction compared to linear mapping. The erasure sequences [1,?] to [6,?] can also have an ESHD of 14; however the ESHD increases for codes above [6,?] as shown in table 3. This is because the choice of remaining PCM code-words is limited due to the fact that the lowest HD ones have already been allocated. Thus the MLSD must average the remaining PCM code-words, which have a higher HD,

and this results in a large HD for the erasure codewords [7,?] to [12,?]. A complication occurs with the erasure codewords [10,?] and [11,?]. Below [10,?] the number of code-words to be averaged is 11 (table 3) and so the MLSD has an odd number of 1s or 0s to consider when making a decision and there is no ambiguity in the result. However, when [10,?] and [11,?] are received, the MLSD will randomly allocate a 1 or a 0 to a data bit because the number of 1s and 0s is the same. This random allocation will generate a large number of errors and so the optimum mapping should avoid this.

Minimisation of the ESHD means that the errors from some false alarm and wrong-slot sequences will also be reduced. Consider the sequence [1,2]. A false alarm will generate a sequence [1,2,x], where $12 \geq x > 2$. The output of the MLSD will be the average of three PCM codewords corresponding to [1,2], [1,x] and [2,x]. If $8 \geq x > 2$, the code-words [1,2] and [1,x] will have a HD of 1 and so only the [2,x] code-word will have to be chosen (preferably with a HD of 2). If $12 \geq x > 8$, only the code-word [1,2] has a HD of 1 while the other two code-words [1,x] and [2,x] have a HD of 2. In general, for [1,x,y] code-words, where $11 \geq x > 1$, $12 \geq y > 2$ and $x > y$, two out of three codewords have a maximum HD of 2. The missing codewords are allocated to minimise the ESHD. As regards wrong-slot errors, some are already optimised because they appear to be pulse erasures (right shift wrong-slot to adjacent pulse for instance). For the others, Gray codes are very efficient because a wrong-slot error causes the next codeword to be chosen and, in Gray coded data, the resulting error will be one bit. However, wrong-slot errors are only significant over a small range of low bandwidths and so minimising their effect is not as important as minimising the effect of erasures.

A mapping such as this that targets erasure errors and affects a large number of false alarm and wrong-slot sequences is optimum, or very close to optimum, and table 4 shows the mapping generated. This mapping can also be adapted for larger systems. For example the mapping generated for the erasure

codeword [1,?], as shown in table 2, can be used in a $\binom{12}{3}$ multiple PPM system to generate the mapping for the erasure codeword [1,2,?] as shown in table 5. Here the $\binom{12}{2}$ mapping forms the basis of the expanded mapping with the extra bit, the Most Significant Bit (MSB), set to 0. This same $\binom{12}{2}$ mapping can be used to generate the mapping for the [2,3,?] erasure sequence but this time the MSB is set to 1 for all codewords. By using this technique, 12 erasure codewords will be generated with a minimum ESHD and a large number will be generated that are close to minimum. The remaining codewords are selected to minimise the HD as before.

4. Results and Discussion

As in a previous publication [12], the receiver system was taken to be a 1.2 GHz bandwidth PINBJT receiver with a frequency independent, input equivalent noise current spectral density of 16×10^{-24} A^2/Hz (double-sided), followed by a classical matched filter and a threshold crossing detector. The transmission medium was plastic optical fibre (POF) with a Gaussian impulse response [15] at an operating wavelength of 650 nm (corresponding to the transmission window in POF). A photodiode quantum efficiency of 100% was taken, and a PCM data rate of 1 Gbit/s was used with an error rate of 1 error in 10^9 pulses. The channel bandwidth, f_n, was normalised to the PCM bit rate and varied between 100 and 1.2. Operation below 1.2 was impossible due to the high level of ISI and IFI. The error rate was calculated for a given fibre bandwidth and coding level.

Table 6 shows the percentage change in error rate for a $\binom{12}{2}$ and $\binom{12}{3}$ multiple PPM system, using linear increment as a base and a PCM error rate of 1 bit in 10^9 bits, at normalised channel bandwidths of 100, 50, 10 and 1.2. It was not possible to obtain results for $f_n < 1.2$ due to excessive ISI and IFI. For normalised bandwidths greater than 10 it can be seen that there is little to be gained by using linear decrement or Gray code. However, if the mapping is random, there is a large deterioration in error rate for both the $\binom{12}{2}$ and $\binom{12}{3}$ systems. The effect of the optimum code can be clearly seen with significant drops in error rate particularly for the $\binom{12}{3}$ system. This is because the $\binom{12}{3}$ system has 220 possible codewords to code only 128 PCM codewords. Thus there is an element of redundancy in the code which means that the optimisation routine has a larger range of codewords available to it and this increases performance.

When operating with $f_n < 10$ there is a large degree of pulse dispersion and this leads to ISI/IFI and an increase in erasure and wrong-slot errors. Indeed, when $f_n = 1.2$, wrong-slot errors are dominant and so the Gray code offers the best performance for the $\binom{12}{2}$ system. As discussed in the previous paragraph, the $\binom{12}{3}$ system has some unused codewords which can be used to combat ISI/IFI and this leads to better performance.

5. Conclusion

This paper has described a novel method of determining the error rate performance of multiple PPM systems. Use of the routine described here identified erasure errors as having the greatest impact on the PCM error rate. Wrong Slot errors are only significant for low normalised bandwidths.

A separate routine was then used to obtain the optimum mapping of PCM to multiple PPM codewords. The routine targets those codewords that have the greatest impact on the PCM error rate (erasure and then false alarm errors) and minimises their effect. Use of this routine clearly showed the effects that optimum mapping can have on the error performance of $\binom{12}{2}$ and $\binom{12}{3}$ multiple PPM systems. This routine can be followed for other (larger or smaller) MPPM systems.

References

1. GARRETT, I.: "Digital pulse-Position Modulation over dispersive optical fibre channels", Presented at Int. Workshop on Digital communications, Tirrenia, Italy, 15-19 August 1983.
2. GARRETT, I.: 'Digital pulse-position modulation over slightly dispersive optical fibre channels', International symposium on *Information theory*, St. Jovite, 1983, pp. 78-79
3. CRYAN, R. A., UNWIN, R. T., GARRETT, I., SIBLEY, M. J. N. and CALVERT, N. M.: 'Optical Fibre Digital Pulse-Position-Modulation Assuming a Gaussian Received Pulse Shape', IEE Proceedings, Part J, Vol 137, No 2, 1990, pp 89-96.
4. MASSARELLA, A. J. and SIBLEY, M. J. N.: 'Experimental Results on Suboptimal Filtering for Optical Digital Pulse-Position Modulation', Electronics Letters, Vol 27, No 21, 1991, pp 1953-1954.
5. MASSARELLA, A. J. and SIBLEY, M. J. N.: 'Experimental Results for Optimum Coded Digital PPM with Ge APD Receiver and Gaussian Noise Approximated Theory', Electronics Letters, Vol 28, No 20, 1992, pp 1857-1858.
6. MASSARELLA, A. J. and SIBLEY, M. J. N.: 'Experimental Error Correction Results for Optical Digital Pulse-Position Modulation', Electronics Letters, Vol 28, No 18, 1992, pp 1684-1686.
7. SIBLEY, M.J.N.: 'Design implications of high speed digital PPM', SPIE conference on *Gigabit Networks*, San Jose, 1994.
8. LEE, G.M., SCROEDER, G.W.: "Optical Pulse Position Modulation with Multiple Positions per Pulsewidth", IEEE Trans. Commun., vol. COM-25, pp. 360-365, 1977

9. GOL'DSTEYN, A. and FREZINSKIY, B.: "An Investigation of the Transmission of a Multi-Position PPM Optical Signal Through a Communications Line Containing Repeaters", Radio. Eng. Electron Phys., vol.24, pp.65-71, 1978.
10. HERO, M.A., HU, L.: "Multi-pulse PPM and a new look at coding for direct detection optical channels using APD receivers", Proceedings of the 23rd Conference on Communication, Control and Computing, IL, USA: University of Illinois, pp 401-410, 1985.
11. SUGIYAMA, H. and NOSU, K.: "multiple PPM: A Method for Improving the Band-Utilization Efficiency in Optical PPM", Journal of Lightwave Technology, vol. 7, no.3, pp 465-472, 1989.
12. SIBLEY, M.J.N.: "Analysis of multiple pulse position modulation when operating over graded-index, plastic optical fibre", IEE Proc.-Optoelectron., vol.151, no.6, December 2004.
13. NIKOLAIDIS, K., SIBLEY, M.J.N.: "Investigation of an Optical Multiple PPM Link over a Highly Dispersive Optical Channel", IET Optoelectronics, June 2007, Volume 1, Issue 3, p. 113-119.
14. NIKOLAIDIS, K., SIBLEY, M.J.N.: "Investigation into the Effects of Data Mapping in an Optical multiple PPM Link", Electronics Letters, September 2007, Volume 43, Issue 19, p. 1042-1044.
15. SHIN, B.-G., PARK, J.-H. and KIM, J.-J.: 'Low-loss, high-bandwidth graded-index plastic optical fiber fabricated by the centrifugal deposition method', *Applied Physics Letters*, 2003, Volume 82, Issue 26, pp. 4645-4647

SYSTEM	SENSITIVITY ANALYSIS SCHEMES	
	NO S/W AND MATHEMATICAL ANALYSIS	ONLY S/W
12-1	<30 min	<1 sec
12-2	<110 min	<1 sec
12-3	<140 min	<1 sec
12-4	<200 min	<1 sec
12-5	<380 min	<3 sec
12-6	<380 min	<3 sec
12-7	<380 min	<3 sec
12-8	<200 min	<1 sec
12-9	<140 min	<1 sec
12-10	<110 min	<1 sec
12-11	<30 min	<1 sec
TOTAL	<2100 min	<18 sec

Table 1: Time needed to predict the efficiency of the 12-Y system using non-automated methods and only software

Erasure Error	**Mapping**		
[1,?]	**Linear Increment**	**Gray Code**	**Optimum**
[1,2]	000000	000000	000001
[1,3]	000001	000001	000100
[1,4]	000010	000011	011000
[1,5]	000011	000010	000110
[1,6]	000100	000110	001100
[1,7]	000101	000111	110000
[1,8]	000110	000101	100000
[1,9]	000111	000100	010000
[1,10]	001000	001100	000000
[1,11]	001001	001101	001000
[1,12]	001010	001111	000010
Averaged [1,?] codeword	000000	000101	000000
Sequence Hamming Distance	17	17	14

Table 2: Erasure Sequence Hamming Distance ESHD for Linear Increment, Gray Code and "Optimum" mapping of the erased [1,?] codeword in a $\binom{12}{2}$ multiple PPM system

Erasure sequence	Averaged codeword	Number of codewords	ESHD
[1,?]	000000	11	14
[2,?]	001011	11	14
[3,?]	010101	11	14
[4,?]	01111 0	11	14
[5,?]	100110	11	14
[6,?]	10110 1	11	14
[7,?]	110011	11	16
[8,?]	11100 0	11	20
[9,?]	110010	11	22
[10,?]	000000	10	21
[11,?]	101001	10	24
[12,?]	011101	9	22

Table 3: Erasure averaged codewords for a $\binom{12}{Y}$ multiple PPM system

15

Optimum mapping for the 12-2 MULTIPLE PPM System							
c/w	**PCM**	**c/w**	**PCM**	**c/w**	**PCM**	**c/w**	**PCM**
[1,2]	000001	**[2,8]**	011011	**[4,7]**	111110	**[6,10]**	101100
[1,3]	000100	**[2,9]**	001010	**[4,8]**	011010	**[6,11]**	101101
[1,4]	011000	**[2,10]**	000011	**[4,9]**	011110	**[6,12]**	001101
[1,5]	000110	**[2,11]**	001001	**[4,10]**	010110	**[7,8]**	110001
[1,6]	001100	**[2,12]**	001011	**[4,11]**	011111	**[7,9]**	110010
[1,7]	110000	**[3,4]**	010100	**[4,12]**	011100	**[7,10]**	010011
[1,8]	100000	**[3,5]**	000111	**[5,6]**	101111	**[7,11]**	110011
[1,9]	010000	**[3,6]**	111101	**[5,7]**	100010	**[7,12]**	110111
[1,10]	000000	**[3,7]**	110101	**[5,8]**	101110	**[8,9]**	111010
[1,11]	001000	**[3,8]**	010001	**[5,9]**	110110	**[8,10]**	101000
[1,12]	000010	**[3,9]**	010111	**[5,10]**	100100	**[8,11]**	111000
[2,3]	011001	**[3,10]**	000101	**[5,11]**	100110	**[8,12]**	111001
[2,4]	001111	**[3,11]**	010101	**[5,12]**	100111	**[9,10]**	010010
[2,5]	100011	**[3,12]**	011101	**[6,7]**	111111	**[9,11]**	101010
[2,6]	101011	**[4,5]**	001110	**[6,8]**	101001	**[9,12]**	110100
[2,7]	111011	**[4,6]**	111100	**[6,9]**	100101	**[10,11]**	100001

Table 4: The estimated "optimum" mapping for the $\binom{12}{2}$ multiple PPM system

Erasure Error	**Mapping**		
[1,2,?]	**Linear Increment**	**Gray Codes**	**Optimum**
[1,2,3]	0000000	0000000	0000001
[1,2,4]	0000001	0000001	0000100
[1,2,5]	0000010	0000011	0011000
[1,2,6]	0000011	0000010	0000110
[1,2,7]	0000100	0000110	0001100
[1,2,8]	0000101	0000111	0000010
[1,2,9]	0000110	0000101	0100000
[1,2,10]	0000111	0000100	0010000
[1,2,11]	0001000	0001100	0000000
[1,2,12]	0001001	0001101	0001000
Averaged [1,2,?] codeword	0000000	0000101	0000000
Averaged Hamming Distance	15	17	12

Table 5: Total and averaged Hamming distance for Linear Increment, Gray Code and "Optimum" mapping of the [1,2,?] ER averaged codeword in a $\binom{12}{3}$ multiple PPM system

17

f_n		100	50	10	1.2
LD	$\binom{12}{2}$	0.0	0.0	-0.2	-0.2
	$\binom{12}{3}$	2.8	0.0	-1.3	0.0
GC	$\binom{12}{2}$	-2.2	0.2	0.8	-50.1
	$\binom{12}{3}$	-2.4	-5.1	-5.6	-21.8
RD1	$\binom{12}{2}$	12.2	12.3	12.6	74.8
	$\binom{12}{3}$	13.6	12.5	11.0	-1.3
RD2	$\binom{12}{2}$	11.3	10.0	11.7	24.6
	$\binom{12}{3}$	10.3	10.6	9.2	5.0
RD3	$\binom{12}{2}$	4.4	7.0	6.5	-0.3
	$\binom{12}{3}$	18.0	15.2	13.7	-6.4
OPT	$\binom{12}{2}$	-20.3	-19.1	-17.0	-24.8
	$\binom{12}{3}$	-23.32	-25.3	-26.0	-30.0

Table 6: Percentage change in error rate for a $\binom{12}{2}$ and a $\binom{12}{3}$ multiple PPM system using linear increment mapping as the reference and a PCM error rate of 1 bit in 10^9 pulses. The mappings considered are Linear Decrement (LD), Gray Code (GC), Random (RD1, RD2, RD3) and Optimum (OPT).

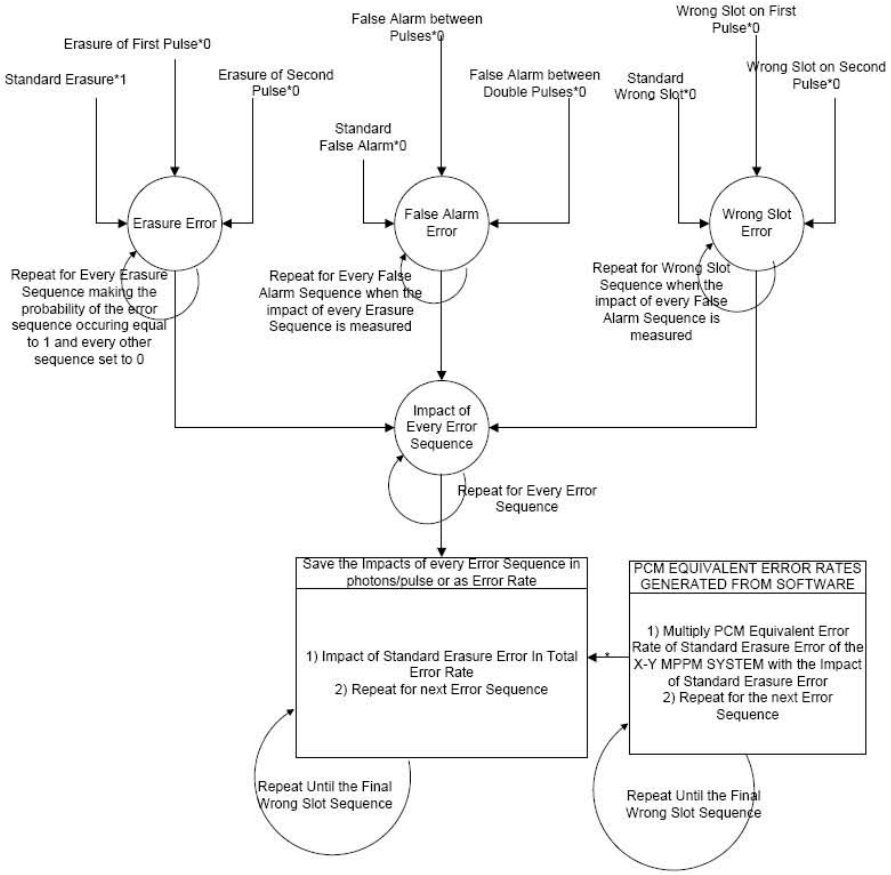


Figure 1: Data flow diagram of the optimization routine

Investigation of Higher Order Optical Multiple PPM Links over a Highly Dispersive Optical Channels

K. Nikolaidis, M.J.N. Sibley

Department of Engineering and Technology

Huddersfield University, Queensgate, Huddersfield, HD1 3DH

This paper describes a performance analysis of eight different multiple PPM systems with 4, 7, 12, 15, 17, 22, 28, 33 slots operating over a plastic optical fibre (POF) channel. The receiver/decoder uses slope detection and a maximum likelihood sequence detector (MLSD). As the analysis of any multiple PPM system is extremely time-consuming, especially when the number of slots is large, a software solution was used. A measure of coding quality that accounts for efficiency of coding and bandwidth expansion was applied and original results showed that the middle systems of these multiple PPM families are the most efficient for a wide range of bandwidths. Although the efficient Gray coding was used for this investigation, a close to optimum mapping is presented for MPPM systems of the families mentioned above. The estimated mappings were found to be much more superior to the efficient Gray codes, linear coding and a series of random mappings. Therefore, an improvement in sensitivity is demonstrated and an investigation of the bandwidth effects. To measure these optimum mappings a very efficient mapping is also considered. This mapping minimises the Hamming distance between all MPPM codewords. This mapping allows repetitions of MPPM codewords and cannot be used in a MLSD scheme. Therefore, it is only used for comparisons with the (close to) optimum mappings. It is shown that the optimum mappings are close to ideal.

Introduction: Digital Pulse Position Modulation (PPM) is a modulation format that codes n bits of PCM into a single pulse that occupies one of 2^n time slots [1, 2]. Digital PPM offers an improvement in receiver sensitivity that comes from a considerable increase in the final data rate of the original PCM [5] which makes implementation difficult. Alternative schemes have been proposed [3-9] of which multiple PPM [6-9], the subject of this paper, offers the best sensitivity without the large bandwidth increase [6]. In multiple PPM, multiple pulses per frame are used, with the pulse positions being determined by the original PCM word. If Y pulses are transmitted in a frame of X slots, the number of combinations is $\binom{n}{k} = \frac{n!}{k!(n-k)!}$. If two pulses are used in a 12 slot frame, a $\binom{12}{2}$ system, six bits of PCM can be encoded and the bandwidth expansion will be 2. The use of higher order codes means that more bits can be encoded leading to further reductions in bandwidth.

The general features of an optical fibre system employing MPPM are shown in figure 1. A PCM source of information provides the input to the system. The PCM input of M bits in a frame of duration T_f. A PCM to MPPM coder converts these M information bits into k pulses in a timeframe of n slots. In the hard decision decoding scheme, the decoder takes the first two threshold crossings as the MPPM word to be decoded. If there is only one crossing within the frame, then the word is decoded according to the MLSD decoder. The average binary error probability due to an erasure can be determined by mapping the impact of all possible erasures and averaging over all of the MPPM frames. The other error sources are treated in a similar way. The logic involved in using a MLSD in a MPPM system is showed in figure 2 when erasure errors occur.

Sugiyama and Nosu [7] proposed a detailed noise performance of a $\binom{12}{2}$ multiple PPM, optical fibre system in the presence of erasure errors. A Maximum Likelihood Sequence Detector (MLSD) was used as the decoder-detector and this same scheme is also used in this paper. They concluded that multiple PPM is more efficient than digital PPM in terms of power and bandwidth utilization, resulting in a best predicted sensitivity of 0.58 bits/photon compared to the 0.5 bits/photon for digital PPM, both operating at an error rate of 1 in 10^{-9}. Pulse dispersion and wrong-slot errors were not accounted for.

Sibley [6] analysed the performance of a $\binom{12}{2}$ multiple PPM when operating over a dispersive optical fibre channel. The effects of Inter-Symbol Interference (ISI) and Inter-Frame Interference (IFI) were taken into account as well as erasure, false alarm and wrong slot detection errors. Specific data sequences were examined to determine the equivalent PCM error rate. The author concluded that this multiple PPM scheme had a 7.36 dB advantage over PCM when operating under wide bandwidth conditions and that ISI between pulses in adjacent time slots caused the sensitivity to reduce at low bandwidths. To counter this, all multiple PPM sequences with adjacent pulses were replaced by three-pulse sequences which had at least one slot between pulses. This hybrid 2/3 pulse system gave a sensitivity of -22.74 dBm at a channel bandwidth of 0.7 times the PCM bit rate – a 3.61 dB improvement over the original two-pulse system.

Nikolaidis and Sibley [8] proposed a novel automated solution that simplified the task of predicting the equivalent PCM error rates of specific sequences when a MLSD is used as the decoder-detector. They also proposed a measure of coding quality that accounts for efficiency of coding and bandwidth

expansion. Results obtained from a mathematical analysis using this measure, showed that a $\binom{12}{6}$ system is the most efficient of the $\binom{12}{Y}$ multiple PPM family for a wide range of fibre bandwidths. In [9] they showed how the performance of a $\binom{12}{Y}$ multiple PPM system is affected by various PCM to multiple PPM mappings. Simulations showed that the Gray code is the most effective as it minimizes the Hamming distance between adjacent multiple PPM words.

In this paper, we present original results obtained from mapping experiments on the following multiple PPM families: $\binom{4}{Y}$, $\binom{7}{Y}$, $\binom{15}{Y}$, $\binom{17}{Y}$, $\binom{22}{Y}$, $\binom{28}{Y}$ and $\binom{33}{Y}$. The PCM equivalent data mapping considered for all systems was the Gray code.

Pulse Detection Errors: As with digital PPM, multiple PPM systems suffer from three types of error, erasure, false alarm and wrong-slot [1, 2]. Erasure errors are generated by noise causing the amplitude of the pulse to fall below the threshold voltage at the decision (sampling) time and so one pulse is removed from the multiple PPM frame. With a false-alarm error, noise in an empty slot causes a threshold violation at the decision time and so an extra pulse appears in the frame. Wrong slot errors occur due to noise on the leading edge of a pulse causing it to appear either before or after the current slot. This depends on the amount of pulse dispersion and the receiver noise. In all three cases, the equivalent PCM error rate is obtained using the MLSD algorithm as described by Sibley [6]. Wrong slot errors are only significant when the pulse dispersion and receiver noise is very large.

In order to study the effects of Inter-Symbol Interference, ISI, and Inter-Frame Interference, IFI, Sibley [6] considered specific pulse sequences and the error probability was determined by applying the MLSD decoding and then weighting by the probability that the particular sequence occurs. This process is very tedious and time consuming with 420 combinations needing to be considered just for false alarm errors in a $\binom{12}{2}$ multiple PPM system. The situation is worse with large systems such as $\binom{22}{5}$. To counter this, a software solution that greatly simplifies the analysis was developed by the authors [9] and this was used in this investigation.

Results and Discussion: As in a previous publication [8], the receiver system was taken to be a 1.2 GHz bandwidth PINBJT receiver with a frequency independent, input equivalent noise current spectral density of 16 x 10^{-24} A^2/Hz (double-sided), followed by a classical matched filter and a threshold crossing detector. The transmission medium was plastic optical fibre (POF) with a Gaussian impulse response [11] at an operating wavelength of 650 nm (corresponding to the transmission window in POF). A photodiode quantum efficiency of 100% was taken, and a PCM data rate of 1 Gbit/s was used with an error rate of 1 error in 10^9 pulses. POF was considered because transmission distance is limited by attenuation and the sensitivity can be increased, and hence transmission distance, by using a PPM variant such as multiple PPM. PPM systems suffer from problems with the dispersion of the pulses and so the hard decision decoding MLSD scheme was adopted to combat dispersion and inter-symbol interference. This dispersion is modelled by the channel bandwidth, f_n, normalised to the PCM bit rate with a range 100 and 1.2. Operation below 1.2 was impossible due to the high level of ISI and IFI. The number of photons per bit, b, was found given the normalised fibre bandwidth and the subsequent pulse dispersion. A measure of coding quality, η, that accounts for efficiency of coding (measured in

photons per optical pulse) and bandwidth expansion (measured as the fraction of MPPM slots over the encoded PCM bits) [8] was used for the systems studied.

Figure 3 shows a surface plot for $\binom{4}{Y}$, $\binom{7}{Y}$, $\binom{12}{Y}$ and $\binom{15}{Y}$ multiple PPM systems operating with a normalized bandwidth of 30. This bandwidth was chosen because the dispersion associated with it lies midway between the dispersion due to the 100 and 1.2 normalized bandwidths. In this figure, the efficiency factor, η, has been plotted on the vertical axis to demonstrate the optimum and it is clear that the middle coding systems are the most efficient (especially for the 50-50 point of interest where sensitivity is equally weighted with bandwidth expansion). This optimum partly occurs because the middle systems in the families can code more bits and so are considered more efficient. The exception to this is the $\binom{4}{Y}$ family, where the $\binom{4}{1}$ system is the most efficient system of the family. This is because all systems in this family can only code 2 bits of PCM and so the most efficient system is the one with the lowest number of pulses. When the sensitivity weighting is greater than 80% the optimum coding level is generally found in smaller systems.

Figure 4 shows the surface plot for the $\binom{17}{Y}$, $\binom{22}{Y}$, $\binom{28}{Y}$ and $\binom{33}{Y}$ multiple PPM families. Again it is obvious that the most efficient systems are those in the middle of the family. A comparison test was also performed for multiple PPM systems using linear mapping. The results showed that, as before, the middle systems were the most efficient. In general Gray codes gave better sensitivities than Linear,

except from some systems commonly the smaller or greater systems of the family such as $\binom{4}{1}$, $\binom{15}{3}$, $\binom{15}{13}$, $\binom{17}{1}$, $\binom{22}{16}$, $\binom{22}{20}$, $\binom{28}{1}$, $\binom{28}{4}$, $\binom{33}{1}$ to $\binom{33}{8}$ and from $\binom{33}{28}$ to $\binom{33}{31}$.

Nikolaidis and Sibley [10] described a methodology of how to obtain an optimum or close to optimum mapping for any MPPM system. The methodology is based on minimising the effects of erasure errors, because these error rates have the greatest impact on the final error probability. Starting from the all zero codeword, the mapping should try to minimise the erasure errors between the weighted codewords and the erasure MLSD weighted codeword. Hence, the choice of suitable erasure MLSD codewords is vital and very time consuming.

As an example consider the $\binom{7}{4}$ MPPM system. The system can encode 5 PCM bits and generates 35 MLSD (erasure) codewords such as [1,2,3,?], [1,2,4,?], [2,3,4,?] and so on. Where (?) symbolises an erasure error. From these 35 erasure MLSD sequences, six (6) codewords are generated from codewords used by other erasure MLSD codewords. Thus, dividing the data specimen of 32 by 26 (32-6), gives a result of 1.23. Rounding the numbers the erasure MLSD codewords are generated. For an MPPM system with 2 pulses every erasure MLSD codeword uses one codeword from every previous erasure MLSD codeword. Thus, in a $\binom{33}{2}$ MPPM system, that can encode 9 PCM bits, 33 erasure MLSD codewords are generated ([1,?] to [33,?]). Again, 6 erasure codewords are generated only from codewords used from other erasure MLSD codewords. Thus, dividing the data specimen 512 by 27 will generate the target erasure MLSD codewords. Another approach is to divide the data specimen to the codewords used from each erasure MLSD codeword. Thus, [1,?] uses 32 codewords, [2,?] 31

codewords, [3,?] 30 codewords, etc. Starting from the all zero codeword, the number of codewords used, are turned to the target erasure MLSD codewords. The second approach gave better results and was easier to follow. All the optimum mappings in this paper are generated using this approach. Figure 5 shows a part from the methodology used in a $\binom{33}{2}$ MPPM system.

Tables 1 and 2 shows the percentage change in error rate for a $\binom{7}{2}$, $\binom{7}{3}$, $\binom{7}{4}$, $\binom{12}{2}$, $\binom{12}{3}$, $\binom{12}{6}$, $\binom{15}{2}$, $\binom{15}{3}$, $\binom{15}{7}$, $\binom{17}{2}$, $\binom{17}{3}$, $\binom{17}{8}$, $\binom{22}{2}$, $\binom{22}{3}$, $\binom{22}{8}$, $\binom{22}{11}$, $\binom{28}{2}$, $\binom{28}{9}$, $\binom{28}{14}$, $\binom{33}{2}$, $\binom{33}{3}$, $\binom{33}{9}$, $\binom{33}{12}$ and $\binom{33}{16}$ multiple PPM system, using linear increment as a base and a PCM error rate of 1 bit in 10^9 bits, at normalised channel bandwidths of 100, 50, 10 and 1.2. It was not possible to obtain results for $f_n < 1.2$ due to excessive ISI and IFI. For normalised bandwidths greater than 10 it can be seen that there is little to be gained by using linear coding or Gray codes. The effect of the optimum code can be clearly seen with significant drops in error rate (in some cases above 30%) especially for large MPPM systems. This is because the large MPPM systems have a large number of possible codewords to code the PCM codewords. Thus there is an element of redundancy in the code which means that the optimisation routine has a larger range of codewords available to it and this increases performance. Even when a series of random mappings were tested there was a large deterioration in error rate for most MPPM systems.

To measure the efficiency of these mappings an "ideal" mapping is proposed. This mapping is referred as ideal because the total Hamming distance (THD) is minimized as it is demonstrated in table 3. This mapping allows repetitions of MPPM codewords and cannot be used in a MLSD scheme (used only for

comparison reasons with the optimum mappings). Table 4 shows the percentage change in error rate for a $\binom{7}{2}$, $\binom{7}{3}$, $\binom{7}{4}$, $\binom{12}{2}$, $\binom{12}{3}$, $\binom{12}{6}$, $\binom{15}{2}$, $\binom{15}{3}$, $\binom{17}{2}$, $\binom{28}{2}$ and $\binom{33}{2}$ multiple PPM system, using linear increment as a base and a PCM error rate of 1 bit in 10^9 bits, at normalised channel bandwidths of 100, 50, 10 and 1.2 using optimum and ideal mappings. The optimum mappings were generally found to have 30% to 50% lower efficiency from the ideal mapping. This consist evidence that the optimum mappings are very efficient coding.

Conclusion: This paper has examined the theoretical performance of a series of $\binom{X}{Y}$ multiple PPM families coding 1 Gbit/s PCM data with Gray code mapping. The receiver/decoder used slope detection and a maximum likelihood sequence detector (MLSD) together with slope detection. The results show, that for small multiple PPM families, the smaller systems within that family are generally the most efficient because the coding efficiency (number of bits coded) does not drastically affect the efficiency of the system. However, for higher order multiple PPM families (the majority of the systems tested) the middle systems were generally found to be the most efficient. That means that using MPPM if bandwidth expansion is an issue then middle systems of the MPPM family should be used. On the other hand smaller systems of a family are more efficient to use. Optimum mappings were also generated, for a number of MPPM systems which were found much more superior than linear and gray codes. Also, they found to be close to ideal mappings that retain minimum THD (but cannot be used because of codeword repetition). Therefore, we show that altering the PCM mapping in a certain way an enhancement of the final error probability of the MPPM system can be achieved.

References

1. GARRETT, I.: "Digital pulse-Position Modulation over dispersive optical fibre channels", Presented at Int. Workshop on Digital communications, Tirrenia, Italy, 15-19 August 1983.
2. GARRETT, I.: 'Digital pulse-position modulation over slightly dispersive optical fibre channels', International symposium on *Information theory*, St. Jovite, 1983, pp. 78-79
3. D. SHIU and KAHN, J.M.,: "Differential Pulse-Position Modulation for Power-Efficient Optical Communication", IEEE Trans. in Communications, vol. com-47, Number 8, pp 1201-1210, August 1999.
4. SIBLEY, M.J.N.: "Dicode pulse-position modulation: a novel coding scheme for optical-fibre communications", IEE Proc. Optoelectronics, vol.150, no.2, pp 125-132, April 2003.
5. SIBLEY, M.J.N.: "Design Implications of High Speed Digital PPM", SPIE Conference Proceedings, 2024, pp 342-352, 1994.
6. SIBLEY, M.J.N.: "Analysis of multiple pulse position modulation when operating over graded-index, plastic optical fibre", IEE Proc.-Optoelectron., vol.151, no.6, December 2004.
7. SUGIYAMA, H. and NOSU, K.: "multiple PPM: A Method for Improving the Band-Utilization Efficiency in Optical PPM", Journal of Lightwave Technology, vol. 7, no.3, pp 465-472, 1989.
8. NIKOLAIDIS, K., SIBLEY, M.J.N.: "Investigation of an Optical Multiple PPM Link over a Highly Dispersive Optical Channel", IET Optoelectronics, June 2007, Volume 1, Issue 3, p. 113-119.
9. NIKOLAIDIS, K., SIBLEY, M.J.N.: "Investigation into the Effects of Data Mapping in an Optical multiple PPM Link", Electronics Letters, September 2007, Volume 43, Issue 19, p. 1042-1044.

10. NIKOLAIDIS, K., and SIBLEY, M.J.N.: "Optimum Mapping in an Optical Multiple PPM link using a Maximum Likelihood Sequence Detection Scheme", IET Optoelectronics, Volume 3, Issue 1, p. 47-53, February 2009.

11. SHIN, B.-G., PARK, J.-H. and KIM, J.-J.: 'Low-loss, high-bandwidth graded-index plastic optical fiber fabricated by the centrifugal deposition method', *Applied Physics Letters*, 2003, Volume 82, Issue 26, pp. 4645-4647

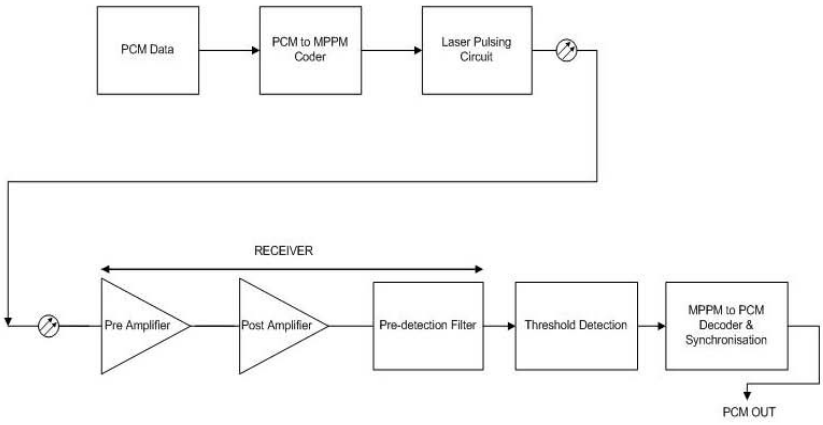


Figure 1: The optical fibre digital MPPM system features.

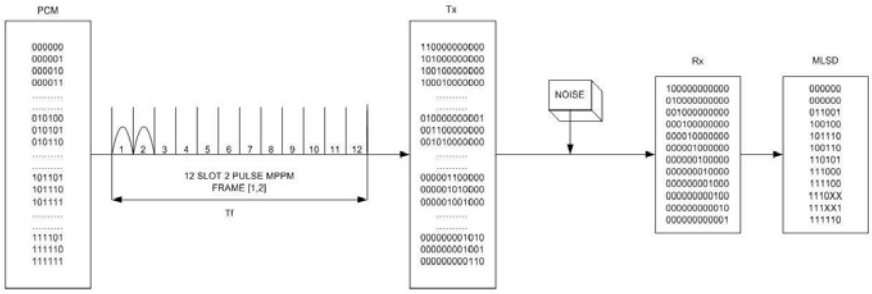


Figure 2: The logic involved in using a MLSD in a MPPM system. PCM data are turned to optical pulses. If an error occurs (i.e. an erasure) the word is decoded according to the MLSD detector.

13

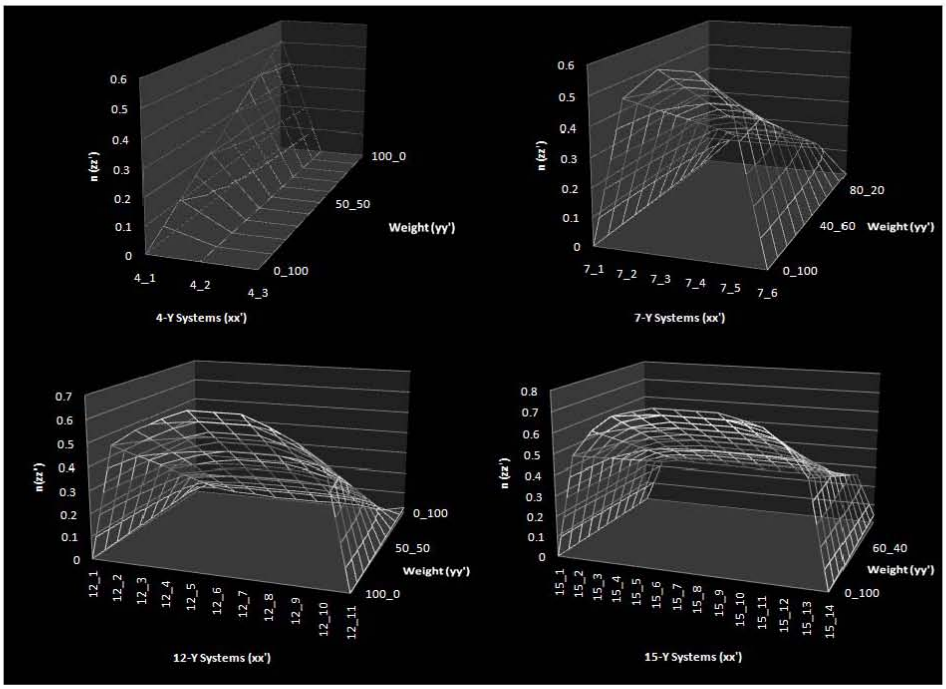


Figure 3: Efficiency (surface weighted sum) plot of a 4-Y, 7-Y, 12-Y and 15-Y multiple PPM systems for a normalized bandwidth of 30 (in the yy' axis from 0 to 49, Bandwidth expansion is dominant from 51 to 100 sensitivity is dominant with equality in 50)

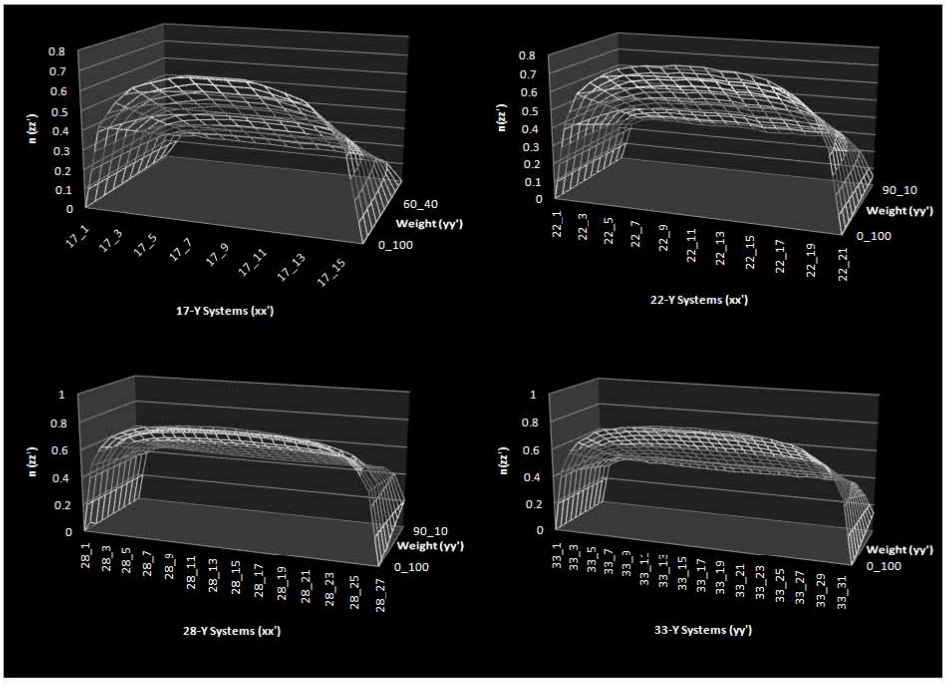


Figure 4: Efficiency (surface weighted sum) plot of a 17-Y, 22-Y, 28-Y and 33-Y multiple PPM systems for a normalized bandwidth of 30 (in the yy' axis from 0 to 49, Bandwidth expansion is dominant from 51 to 100 sensitivity is dominant with equality in 50)

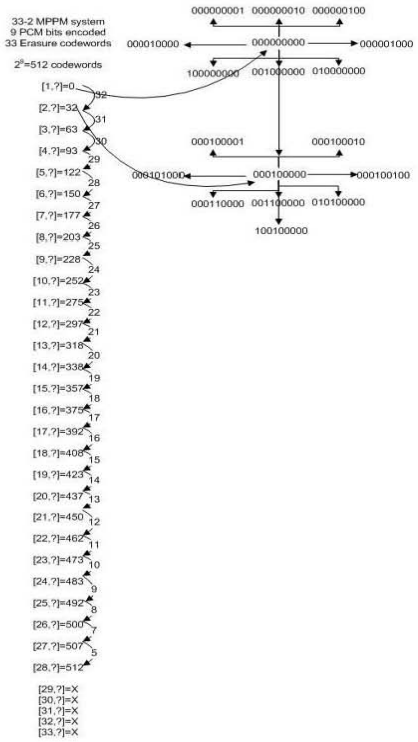


Figure 5: The methodology used to obtain optimum mapping in a $\binom{33}{2}$ MPPM system

	f_n	100	50	10	1.2	f_n	100	50	10	1.2
GC	$\binom{7}{2}$	-13.0	-12.8	-13.2	-25.6	$\binom{15}{2}$	-10.8	-10.9	-10.9	0.0
	$\binom{7}{3}$	14.2	10.5	13.9	24.8	$\binom{15}{3}$	-1.8	-1.8	-1.0	-13.6
	$\binom{7}{4}$	1.5	1.5	1.2	0.8	$\binom{15}{7}$	-4.3	-5.3	-5.6	-16.7
	$\binom{12}{2}$	-2.2	0.2	0.8	-50.1	$\binom{17}{2}$	0.0	0.3	0.0	0.0
	$\binom{12}{3}$	-2.4	-5.1	-5.6	-21.8	$\binom{17}{3}$	-4.8	-4.9	-5.4	-16.9
	$\binom{12}{6}$	-10.2	-9.2	-9.2	-8.9	$\binom{17}{8}$	-11.2	-10.2	-9.8	-19.3
OPT	$\binom{7}{2}$	-33.1	-33.1	-31.7	-65.8	$\binom{15}{2}$	-12.5	-13.5	-16.4	0.0
	$\binom{7}{3}$	-35.2	-35	-34.7	-23.7	$\binom{15}{3}$	-4.9	-5.2	-4.1	-4.7
	$\binom{7}{4}$	-23.4	-22.9	-22.3	-24.3	$\binom{15}{7}$	-6.6	-6.2	-7.1	-0.1
	$\binom{12}{2}$	-20.3	-19.1	-17.0	-24.8	$\binom{17}{2}$	-8.3	-8.9	-8.6	0.0
	$\binom{12}{3}$	-23.3	-25.3	-26.0	-30.0	$\binom{17}{3}$	-5.6	-7.6	-8.2	-7.2
	$\binom{12}{6}$	-14.2	-15.1	-16.2	-4.9	$\binom{17}{8}$	-19.1	-19.9	-20.3	0.2

Table 1: Percentage change in error rate for a $\binom{7}{2}$, $\binom{7}{3}$, $\binom{12}{2}$, $\binom{12}{3}$, $\binom{12}{6}$, $\binom{15}{2}$, $\binom{15}{3}$, $\binom{15}{7}$, $\binom{17}{2}$, $\binom{17}{3}$ and $\binom{17}{8}$ multiple PPM system using linear increment mapping as the reference and a OCM error rate of 1 bit in 10^9 pulses. The mapping considered are Gray Cide (GC) and Optimum (OPT).

f_n		100	50	10	1.2	f_n	100	50	10	1.2
GC	$\binom{22}{2}$	-10.8	-10.6	-8.8	0.0	$\binom{28}{14}$	-7.0	-9.4	-10.1	-13.7
	$\binom{22}{3}$	-3.2	-5.1	-6.7	-7.7	$\binom{33}{2}$	-10.3	-10.6	-9.8	0.0
	$\binom{22}{8}$	-8.7	-9.1	-10.3	-11.2	$\binom{33}{3}$	-7.1	-6.7	-8.8	-12.3
	$\binom{22}{11}$	-5.8	-9.3	-12.0	-11.9	$\binom{33}{9}$	-8.9	-9.4	-10.1	-11.4
	$\binom{28}{2}$	-5.2	-5.2	-4.2	0.0	$\binom{33}{12}$	-10.4	-13.7	-16.7	-17.1
	$\binom{28}{9}$	-6.1	-7.9	-8.1	-10.4	$\binom{33}{16}$	-11.1	-13.3	-15.7	-19.2
OPT	$\binom{22}{2}$	-11.7	-13.5	-16.6	0.0	$\binom{28}{14}$	-18.7	-20.7	-25.6	-4.7
	$\binom{22}{3}$	-6.7	-7.1	-8.3	0.4	$\binom{33}{2}$	-21.4	-22	-22.3	0.0
	$\binom{22}{8}$	-17.8	-19.2	-20.1	-5.6	$\binom{33}{3}$	-22.1	-24.8	-25.7	0.8
	$\binom{22}{11}$	-22.0	-21.8	-20.4	-8.9	$\binom{33}{9}$	-28.6	-29.9	-31.8	-0.5
	$\binom{28}{2}$	-6.7	-7.8	-13.2	0.2	$\binom{33}{12}$	-31.2	-33.7	-37.7	-5.1
	$\binom{28}{9}$	-10.1	-12.2	-15.7	-0.8	$\binom{33}{16}$	-33.3	-37.8	-39.2	-7.8

Table 2: Percentage change in error rate for a $\binom{22}{2}, \binom{22}{3}, \binom{22}{8}, \binom{22}{11}, \binom{28}{2}, \binom{28}{9}, \binom{28}{14}, \binom{33}{2}, \binom{33}{3}, \binom{33}{9}, \binom{33}{12}$ and $\binom{33}{16}$ **multiple PPM system using linear increment mapping as the reference and a OCM error rate of 1 bit in 10^9 pulses. The mapping considered are Gray Cide (GC) and Optimum (OPT).**

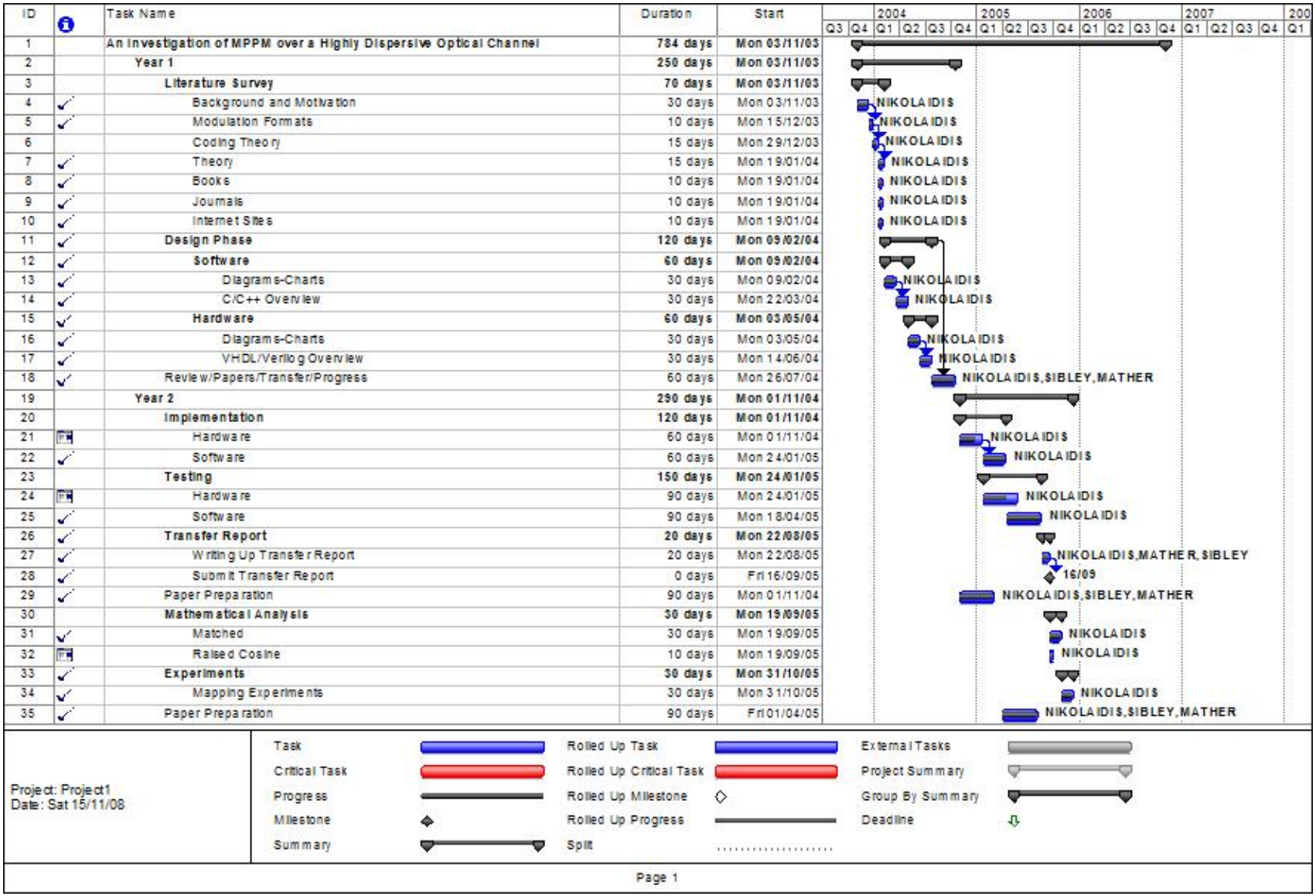
Erasure Error	**Mapping**	
[1,2,?]	**Optimum**	**Ideal**
[1,2]	000001	000000
[1,3]	000100	000001
[1,4]	011000	000010
[1,5]	000110	000100
[1,6]	001100	001000
[1,7]	000010	010000
[1,8]	100000	100000
[1,9]	010000	000001
[1,10]	000000	000010
[1,11]	001000	000100
[1,12]	010000	001000
Averaged [1,?] codeword	000000	010000
Averaged Hamming Distance	13	11

Table 3: Total Hamming distance for "Optimum" and "Ideal" mapping of the [1,?] ER averaged codeword in a $\binom{12}{2}$ multiple PPM system.

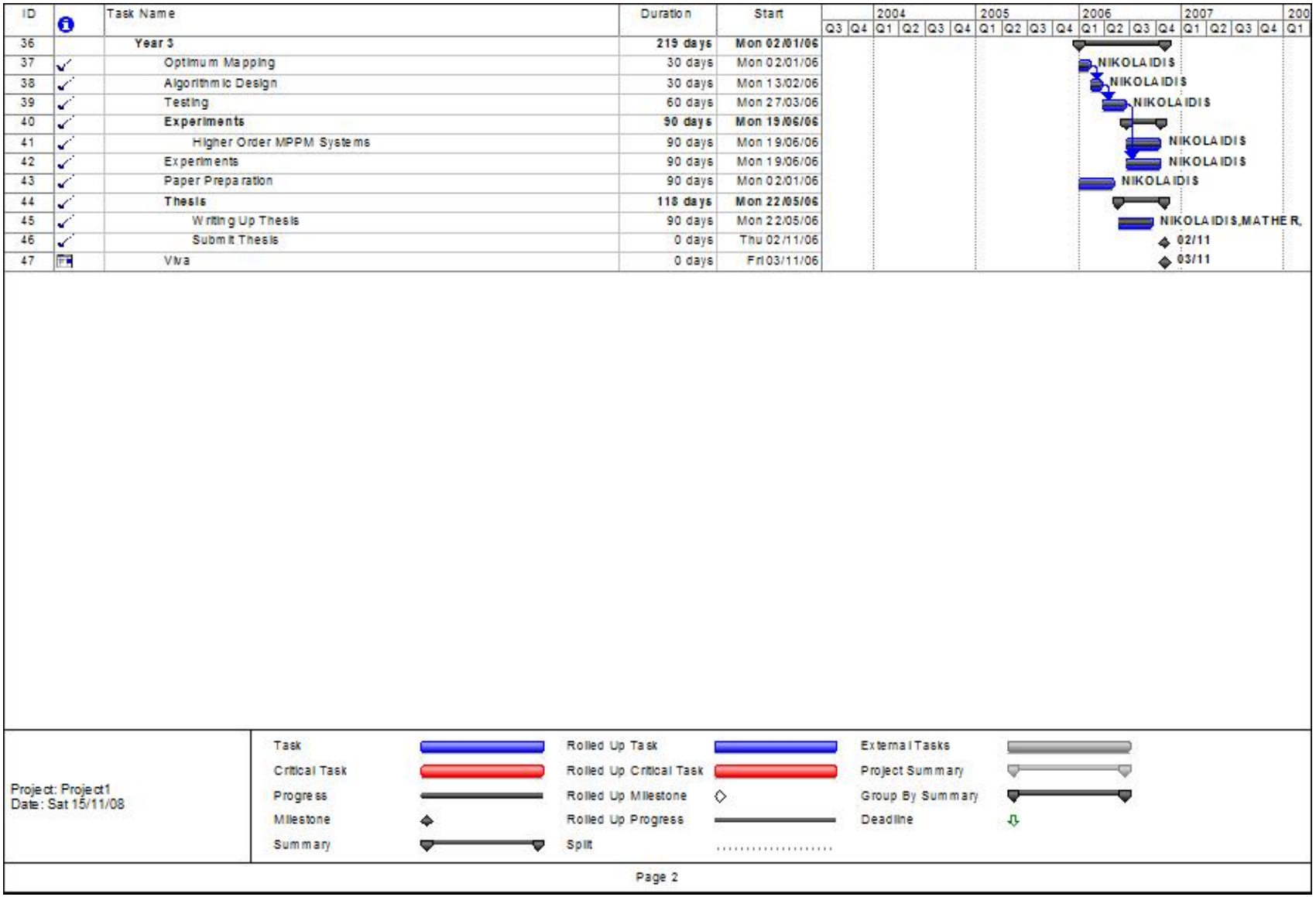
19

APPENDIX G

Project Plan



ID	Task Name	Duration	Start
1	An Investigation of MPPM over a Highly Dispersive Optical Channel	784 days	Mon 03/11/03
2	Year 1	250 days	Mon 03/11/03
3	Literature Survey	70 days	Mon 03/11/03
4	Background and Motivation	30 days	Mon 03/11/03
5	Modulation Formats	10 days	Mon 15/12/03
6	Coding Theory	15 days	Mon 29/12/03
7	Theory	15 days	Mon 19/01/04
8	Books	10 days	Mon 19/01/04
9	Journals	10 days	Mon 19/01/04
10	Internet Sites	10 days	Mon 19/01/04
11	Design Phase	120 days	Mon 09/02/04
12	Software	60 days	Mon 09/02/04
13	Diagrams-Charts	30 days	Mon 09/02/04
14	C/C++ Overview	30 days	Mon 22/03/04
15	Hardware	60 days	Mon 03/05/04
16	Diagrams-Charts	30 days	Mon 03/05/04
17	VHDL/Verilog Overview	30 days	Mon 14/06/04
18	Review/Papers/Transfer/Progress	60 days	Mon 26/07/04
19	Year 2	290 days	Mon 01/11/04
20	Implementation	120 days	Mon 01/11/04
21	Hardware	60 days	Mon 01/11/04
22	Software	60 days	Mon 24/01/05
23	Testing	150 days	Mon 24/01/05
24	Hardware	90 days	Mon 24/01/05
25	Software	90 days	Mon 18/04/05
26	Transfer Report	20 days	Mon 22/08/05
27	Writing Up Transfer Report	20 days	Mon 22/08/05
28	Submit Transfer Report	0 days	Fri 16/09/05
29	Paper Preparation	90 days	Mon 01/11/04
30	Mathematical Analysis	30 days	Mon 19/09/05
31	Matched	30 days	Mon 19/09/05
32	Raised Cosine	10 days	Mon 19/09/05
33	Experiments	30 days	Mon 31/10/05
34	Mapping Experiments	30 days	Mon 31/10/05
35	Paper Preparation	90 days	Fri 01/04/05


ID
Task Name
Duration
Start
2004
2005
2006
2007
Q3 Q4 Q1 Q2 Q3 Q4 Q1 Q2 Q3 Q4 Q1 Q2 Q3 Q4 Q1 Q2 Q3 Q4 Q1
36 Year 3 219 days Mon 02/01/06
37 Optimum Mapping 30 days Mon 02/01/06
38 Algorithmic Design 30 days Mon 13/02/06
39 Testing 60 days Mon 27/03/06
40 Experiments 90 days Mon 19/06/06
41 Higher Order MPPM Systems 90 days Mon 19/06/06
42 Experiments 90 days Mon 19/06/06
43 Paper Preparation 90 days Mon 02/01/06
44 Thesis 118 days Mon 22/05/06
45 Writing Up Thesis 90 days Mon 22/05/06
46 Submit Thesis 0 days Thu 02/11/06
47 Viva 0 days Fri 03/11/06
NIKOLAIDIS
NIKOLAIDIS
NIKOLAIDIS
NIKOLAIDIS
NIKOLAIDIS
NIKOLAIDIS
NIKOLAIDIS
NIKOLAIDIS,MATHER,
02/11
03/11
Project: Project1
Date: Sat 15/11/08
Task
Critical Task
Progress
Milestone
Summary
Rolled Up Task
Rolled Up Critical Task
Rolled Up Milestone
Rolled Up Progress
Split
External Tasks
Project Summary
Group By Summary
Deadline
Page 2