# DESIGN OF AN INFORMATION SYSTEM TO PROVIDE HOME CONTROL FOR THE ELDERLY AND DISABLED

JON WADDINGTON

A thesis submitted to the University of Huddersfield in partial fulfilment of the requirements for the degree of Master of Science by Research

The University of Huddersfield

December 2010

## Contents

Word count: 15308

**Copyright Statement**

**Abstract**

This report describes the development of an Information System which aids the elderly and disabled perform home control.

The Information System acts as a set-top box which generates video signals to display a graphical user interface (GUI) on a television. The GUI can be navigated by the user through a remote control.

The Information System can communicate with X10 modules, therefore being capable of turning lights and appliances around the home on and off.

The internal and external temperatures can also be displayed on the television screen.

A clock also keeps the time and date and these are displayed at the top of the Information System's GUI.

## 1 Introduction

The aim of this project is to design an information system which can provide home control to the elderly and disabled to realise their wish to remain independent at home while reducing health care costs [1].

Research has shown that 17% of all UK adults spend 'all or nearly all' of their leisure time at home. This rises to 37% in people aged 65 and over and increases further with people who have a disability, those aged 75 or over, and those living alone [2]. A study has shown that people over the age of 65 spend about 25% of their time watching television [3]. For this reason, their existing TV is an ideal choice to be used to display the Information System's graphical user interface.

It has been found that 15% of older people have used a mobile phone, the internet or the red button of the remote control to interact with something they have seen on the television [2]. This shows that older people can adapt to new and diverse ranges of technology and should have no problem configuring the Information System.

The research also found that 40% of people aged 65 or over claim that they make an effort to keep up with new technology, and 43% say they are interested in it. This suggests that elderly people are not afraid of becoming accustomed to a new device. Despite this, almost 70% of older people say that simple and straight forward technology is favoured [2]. This is the reason that the Information System's user interface is to be made as simple as possible.

### 1.1 Aims and Objectives

The Information System will benefit the elderly and disabled by potentially providing humanitarian and financial benefits of lowering the user's reliance on personal assistance, such as reducing care givers strain and lowering the costs of care [4].

The information system must have a simple graphical user interface with large text to allow the visually impaired to use it with ease.

The Information System's interface needs to be designed to be as user friendly as possible. The menu must be kept simple without excessive or unnecessary menus. The menu will be navigated by remote control, using just three buttons; up, down and enter. This is likely to be a familiar environment for the user, due to the use of remote controls

with televisions for years.

Ideally, no re-wiring should be needed, making the information system simple enough to be installed and customised by the user. This should also make the Information System more cost effective as a specialist is not required to install it.

### 1.2 Target Market

The Information System is aimed at assisting the elderly and disabled with performing simple tasks around the home by providing a means of home control.

Home control is expected to be a major, positive, role for the care of elderly people in their homes or in care homes. Home control can support the elderly in their own homes and still allow them to get the support they need from friends or family. This is a preferable alternative to being placed in a care home, receiving assistance from care workers. The technology can also benefit both the elderly user and the care provider [5], as it allows the elderly user to function more independently [4].

Another area to target is the physically disabled. By combining technology with personal assistance, the user's disability is modified, therefore reducing the severity of the disability. The Information System helps to modify the disability through reducing the task demands through environmental modification [4].

Whether the Information System is used to replace or to work alongside personal assistance, an whether the system is used all the time or not, the need of such a device aimed at reducing the efforts of personal assistance of the elderly and disabled is apparent [4].

### 1.3 Issues and Limitations

Although home automation should make a significant contribution to the safety, security, independence and quality of life of elderly and disabled people living in their home, there are some limitations.

Not everyone can benefit from the Information System and some will not accept it, and the needs of each user must be inspected carefully to provide greatest assistance [5].

The aim of the Information System is not to completely eliminate the need for personal

assistance from friends, family or professionals. The aim is to aid the user with simple, repetitive tasks, reducing the required personal assistance with technological assistance. This reduces the strain on the helpers and makes the user feel more independent [5].

Since the interface of the Information System is completely visual, usability is reduced in those with more serious visual impairments. Also, if the user is physically disabled and has trouble using their hands, they will have trouble using the remote control [5].

There are a number of issues which must be addressed regarding the Information System. These include the user friendliness, lack of human response and the need for customized training. However, if the Information System is designed to address the needs of the individual user, the technology has the potential to improve their lifestyle [1].

## 2 Review of Current Technology

Home automation refers to devices which are used to control elements in the home, either remotely or automatically. This can include turning lights on or off remotely, by phone, remote control or from a computer. The lights could also be set on a schedule to be turned on at night when it starts to get darker and switch off later at night while the user sleeps. This method can also be used as a deterrent for burglars by giving the impression that the residents are in the house while they are in fact absent [6].

There are currently two main technologies aimed at providing home automation. These are X10 and Z-wave, which are described below.

### 2.1 X10

X10 modules communicate through the wiring inside the building [7], meaning that no re-wiring needs to be done. This also limits the speed of communications between X10 devices, as information is only transmitted on the 0V crossover of the 50Hz AC power-line. X10 modules act as plug sockets or light sockets which can be managed by an X10 controller. The X10 controller is operated in some way by the user to communicate with the X10 modules, either turning the device plugged into the module on or off.

### 2.2 Z-Wave

Z-Wave is an alternative method of implementing home automation [8] and hasn't been around for as long as X10. Z-Wave devices work similarly to X10 devices but communicate via RF waves and are therefore not limited to the slow communication speeds by the 50Hz AC power-line.

The technology which is to be used in the Information system is X10 due to the lower price of the modules [9] and the higher availability [10].

There are already a number of existing products available which cater towards providing home automation using X10 modules. The list below contains a brief description of a selection of these products.

### 2.3 Alternative Software (HomeSeer)

HomeSeer is a software package compatible with Windows 2000 and windows XP [11]. This software can be tricky to install for elderly or handicapped people who aren't

familiar with computers and the interface is not aimed specifically aimed at them. A home automation setup using this software would also require a computer to be switched on constantly to schedule events. The software is also quite expensive at £141.00 [12] in addition to the cost of a computer if one is not already owned.

### 2.4 Easytouch Panel10

The Easytouch Panel 10, made by Marmitek, has a 10 inch touch screen to wirelessly control appliances round the home. The device is customizable and large enough to be set up for elderly and handicapped people. The device is efficient, using just 2 AAA batteries to power the screen and the software. The major disadvantage of this device is the moderately large price of £179 [13].

### 2.5 Easy Icon 10RF

The Easy Icon 10RF is a remote control with an LCD screen for controlling appliances in the home. The screen is quite small with a 176 x 220 pixel [14] resolution meaning it may be hard to see for visually impaired people. The device also has a complicated setup [15] which is not aimed at elderly or handicapped people.

As these devices are not appropriate for use by the elderly and disabled, a new device with suitable software must be developed.

### 2.6 Television Display Method

A method of displaying the Information System's user interface on a television must be decided upon.

One method of generating a user interface is to use an On-Screen-Display (OSD) chip. For example, the MAX456 [16] contains a font ROM and is capable of generating PAL and NTSC compatible video signals. This OSD chip can communicate with a microcontroller using the Serial Peripheral Interface (SPI) bus to display a menu for the Information System. This method requires two separate ICs, the OSD and a microcontroller to generate the user interface and cannot generate colour signals.

Another method of displaying the user interface could be made possible with a Complex Programmable Logic Device (CPLD), programmed to generate VGA video signals [17]. This method would allow the display of 3-bit RGB colour and this OSD would be largely customizable for the needs of the Information system. The disadvantages of this method are that it would need external ROM to hold the font ROM and there would still be a need

for a microcontroller to control the display.

The Parallax Propeller is a microcontroller which contains an internal font ROM and is capable of generating PAL and NTSC video signals [18]. The Propeller contains 8 processors, called cogs, which means that the video signal can be generated by a cog while the rest of the Propeller can manage the user interface.

The Parallax Propeller was chosen to be the central component of the Information System due to its video capabilities.

### 2.7 Radio Transmission

A method of transmitting information wirelessly must be used to transmit the external temperature to the Information System. There are two main technologies relevant to this purpose, ZigBee and Bluetooth.

Bluetooth is generally focused on the connection between mobile phones, laptops and PDAs, etc. Therefore, the data rate is high, dealing with large packets of data. Since the technology is based on devices with rechargeable batteries, the power consumption is fairly high [19].

ZigBee is generally used for control and automation. The data rate is lower and it deals with small packets of data. The aim of ZigBee is to allow the unit to work for months to years on a single set of batteries [19]. Digi International produces Xbee modules which use the ZigBee protocol [20]. This technology was chosen as the method of transmitting the external temperature in Information System.

## 3 Design of the Information System

The central component of the Information System is the Propeller chip. This is connected to all the separate features of the system, which are the television circuit, clock, temperature sensors (indoor and outdoor), the IR receiver and the X10 controller. A schematic of the Information System can be seen in figure 3.1.

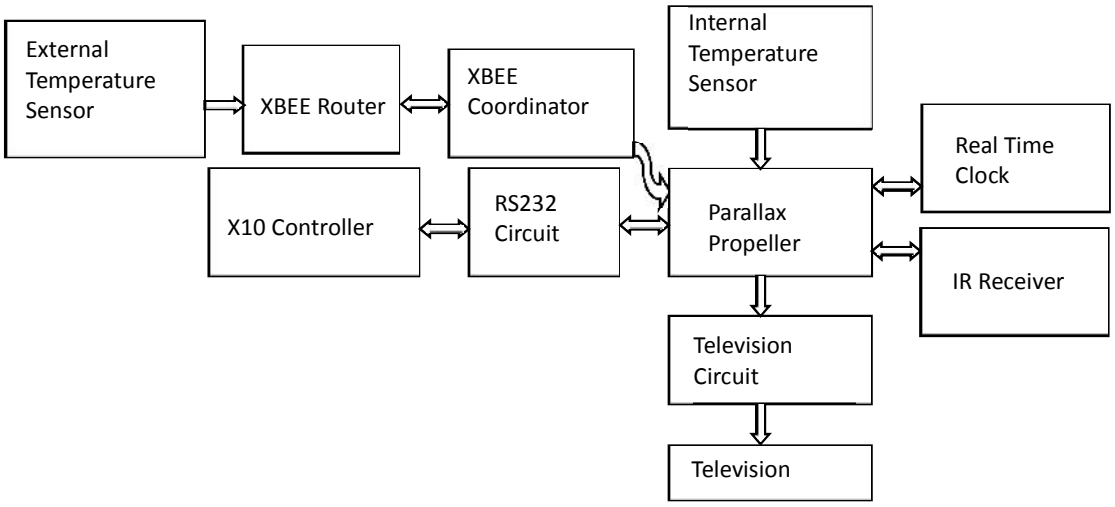


Figure 3.1. Schematic Diagram of the Information System

Figure 3.2 shows an annotated photograph of the Information System circuit. This is connected to the Propeller demo board.

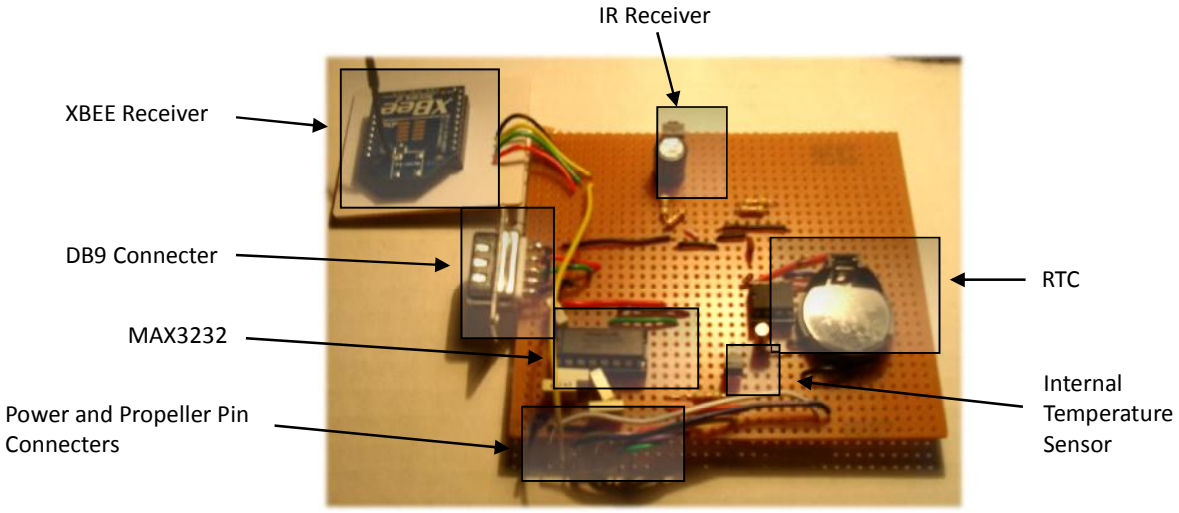


Figure 3.2. Photograph of the Information System Circuit

### 3.1 The Parallax Propeller

The Parallax Propeller is a powerful chip capable of high speed processing (up to 80MHz). The chip has eight processors, called cogs, which can perform independently or cooperatively, as the program dictates. This eliminates the need for interrupts as a cog can be dedicated to a single task, leaving the main program to continue undisturbed [21].

A Propeller application contains code written in the chip's high level SPIN language and, optionally, some Propeller Assembly (PASM) language. The SPIN code is interpreted at run time by the chips SPIN interpreter and the PASM is run directly by a cog [21].

A Propeller application contains one or more SPIN files, called objects. Each object can contain up to six different types of blocks which are shown in the table below [21].

| Block | Purpose |
|---|---|
| CON | The Constant block which defines the constants |
| VAR | The Variable block defines the global variables |
| OBJ | The Object block defines the referenced objects |
| PUB | The Public blocks contain SPIN code |
| PRI | The Private blocks contain SPIN code |
| DAT | The Data block contains PASM |

The Propeller has a memory of 64KB. This consists of 32KB of RAM and 32KB of ROM. The ROM contains the Boot Loader, SPIN Interpreter, math functions and the font ROM. The RAM is used for the Propeller application, meaning an external EEPROM must be used to store the program after the Propeller has been switched off. The Boot Loader is responsible for loading the program into the RAM from the EEPROM [21].

### 3.1.1 Propeller Demo Board

The Propeller demo board is a prototype board for the Propeller chip. It includes a Propeller chip, connected to numerous peripheral devices. The board includes audio out, composite video out, VGA out, 8 LEDs, a microphone, keyboard and mouse inputs and a USB port. There is also a breadboard on the board for prototyping and there are 8 input/output pins to add functionality.

A photograph of the Propeller board can be seen in Figure 3.1.1.

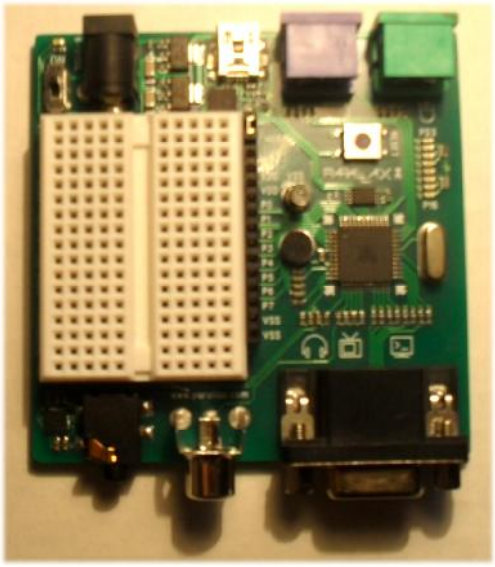

Figure 3.1.1. The Propeller Demo Board

### 3.1.2 Propeller Software

Two pieces of software were used with the Propeller chip. These were the Propeller Tool and the Parallax Serial Terminal.

#### 3.1.2.1 The Propeller Tool

The Propeller Tool provides a free development environment [22] for the Propeller chip which is capable of compiling SPIN and PASM and downloading the code to the Propeller or to the EEPROM.

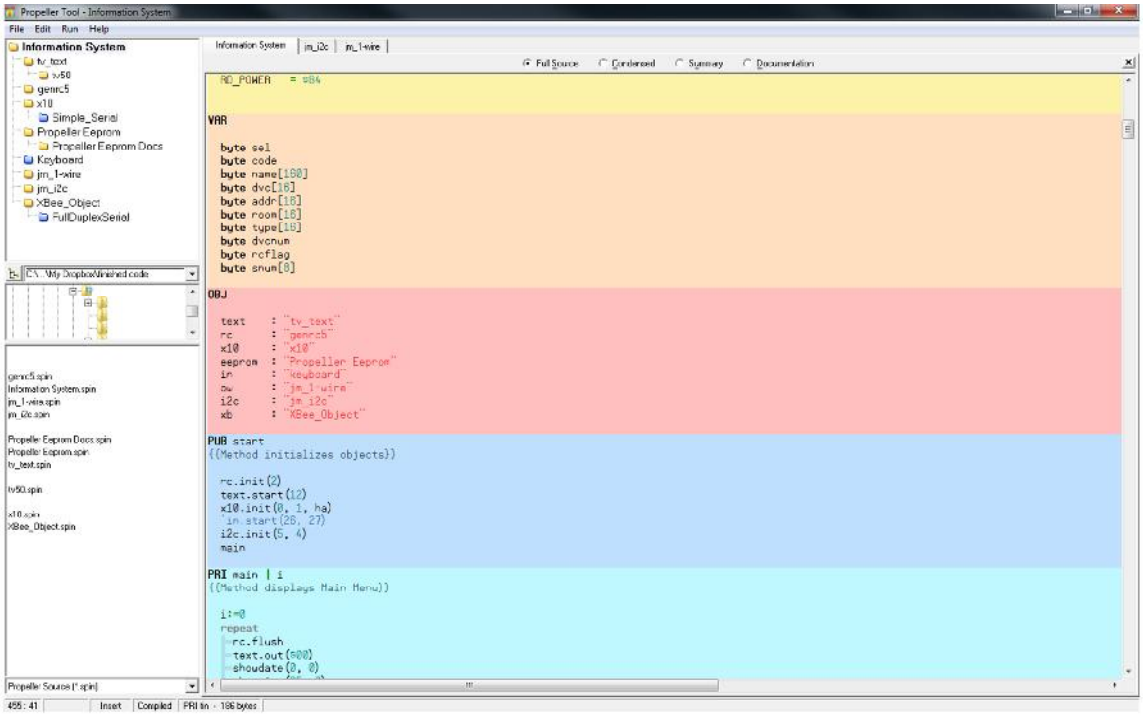


Figure 3.1.2.1 . Screenshot of the Propeller Tool

Figure 3.1.2.1 shows a screenshot of the Propeller Tool. The area on the top left of the screen shows the hierarchy of object files. The area below that is the file explorer which can be used to open object files.

The large area on the right is the editor. This is where the SPIN files are edited. It can be seen in the screenshot that different method blocks have different coloured backgrounds. This allows for the user to distinguish the separate methods easier.

Each object file opens in a new tab, meaning the user can work on many different objects simultaneously.

### 3.1.2.2 Parallax Serial Terminal

The Parallax Serial Terminal is a piece of software which can help when debugging a program. The Propeller can send information to it through the USB port, using the "Parallax Serial Terminal" object which is included in the Propeller Tool. The information is sent to the Parallax serial terminal through the same USB cable which is used to download code to the Propeller.

The Propeller Object Exchange is an online repository for community written objects for the Propeller chip. The objects have been written by Parallax engineers and customers and are for use by the Propeller community [23]. The files are provided under the MIT license, meaning they are free to use and modify, without limitations, providing the license is left in the code [24].

### 3.1.3 Code Structure

The Information System's code is structured with the top object file referring to each object separately, as shown in figure 3.1.3. The diagram also shows how each object links together and explains what each one does.

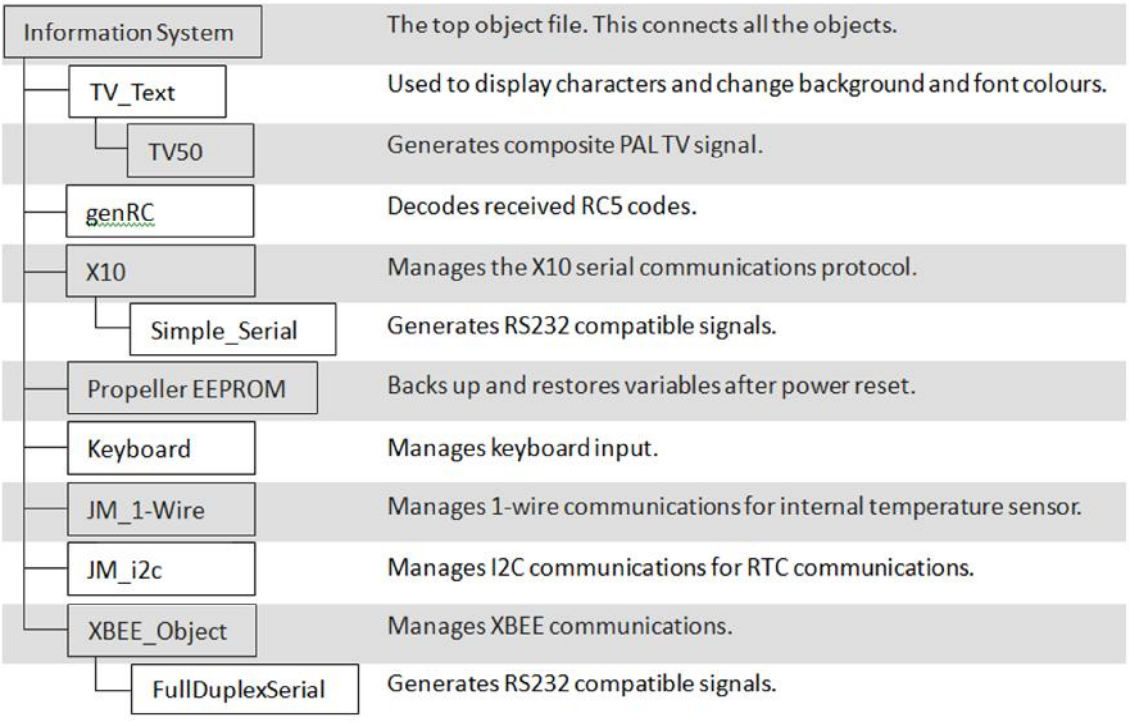


Figure 3.1.3. The Code Structure of the Information System

### 3.2 PAL Video

Video signals are displayed on a television screen by drawing each line from left to right, from top to bottom. The amplitude of the waveform represents the brightness of the screen with a high voltage being white and a low voltage being black. The video signal drops below the black level after each line has been displayed. This is the horizontal sync pulse, used to inform the television that a new line is to be drawn [25]. After all the lines have been drawn, the signal issues pulses to inform the television that the whole frame has been displayed and that the first line is to be drawn again.

In PAL (Phase Alternation Line) video systems, after the line has been displayed, the signal drops to the black level for 1.5μs. This is known as the front porch. The signal then drops for 4.7μs before rising to the black level for 5.8μs. These portions of the signal are known as the horizontal sync pulse and the back porch respectively. The back porch also contains a colour burst, a 4.43Mhz signal [26] which the television uses as a reference to display the correct colours. The full horizontal synchronisation signal is seen in figure 3.2.1. After this horizontal blanking, the active video begins. Each line of active video is 52μs in duration.

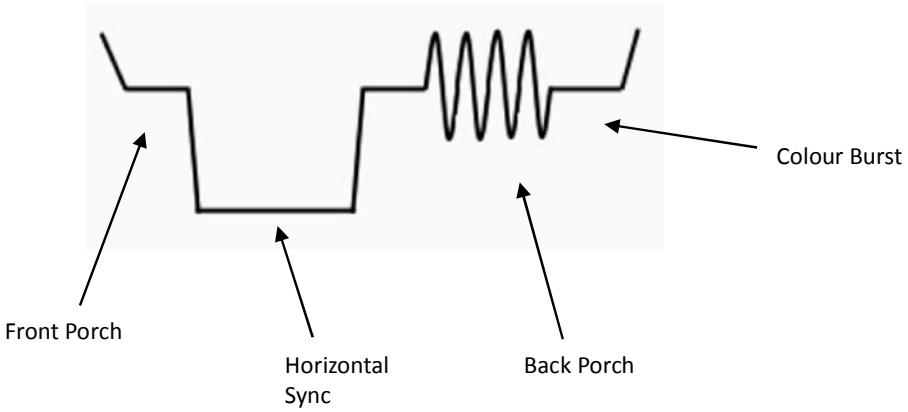


Figure 3.2.1 Horizontal Blanking Waveform

The vertical blanking consists of five equalizing pulses, five broad pulses and another five equalizing pulses. This is shown in figure 3.2.2. This informs the television that the whole frame has been drawn and the next line will be the top line [27].

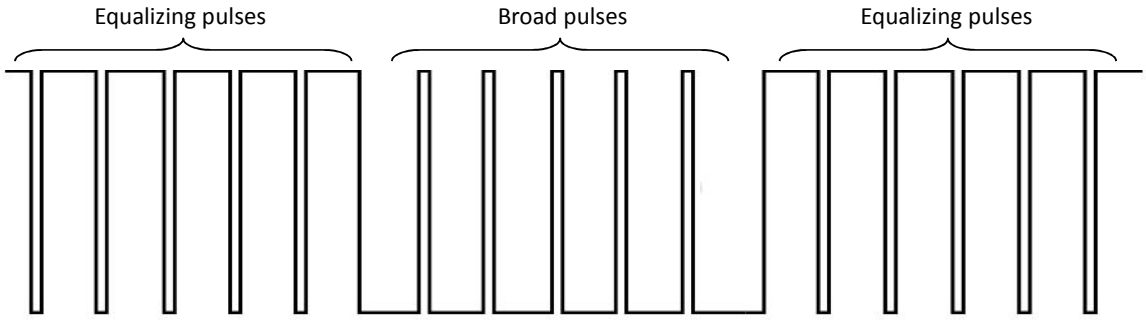


Figure 3.2.2 Vertical Blanking Waveform

Interlaced video means that two fields of alternate lines are drawn and interleaved to show a single frame. This means that there are two blanking intervals for every frame which is displayed. Interlaced scanning reduces flicker by appearing to double the frame rate from 25 frames per second to 50 frames per second [27].

Composite video is a video standard which contains luminance (Y) and chrominance (U and V) to display colour video. The Horizontal and Vertical syncs are superimposed on the same signal as the picture information.

Composite video comprises of two parts, the luminance and chrominance. The luminance (Y) contains the black and white (brightness) information of the video and the chrominance (U and V) contains the colour information. Colour is represented by 4.43MHz signals with different phases and at different amplitudes. The amplitude represents the amount of colour (saturation) and the phase represents the tint of the colour (hue) [25].

### 3.2.1 TV Cirtcuit

Each cog of the Propeller has an integrated video generator that makes generating video signals possible. Access and control of the video generator are provided by two registers, the Video Scale register and the Video Configuration register. The value in the Video Scale register determines the number of clock cycles before the next frame of data is fetched and it also indicates the number of clock cycles there are for each pixel. The Video Configuration register specifies the settings for the video signal. For example whether it is VGA or composite and what pins to output the signal on. [18]

The circuit in figure 3.2.1 shows how the chroma and luminance is combined to generate the 75Ω, 1V, baseband video signal. [18]

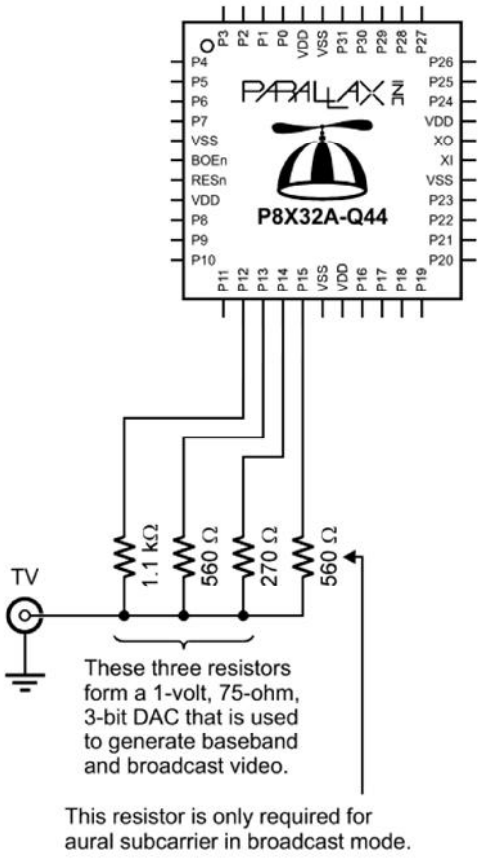


Figure 3.2.1. Schematic of the TV Circuit

The Propeller Object Exchange provided an object specifically intended to generate PAL video signals. This object was used to generate the video for the Information System.

### 3.3 Menu System

A menu is displayed on the television which shows four items for the user to choose from. Navigation of the menu is done using the Remote control. The up and down arrows control which item is selected and pressing the "stand-by" button enters the item. The current time and date is constantly displayed along the top of the menu. Figure 3.3 shows the hierarchy of the Information System's menus.

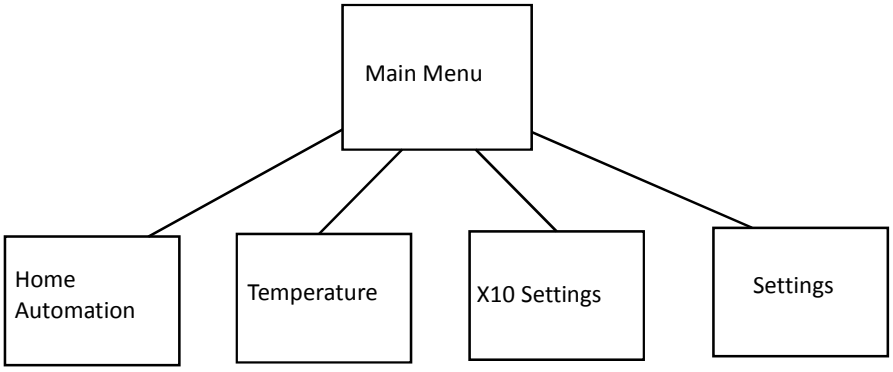


Figure 3.3. Diagram of the Menu Layout

The code below shows the "main" method which displays the main menu. This is similar to how every menu in the program is displayed.

```
PRI main
{{Method displays Main Menu}}

  repeat
      rc.flush                                          'flush RC5
      text.out($00)                                     'clear screen
      showdate(0, 0)                                    'show date and time
      showtime(35, 0)
      text.str(string(13,"        Information System"))          'display title
      text.str(string(13,13,13,"       .  Home Automation",13,13,"       .
Temperature",13,13,"       .  X10
Setup",13,13,"       .  Settings"))                   'display items
      code := 0                                         'flush code
      repeat until (code == enter)                      'repeat until enter is pressed
         case sel                                 'case statement displays cursor on items

             0 : text.str(string($0A,7,$0B,5,"> "))
                 text.str(string($0A,7,$0B,7,"2."))
                 text.str(string($0A,7,$0B,9,"3."))
                 text.str(string($0A,7,$0B,11,"4. "))
             1 : text.str(string($0A,7,$0B,5,"1."))
                 text.str(string($0A,7,$0B,7,"> "))
                 text.str(string($0A,7,$0B,9,"3."))
                 text.str(string($0A,7,$0B,11,"4. "))
```

```
        2 : text.str(string($0A,7,$0B,5,"1."))
            text.str(string($0A,7,$0B,7,"2."))
            text.str(string($0A,7,$0B,9,"> "))
            text.str(string($0A,7,$0B,11,"4. "))
        3 : text.str(string($0A,7,$0B,5,"1."))
            text.str(string($0A,7,$0B,7,"2."))
            text.str(string($0A,7,$0B,9,"3 "))
            text.str(string($0A,7,$0B,11,"> "))
    code := wait

    if ((code => 1) & (code =< 4))          'if number of item is pressed
        sel := code - 1                     'move cursor to item
    if (code == up)                         'if up is pressed
        ++sel                               'increment 'sel'
        if sel == 4                         'wrap 'sel' to 0
            sel := 0
    if (code == down)                       'if down is pressed
        sel := --sel <# 3                   'decrement 'sel' and wrap to 3

code := 0
case sel                                    'enter item
    0 : HomeAutomation
    1 : Temperature
    2 : X10Setup
    3 : Settings
```

The code begins by clearing the screen and showing the time and date. The menu is then shown with the title at the top and a numbered list of items below it. A case statement uses the “sel” variable to place the cursor on the relevant item. When a number from one to four is pressed, the “sel” variable is changed accordingly and the cursor moves. When the up or down button is pressed, “sel“ is incremented or decremented, moving the cursor. “sel” also wraps around, meaning that if it is incremented above 3, it becomes 0 and if it is decremented below 0, it becomes 3. Hence the cursor reaches the bottom and emerges from the top and vice versa.

### 3.4 Home Automation

The first item, "Home Automation", allows the user to directly control the X10 devices. A full list of devices in the Propeller's memory can be listed or, if required, the devices can be listed by the room which they are in. Once a device is selected, the user can select whether they wish to turn the device on or off. In the case of a lamp module, there is an option to dim the light. The user is then prompted to enter a value between 0 and 100 to dim the light, 0 turns the light off and 100 leaves the light on at its original brightness.

#### 3.4.1 X10

X10 is an industry standard method of communicating with electrical appliances around a building over the mains power line. The data consists of bursts of radio frequency signals to represent digital data. This method of transmitting and receiving data means that no additional wiring is required to install X10 devices when implementing home automation [28].

The data is sent from the X10 controller to the modules during the 200ms time period following a 0v transition of the AC mains power signal, as this is the point where less noise is likely to be present in the signal. The burst of data is a 120kHz signal which lasts for approximately 1ms. The data is sent another 2 times to coincide with the 0v transitions of the other phases in three-phase systems. This ensures the X-10 device can be used in industrial and commercial buildings where the three-phase system is used [29].

The information transmitted through the power-line can be separated into 3 sections. There is a 4 bit start code, which initiates the communication, there is a 4 bit house code, which is generally the address shared by all the devices in the building, and there is a 5 bit data code which may represent the address of the particular device or the required function for a device to perform.

The start code is always '1110'. This code is transmitted over the zero crossing points over 2 cycles of the AC mains signal.

### 3.4.2 Serial Interface

The CM12 X10 controller has a PC interface which uses RS232 serial communications. The controller generates X10 signals and transmits them over the power line, performing the corresponding function on the modules, depending on the user's input on the computer. The Information System uses an object to imitate the computer's serial interface and communicate with the controller.

The required voltage levels of RS232 signals are bi-polar. The voltage level of +3V to +12V indicates a logic '0' and a voltage level of -3V to -12V indicates a logic '1' [30].The Propeller uses lower voltage logic levels where a logic '1' is 3.3V and logic '0' is 0V, therefore, a voltage level converter IC is required to invert the logic level and also to convert the voltage to RS232 standards. The MAX3232 IC was chosen for this purpose. The circuit diagram is shown in figure 3.4.2. Pin 0 of the Propeller is connected to the input 'T1IN' of the MAX3232. This is the data to be transmitted. Pin 1 of the Propeller is connected to the 'R1OUT' output of the MAX3232. This is the output from the X10 controller.

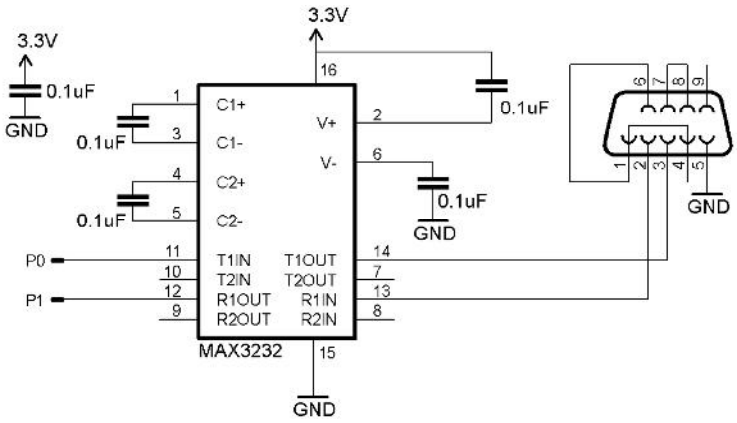


Figure 3.4.2. Circuit Diagram showing the Serial Interface

An RS232 message begins with a start bit of a logic '0' which syncs the clock of the two devices. A byte is then transmitted, starting with the least significant bit. Each bit period is the same as the start bit period [31].

A stop bit of logic '1' is sent to indicate that the message has ended. This can act as a delay before the next message is transmitted as the line idles at logic '1' until it is pulled down by the start bit.

### 3.4.3 CM12 Protocol

The serial communications protocol for the CM12 states that the baud rate is 4800 bits per second (bps). There is a start bit, a stop bit and no parity bit.

Figure 3.4.3.1, below shows how the serial communication protocol between the Propeller and the CM12 looks. The transmission is in two parts; the first part addresses the device and the second part sends the function to be performed.

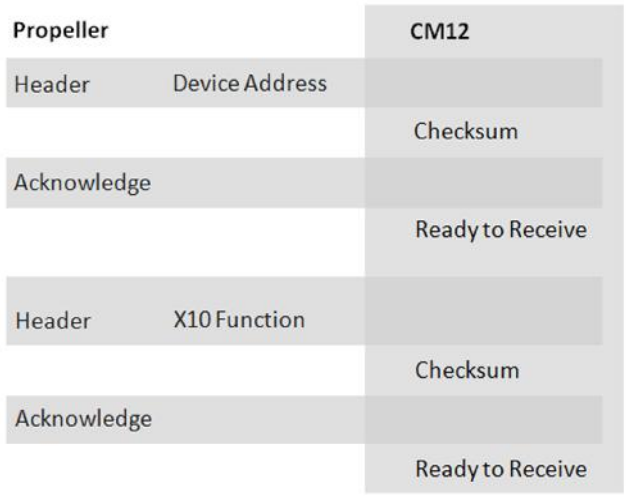


Figure 3.4.3.1. A Standard Transmission between the Propeller and the CM12

The first byte to be sent is the Header. The structure of the Header is shown in the table below. The upper 5 bits define the amount to dim the device. Bit 2 ensures that the CM12 stays synchronized by being set to '1'. Bit 1 defines whether the following byte will be the address (0) of the device or a function (1) and bit 0 defines whether the following byte will be an extended transmission (1) or a standard transmission (0) [32]. For the purpose of the Information systems, extended transmissions were not applicable.

| Header | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | Number of Dims | | | | | 1 | F/A | E/S |

The number of dims is a value between 0 and 22 where 22 is equivalent to 100% and 0 is 0% so a value of 11 would half the brightness of the light. Only specified lamp modules can be dimmed.

The next byte holds the address of the device or the function. The 4 most significant bits

contain the house code and the 4 least significant bits contain the device code or the function code.

The checksum is calculated by calculating the sum of the first 2 bytes. This is then transmitted to the Propeller which confirms that the data is correct by transmitting an acknowledge byte of '0x00'. When the CM12 is ready to receive again it transmits '0x55' to the Propeller.

The SPIN code below shows a method which handles the full transmission with the CM12 controller. The device code and function code are passed into the method and the Simple_Serial object is used to transmit and receive the signals from the CM12.

```
PUB do(device, function) | rx
{method to perform a function on X10 devices}

  repeat until rx == (($04 + (housecode + device))& $FF)    'repeat until correct
checksum received
    serial.tx($04)                                          'transmit header
    serial.tx(housecode + device)                           'transmit code
    rx := serial.rx                                         'receive checksum
    if rx == $A5                                        'if controller is polling system
      serial.tx($C3)                                                'respond to polling

  serial.tx($00)                                    'send acknowledgement of checksum
  repeat until rx == $55                          'wait until 'ready to receive' is received
    rx := serial.rx



  repeat until rx == (($06 + (housecode + function))& $FF)  'repeat until correct
checksum received
    serial.tx($06)                                          'transmit code
    serial.tx(housecode + function)                         'transmit code
    rx := serial.rx                                         'receive checksum

  serial.tx($00)                                    'send acknowledgement of checksum
  repeat until rx := $55                          'wait until 'ready to receive' is received
    rx := serial.rx
```

When the CM12 receives data from the power-line, it begins to poll the Propeller to upload its buffer. The poll signal is a byte (0xA5) which is transmitted once every second until the Propeller responds by transmitting a byte with the value of '0xC3'. While the CM12 is polling the Propeller, a transmission cannot take place.

Figure 3.4.3.2 shows the binary house codes and address codes which are assigned to the X10 devices [33]. Figure 3.4.3.3 shows the binary function codes which are transmitted to the previously addressed X10 device to perform a desired function [33].

| | House Codes | | | |
|---|---|---|---|---|
| | H1 | H2 | H4 | H8 |
| A | 0 | 1 | 1 | 0 |
| B | 1 | 1 | 1 | 0 |
| C | 0 | 0 | 1 | 0 |
| D | 1 | 0 | 1 | 0 |
| E | 0 | 0 | 0 | 1 |
| F | 1 | 0 | 0 | 1 |
| G | 0 | 1 | 0 | 1 |
| H | 1 | 1 | 0 | 1 |
| I | 0 | 1 | 1 | 1 |
| J | 1 | 1 | 1 | 1 |
| K | 0 | 0 | 1 | 1 |
| L | 1 | 0 | 1 | 1 |
| M | 0 | 0 | 0 | 0 |
| N | 1 | 0 | 0 | 0 |
| O | 0 | 1 | 0 | 0 |
| P | 1 | 1 | 0 | 0 |

| | Address Codes | | | |
|---|---|---|---|---|
| | D1 | D2 | D4 | D8 |
| 1 | 0 | 1 | 1 | 0 |
| 2 | 1 | 1 | 1 | 0 |
| 3 | 0 | 0 | 1 | 0 |
| 4 | 1 | 0 | 1 | 0 |
| 5 | 0 | 0 | 0 | 1 |
| 6 | 1 | 0 | 0 | 1 |
| 7 | 0 | 1 | 0 | 1 |
| 8 | 1 | 1 | 0 | 1 |
| 9 | 0 | 1 | 1 | 1 |
| 10 | 1 | 1 | 1 | 1 |
| 11 | 0 | 0 | 1 | 1 |
| 12 | 1 | 0 | 1 | 1 |
| 13 | 0 | 0 | 0 | 0 |
| 14 | 1 | 0 | 0 | 0 |
| 15 | 0 | 1 | 0 | 0 |
| 16 | 1 | 1 | 0 | 0 |

Figure 3.4.3.2 Table of House Codes and Device Address Codes

| | Function Codes | | | | |
|---|---|---|---|---|---|
| | D1 | D2 | D4 | D8 | D16 |
| All Units Off | 0 | 0 | 0 | 0 | 1 |
| All Units On | 0 | 0 | 0 | 1 | 1 |
| On | 0 | 0 | 1 | 0 | 1 |
| Off | 0 | 0 | 1 | 1 | 1 |
| Dim | 0 | 1 | 0 | 0 | 1 |
| Bright | 0 | 1 | 0 | 1 | 1 |
| All Lights Off | 0 | 1 | 1 | 0 | 1 |
| Extended Code | 0 | 1 | 1 | 1 | 1 |
| Hail Request | 1 | 0 | 0 | 0 | 1 |
| Hail Acknowledge | 1 | 0 | 0 | 1 | 1 |
| Pre-Set Dim | 1 | 0 | 1 | X | 1 |
| Extended Data (Analogue) | 1 | 1 | 0 | 0 | 1 |
| Status=on | 1 | 1 | 0 | 1 | 1 |
| Status=off | 1 | 1 | 1 | 0 | 1 |
| Status Request | 1 | 1 | 1 | 1 | 1 |

Figure 3.4.3.3. Table of Function codes

### 3.5 Temperature

The "Temperature" item displays the inside and outside temperatures in Celsius on the screen until the user wishes to return to the main menu. Two temperature sensors are used in the Information System. An internal temperature sensor measures the temperature inside the home and is wired directly to the Propeller chip. An external temperature sensor measures the temperature outside the home and remotely transmits the data to the Propeller chip.

#### 3.5.1 Internal Temperature

Figure 3.5.1 shows the circuit diagram for the internal temperature sensor. The DS18B20 chip [35] was used as it communicates digitally with 1 wire to the Propeller chip. A device with an analogue output was not used due to the Propeller chip not having analogue to digital conversion capabilities. The DS18B20 is capable of reading temperatures from -55°C to 125°C which is easily enough range for internal temperatures.

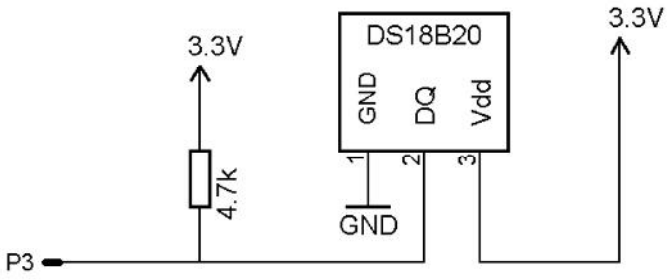


Figure 3.5.1. Circuit Diagram showing the Internal Temperature

#### 3.5.2 1-Wire Communications

To communicate with the Propeller, the DS18B20 uses a 1-Wire protocol. There are six signal types defined by this protocol. These are the Reset pulse, Presence pulse, write 1, write 0, read 1 and read 0. These signals are each initiated by the master device, in this case, the Propeller, with the exception of the Presence pulse.

To begin communicating, the Propeller pulls the 1-Wire bus down for a minimum of

480µs. This is the Reset pulse. The Propeller then releases the bus to be pulled high by the 4.7kΩ resistor. The DS18B20 responds to this with a Presence pulse by waiting 15 to 60µs for the bus to go high and then pulling it low for between 60 and 240µs. These pulses are shown in figure 3.5.2.1 [34].

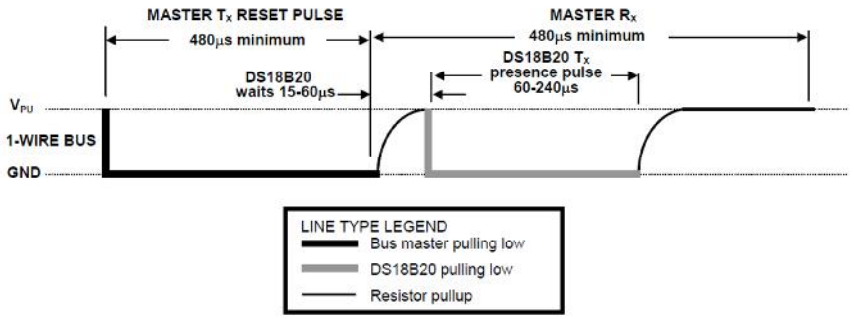


Figure 3.5.2.1. Diagram of the 1-Wire Reset and Presence Pulses

The read and write slots are issued by the Propeller and allow for one bit to be transmitted. By issuing a write slot, the Propeller is able to write data to the DS18B20. By issuing a Read time slot, the Propeller is able to read data from the DS18B20.
Each Read or Write time slot is initiated by the Propeller pulling the 1-Wire bus down. To write '0', the bus is pulled down and remains pulled down for a minimum of 60µs. To write '1', the bus is pulled low by the Propeller and released within 15µs to be pulled up by the 4.7kΩ resistor [34]. This is shown in figure 3.5.2.2.

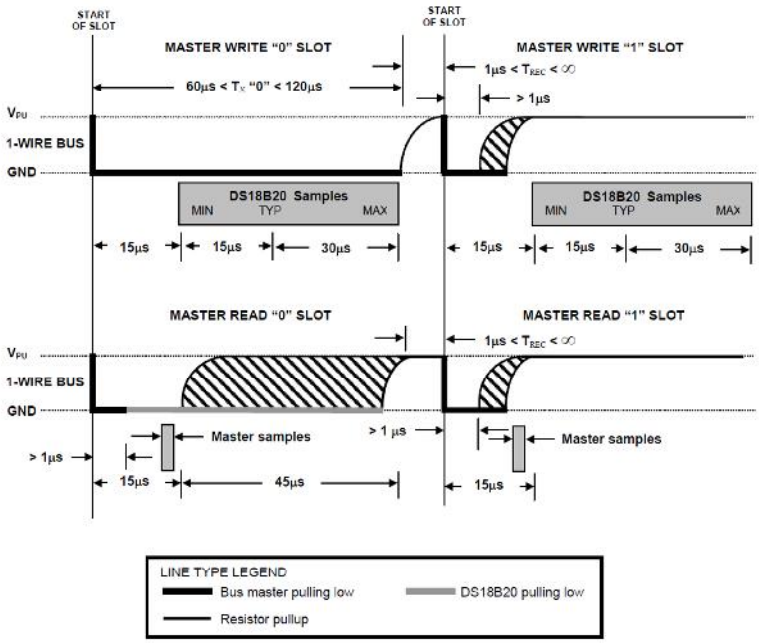


Figure 3.5.2.2 . Diagram of the Write and Read Time Slots

After the Propeller issues a Read time slot, the bus is released. The DS18B20 can then leave the bus to go high to transmit a '1' or pull the bus low to transmit a '0'. After a minimum of 60µs, the bus is released and pulled high by the 4.7kΩ resistor shown in figure 3.5.1.

The DS18B20 has a unique serial number to distinguish itself from other devices which may be present on the 1-Wire bus. The lowest byte contains the DS18B20 family's code, 0x28. The next six bytes contain the unique serial number and the most significant byte contains the cyclic redundancy check (CRC). The CRC is calculated based on the 6 byte serial number and the DS18B20 family's code. The Propeller can calculate the CRC and compare it with the read value of the CRC to verify that the data is correct. The last byte of the Scratchpad also contains a CRC which can be used to check that the information received from the Scratchpad is correct.

The DS18B20's Scratchpad is a block of ROM containing 9 bytes. The first byte holds the least significant byte of the converted temperature and the second byte contains the most significant byte of the converted temperature.
Bytes 2 and 3 are the temperature alarm triggers for high temperatures ($T_H$) and low temperatures ($T_L$) respectively. These registers are not used in the Information System.

Byte 4 is the configuration register. Bits 5 and 6 (R0 and R1 respectively) of this register are used to set the resolution of the temperature conversion. This is shown in the table below.

| R1 | R0 | Resolution (Bits) | Max Conversion Time | |
|---|---|---|---|---|
| 0 | 0 | 9 | 93.75ms | $t_{CONV}/8$ |
| 0 | 1 | 10 | 187.5ms | $t_{CONV}/4$ |
| 1 | 0 | 11 | 375ms | $t_{CONV}/2$ |
| 1 | 1 | 12 | 750ms | $t_{CONV}$ |

Bytes 5, 6 and 7 are unused and byte 8 is the Scratchpad CRC.

The temperature registers are arranged as shown in the tables below.

| Most Significant Byte | | | | | | | | Least Significant Byte | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| S | S | S | S | S | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ | $2^{-1}$ | $2^{-2}$ | $2^{-3}$ | $2^{-4}$ |

Bits 11 to 15 are the sign bits. These define whether the temperature is positive or negative. If it's positive, these bits will be logic '0'. If the temperature is negative, these bits will be logic '1', and the temperature will be the two's compliment of the temperature.

The R0 and R1 bits of the configuration register are set to 0, meaning the temperature resolution is 9 bits. This makes bits 0, 1 and 2 of the temperature redundant and the temperature can be read in steps of 0.5°C. Examples of the temperature register are shown in the table below

| Temperature | MS Byte | LS Byte |
|---|---|---|
| +30.5 | 00000001 | 11101000 |
| +25°C | 00000001 | 10010000 |
| 0°C | 00000000 | 00000000 |
| -25°C | 11111110 | 11110000 |
| -30.5 | 11111110 | 00011000 |

The Spin code below shows how the temperature is read from the DS18B20 The "tin" method initialises the 1-Wire object which starts a cog, dedicated to 1-Wire communications. The program then waits for the cog to begin and a reset pulse is transmitted. If a device is present, a CRC check is performed. If no errors are found in the CRC, the "readtc" method is called which returns the value of the DS18B20's temperature registers. This is then passed to the "showc" method, which manipulates the data to display it on the television screen. The cog is then finalized, freeing a cog to do other tasks if required.

```
PRI tin | status, temp, crc

  ow.init(3)                                  'initialize 1-wire comms, pin 3 input
  status := ow.reset                              ' check for device
  if (status == %10)                               ' good 1W reset
    crc := ow.crc8(@snum, 7)                           ' calculate CRC
    if (crc <> snum[7])                              ' compare with CRC in SN
      text.str(string("    "))
      text.hex(crc, 2)
      text.str(string(" - bad CRC"))
      repeat

    else
      temp := readtc                               ' read the temperature
      showc(temp)                                  ' display in °C

  ow.finalize                     'free a cog
```

The Spin code below shows the "readtc" method. This begins by transmitting a reset

pulse, followed by the "Skip ROM" command. This skips the ROM transmission and the code then transmits the "ConvertT" command which notifies the DS18B20 to start converting the temperature.

The code then transmits read slots on the bus and the DS18B20 continues to transmit '0' until the conversion is complete and it transmits a '1'.
The code continues by transmitting another "Skip ROM" command and the "Read Scratchpad" command. The "Read Scratchpad" command notifies the DS18B20 to begin transmitting the contents of the Scratchpad, starting with the first byte, the least significant bit of the temperature, and finishing with the CRC.

After receiving the first two bytes and joining them together, the code transmits a reset pulse which informs the DS18B20 that no more information is to be received.

```
PRI readtc | tc

'' Reads temperature from DS1820

  ow.reset
  ow.write(SKIP_ROM)
  ow.write(CVRT_TEMP)
  repeat                                    ' let conversion finish
    tc := ow.rdbit
  until (tc == 1)
  ow.reset
  ow.write(SKIP_ROM)
  ow.write(RD_SPAD)
  tc := ow.read                              ' lsb of temp
  tc |= ow.read << 8                         ' msb of temp
  ow.reset

  return tc
```

The "showc" method is shown below, this shows the temperature on the television screen in degrees Celsius. It begins by shifting the bits left 16 places as the variable is 32 bits in length. This is to check if the temperature is positive or negative. If it is negative, a minus sign is shown and the sign of the variable is changed.

The variable is shifted back to the right 16 bits and then shifted another 4 bits to discard the lower 4 bits. The temperature is then displayed on the screen, followed by "°C".

```
PRI showc (tc) | t', dp
```

```
'' Show the temperature
tc <<= 16             'shift left 16 bits
if (tc < 0)           'check temperature sign
  text.str(string("-"))       'display minus sign
  ||tc                'convert sign
tc >>= 16             'shift bits back
t := tc >> 4          'discard lower 4 bits
text.dec(t)           'display temperature
text.out($B0)                 'display degrees sign
text.out($43)                 'display 'C'
```

### 3.6 External Temperature Sensor

The external temperature circuit is powered by 4 AA batteries, giving 6V to the input of the 3.3V voltage regulator. The temperature sensor and XBEE are both powered by 3.3V.

The temperature sensor used in this circuit is an MCP9700A. This IC was chosen due to its analogue output which can be connected directly to one of the XBEE's analogue input. The MCP9700A's output voltage varies by 10mV for every 1°C change in temperature. The voltage on the XBEE's analogue input is sampled every 4 seconds and transmitted to the receiving XBEE.

#### 3.6.1 XBEE

Two XBEE Series 2 modules were used in the Information System, a receiver and a transmitter. They are designed to mount into a socket, allowing the modules to be removed with ease to place them into development kits to allow the parameters to be modified.

The schematic in figure 3.6.1.1 shows the remote temperature circuit diagram. It can be seen that the output voltage from the MCP9700A is connected to the analogue input, AD0, of the XBEE.

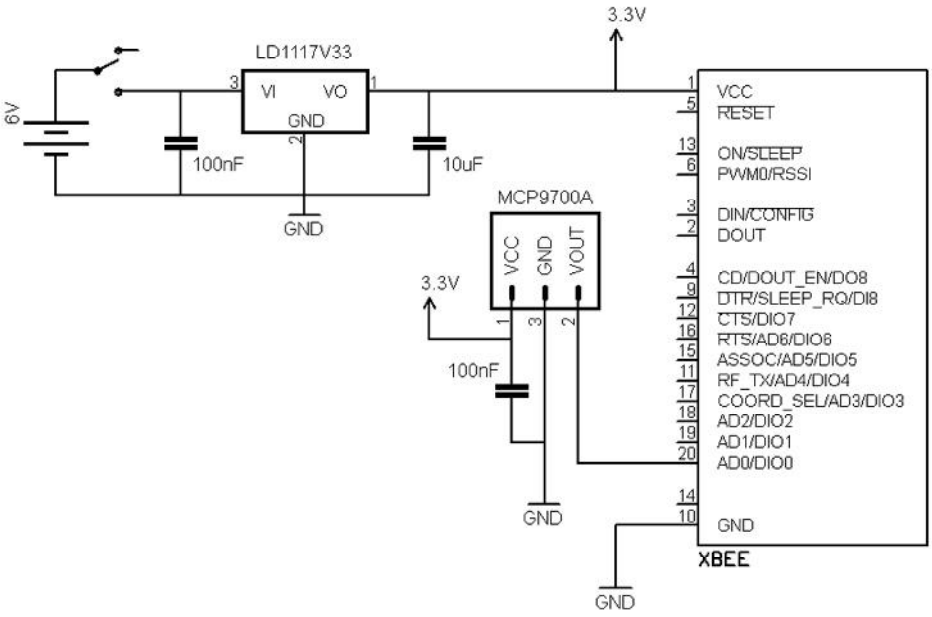


Figure 3.6.1.1. Schematic of the XBEE Transmitter

The remote XBEE module is loaded with the router firmware. It samples the ADC pin and transmits the sample to the receiving module. The sensor gives a linear voltage output of 10mV/°C, meaning the temperature can be calculated using a simple equation.

The settings on the XBEE modules were entered using the X-CTU software. The table below shows the relevant settings of the transmitter remote XBEE module. The parameters in bold are unique to the module and cannot be edited.

| Router | Description |
|---|---|
| ATID 3456 | PAN ID address |
| **ATSH 13A200** | Serial number (high) |
| **ATSL 40624DD8** | Serial number (low) |
| ATDH 13A200 | Destination address (high) |
| ATDL 403D8E44 | Destination address (low) |
| ATD0 2 | AD0 analogue input |
| ATIR FAO | Sample rate 4000ms |

ATID is the PAN ID address which is, essentially, the address of the network. The router and coordinator XBEE modules share this address to allow for remote communications. ATDH and ATDL contain the corresponding serial number (ATSH and ATSL) of the XBEE module which the information is to be sent to. ATD0 is used to change the setting of the AD0/DIO0 pin. Changing this value to '2' informs the XBEE that the pin is to be used as an analogue input. ATIR is used to set the sample rate of the input in milliseconds. 0xFAO is equivalent to 4000ms so the router XBEE samples the analogue input of the AD0 pin every four seconds and transmits it to the device sharing the same PAN ID and with the serial number '13A200403D8E44' [35].

Figure 3.6.1.2 shows the serial out (DOUT) pin of the receiving XBEE module connected to pin 6 of the Propeller. This Module is loaded with the coordinator firmware which means that it controls the network.

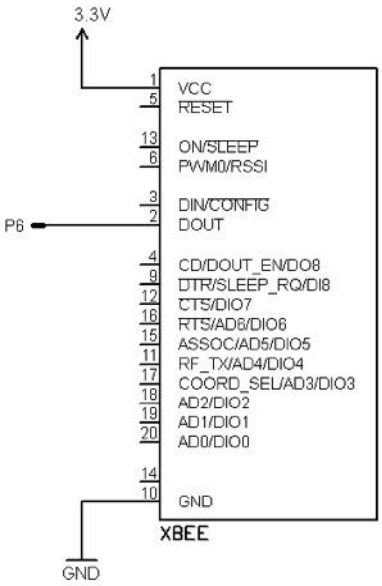


Figure 3.6.1.2. Schematic of the XBEE Receiver

The table below shows the relevant settings of the coordinator XBEE module's parameters.

| Coordinator | Description |
|---|---|
| ATID 3456 | PAN ID address |
| **ATSH 13A200** | Serial number (high) |
| **ATSL 403D8E44** | Serial number (low) |
| ATDH 13A200 | Destination address (high) |
| ATDL 40624DD8 | Destination address (low) |

API (Application Programming Interface) operations are used for the communications, meaning that packets of information are sent which follow a specific structure [35].

Figure 3.6.1.3 shows the API structure for a packet containing samples of data [35]. This is the information which the router XBEE transmits to the coordinator XBEE. It is also the information which the coordinator transmits to the Propeller.

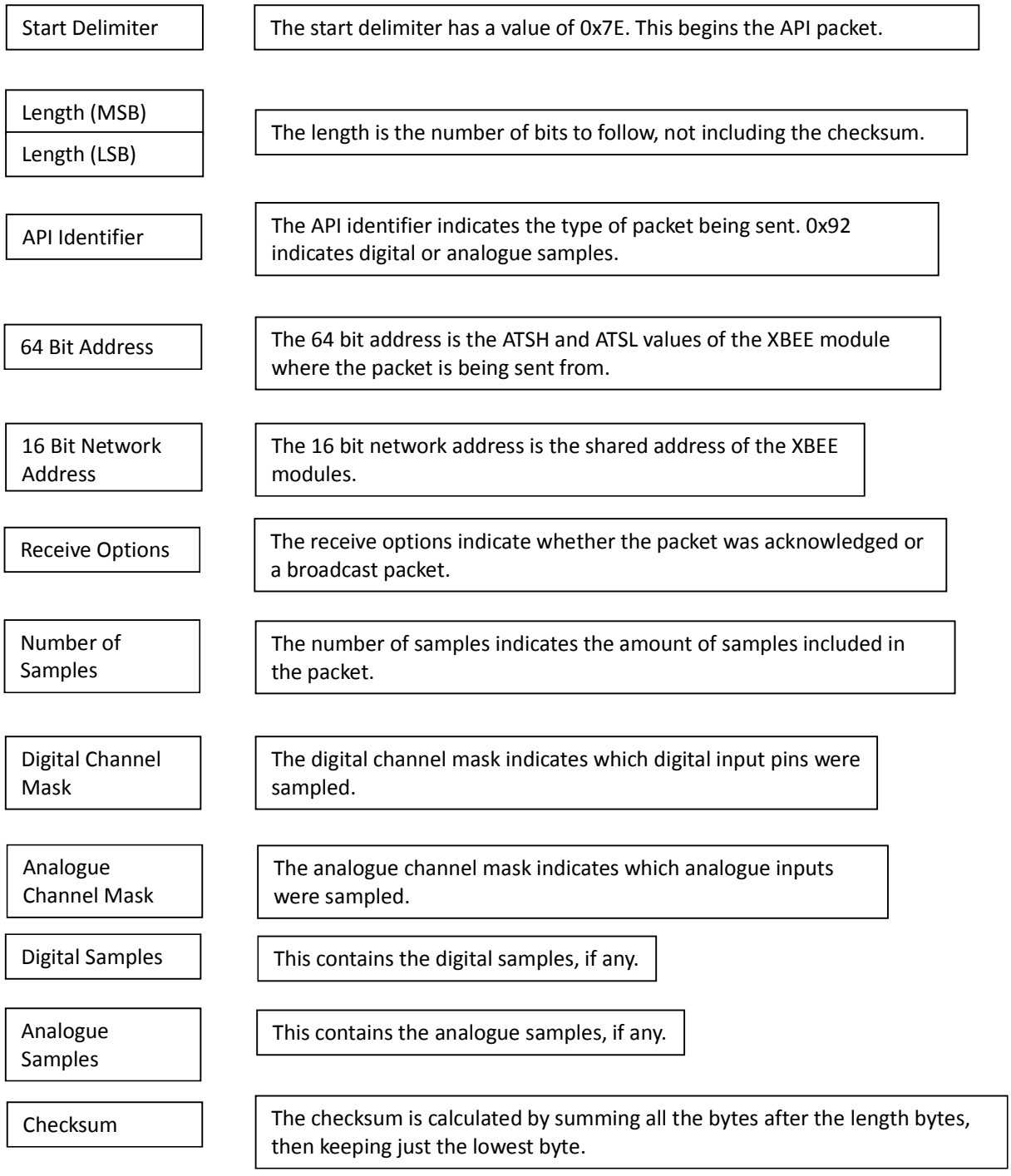


Figure 3.6.1.3. A Diagram Showing a Typical API Packet

The received sample is extracted from the API packet and then converted into a voltage using the following formula [35].

$$AD_{(mV)} = \frac{ADIO}{0x3FF} \cdot 1200mV \qquad (3.6.1 - 1)$$

The voltage is then used to calculate the temperature using another formula [36].

$$T_a = \frac{(V_{out} - V_{0°C})}{T_C} \qquad (3.6.1 - 2)$$

Combining these equations, and rearranging them to avoid making the Propeller chip perform floating point math, gives the following equation.

$$T_a = \frac{\frac{ADIO \times 1200}{0x3FF} - V_{0°C}}{T_C} \qquad (3.6.1 - 3)$$

### 3.7 X10 Settings

Adding and removing X10 devices from the information system can be done in "X10 Settings". When adding a device, the user is first prompted to enter the name of the device via the remote control or a keyboard. The device must then be assigned to one of the four rooms; "Living Room", "Dining Room", "Kitchen" or "Bedroom". The user must then specify whether the device is an appliance or lamp and the device address is then entered.

The code below shows the variables used for storing devices.

```
byte name[160]      'device name array
byte dvc[16]        'device number array
byte addr[16]       'device address array
byte room[16]       'device room array
byte type[16]       'device type array
byte dvcnum         'device number
```

"name" is an array, used to store the name of the device. It is an array of 160 bytes which means that each device can have a name of up to 10 characters, taking into account that there is a maximum number of 16 modules for each housecode.

"dvc" is a 16 byte array which stores the hexadecimal address of the X10 module. A look up table is then used to give the correct X10 device code to the "addr".

The "addr" array stores the X10 device codes of the modules.

"room" corresponds to the room of the device, where a value of "0" is the "bedroom", "1" is the "living room", "2" is the "dining room" and "3" is the kitchen".

The device type is stored in "type". A value of "0" means it is a device module and a value of "1" means it is a lamp module.

"dvcnum" holds the number of devices. This can vary between 0 and 15 as the maximum number of devices is 16.

The code below shows how the name of a device is input with the remote control.

```
PRI rcname_in(i) | j, t, x

   code := 0                                        'flush code
   t := i                                           'backup device number
   repeat j from i to i+9                           'clear bytes in name
      name[j]~
   i := t                                           'i = device number
   text.out($00)                                    'clear screen
   showdate(0, 0)                                       'show date and time
   showtime(35, 0)
   text.str(string(13,"      Add Device (Remote)",13,13,"      Name: "))          'prompt
user
   x := 12                                                  'x = position on screen
   repeat until ((code == enter) OR (i == t+9))             'repeat until enter is pressed or
all bytes input
      code := wait                                          'wait for button press
      if (code =< 9)                                    'if button pressed is a number
         text.xy(x, 4)                              'move text cursor to x, y position
         text.dec(code)                                 'display code
         name[i] := code + 48                           'save code as ASCII number
         i++                                            'move pointer to next byte
         x++                                            'increment cursor position
```

When entering a new name for a device, the 10 bytes which will be the name are cleared as a precaution. This prevents errors when displaying the name which would occur if there was any data already in these bytes. The screen is cleared and the date and time are shown. The user is then prompted to enter the name of the device. A loop is then entered which displays the key that the user inputs and moves the text cursor to the next tile and increments the data pointer to write the next byte of the name.

This method is similar for entering, editing and deleting devices. When a device has been added, the "dvcnum" variable is incremented so that when a new device is added, the program knows where to write the data in the arrays.

When a device is deleted, the data in the arrays are shifted down so that there is no empty space in the arrays. The "dvcnum" array is also decremented.

A variable can be assigned to point to a byte from the array to read information for a specific device. For example, to display the value of device number 2, "dp"(device pointer) can be set to "2". To show the name, a loop is used to display the byte at name[dp*10], then name[(dp*10)+1] and so on, until the name ends or the $10^{th}$ byte is displayed.

### 3.7.1 Memory backup

I2C is also used by the Propeller to write to the memory in EEPROM. This can be used to backup contents of the Propeller's RAM, meaning that when the system is reset, the backed up content is automatically restored.

This is used in the Information System when the user adds an X10 module to the system, the Propeller writes all the associated information to the EEPROM. The same process is undergone if the user deletes or edits an X10 module.

The code below shows how the backup method backs up the data.

```
VAR

    byte sel
    byte code
    byte name[160]
    byte dvc[16]
    byte addr[16]
    byte room[16]
    byte type[16]
    byte dvcnum
    byte rcflag
    byte snum[8]


PRI backup
{{backs up user entered x10 devices}}
    eeprom.VarBackup(@name, @dvcnum)
```

The main program calls the backup method and passes the first name of the variable to be backed up and the last name. All the variables between, as listed in the VAR block, are also backed up.

The code below shows the restore method. This is called when the Information System is initialising after boot up. It has the same names of variables passed into it.

```
PRI restore
{{restores backed-up user entries}}
    eeprom.VarRestore(@name, @dvcnum)
```

### 3.8 Settings

The fourth item allows the user to set the time and date using the remote control.

#### 3.8.1 Real Time Clock

Keeping the date and time on the information system required a Real-Time Clock (RTC). A PCF8563 is used for this purpose. It provides the month, date, day, hour, minutes and seconds, based on a 32.768kHz crystal. The circuit diagram for the RTC is shown below in figure 3.8.1. It uses a CR2032, 3V, battery to allow it to keep the correct time and date while the rest of the circuit is powered down. The clock (SCL) of the I2C bus is connected to pin 5 of the Propeller chip and the date (SDA) of the I2C bus is connected to pin 4 of the Propeller chip.

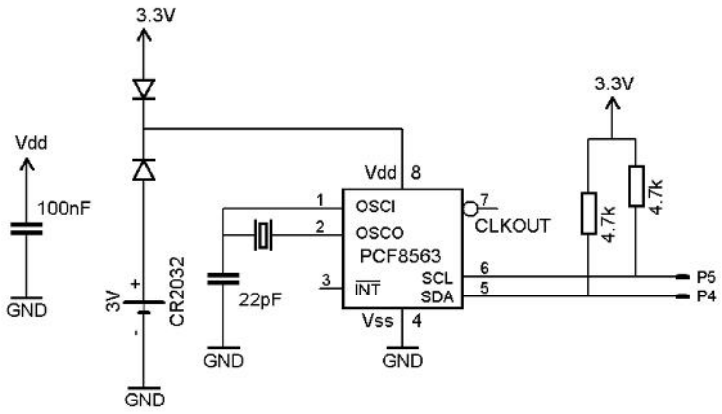


Figure 3.8.1. Schematic of the RTC Circuit

### 3.8.2 I2C

The I2C bus allows for bi-directional communications using a Serial Data (SDA) line and a Serial Clock (SCL) line. Both of these lines are connected to the positive line via a 4.7kΩ resistor, as shown in figure 3.8.1. One bit of data is sent on every clock pulse and the SDA line must be stable during the high state of the SCL line.

In the Information System, the Propeller chip acts as the master as it controls the messages on the I2C bus.

To start communicating on the bus, a Start condition must be performed. This involves the Propeller chip pulling the SDA line down while the SCL line remains high.

The first byte of a message is the addressing byte which is used to determine which device is to be communicated with. The PCF8563 has two addresses, 0xA3 is used to read a register and 0xA2 is used to write a value to a register. After the PCF8563 has been addressed, it transmits an acknowledge bit to the Propeller chip by pulling the SDA line low during the corresponding SCL high pulse.

A register address is sent from the Propeller on the next byte, which is the address of the register which is to be read or written to. Another acknowledge bit is then sent from the PCF8563.

If the read address was transmitted first, the PCF8563 transmits the value of the desired register and receives and leaves the SDA line high to signal the end of the data transfer without an acknowledge bit. If the write address was transmitted first, the Propeller transmits the value to be stored in the desired register in the PCF8563 and the RTC transmits an acknowledge bit.

The message is ended with a Stop condition, where the SDA line goes high while the SCL line is high.

The address and contents of the registers are shown below in figure 3.8.2.1 [37].

| Address | Register name | Bit | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Control and status registers | | | | | | | | | |
| 00h | Control_status_1 | TEST1 | N | STOP | N | TESTC | N | N | N |
| 01h | Control_status_2 | N | N | N | TI_TP | AF | TF | AIE | TIE |
| Time and date registers | | | | | | | | | |
| 02h | VL_seconds | VL | SECONDS (0 to 59) | | | | | | |
| 03h | Minutes | x | MINUTES (0 to 59) | | | | | | |
| 04h | Hours | x | x | HOURS (0 to 23) | | | | | |
| 05h | Days | x | x | DAYS (1 to 31) | | | | | |
| 06h | Weekdays | x | x | x | x | x | WEEKDAYS (0 to 6) | | |
| 07h | Century_months | C | x | x | MONTHS (1 to 12) | | | | |
| 08h | Years | YEARS (0 to 99) | | | | | | | |
| Alarm registers | | | | | | | | | |
| 09h | Minute_alarm | AE_M | MINUTE_ALARM (0 to 59) | | | | | | |
| 0Ah | Hour_alarm | AE_H | x | HOUR_ALARM (0 to 23) | | | | | |
| 0Bh | Day_alarm | AE_D | x | DAY_ALARM (1 to 31) | | | | | |
| 0Ch | Weekday_alarm | AE_W | x | x | x | x | WEEKDAY_ALARM (0 to 6) | | |
| CLKOUT control register | | | | | | | | | |
| 0Dh | CLKOUT_control | FE | x | x | x | x | x | FD[1:0] | |
| Timer registers | | | | | | | | | |
| 0Eh | Timer_control | TE | x | x | x | x | x | TD[1:0] | |
| 0Fh | Timer | TIMER[7:0] | | | | | | | |

Figure 3.8.2.1. The addresses and Contents of the PCF8563's Registers

The values in the registers are in Binary Coded Decimal (BCD), meaning that each digit, from 0 to 9, is represented by 4 bits. The lower 4 bits in the registers represent the units digit and the upper bits represent the tens digit. The Table below shows a decimal value of 34 in BCD. The 4 MSBs (the tens) have a value of 3 and the LSBs (the units) have the value of 4.

| | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|
| BCD | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| Decimal | 3 | | | | 4 | | | |

Figure 3.8.2.2 shows an example of the Propeller, reading the minute register of the PCF8563.

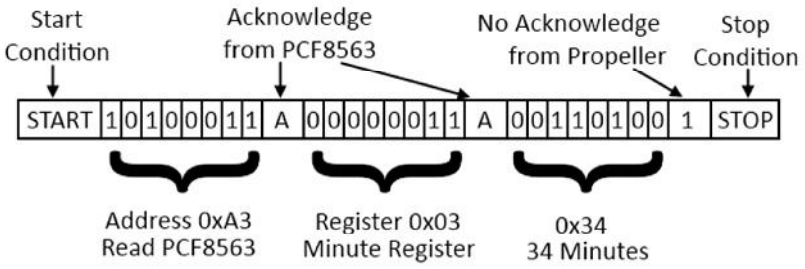


Figure 3.8.2.2. An Example of an I2C Message

In this example, the Propeller transmits the Start Condition, followed by the read address for the PCF8563. An acknowledge bit is received and the Propeller sends the address of the minute register. The PCF8563 acknowledges the request and then sends the value of the register. The PCF8563 then leaves the SDA line high to inform the Propeller that no more data will be transmitted and the Propeller transmits a Stop Condition.

### 3.8.3 Setting the Time and Date

The SPIN code below shows how the hours are set when setting the time in the program.

```
PRI sett | h, m
{{shows the set time menu}}

  text.out($00)                 'clear display
  showtime(3, 3)                'show current time at X, Y

  h := gethour                  'get hour and minute
  m := getminute

  text.colour(2)                'change text colour to yellow
  repeat until code == enter            'repeat until enter is pressed
    text.xy(3, 3)
    text.dec(h>>4 & $03)                'show hour (BCD)
    text.dec(h & $0F)
    repeat until rc.getcommand          'wait until a button is pressed
    code := rc.getcommand
    rc.flush
    if code == down             'increment hour
      ++h
      if (h & $0F) => 10
        h &= $30
        h += 16
      if (h & $3F) => $24               'loop hour
        h := 0
    if code == up               'decrement hour
      if (h & $0F) > 1
        --h
      elseif (h & $30) > 0
        h -= 16
        h |= $09
      else                              'loop hour
        h := $23

  text.xy(3, 3)                 'show set hour in white
  text.colour(0)
  text.dec(h>>4 & $03)
  text.dec(h & $0F)

  text.str(string(":"))
```

The code first displays the time in white text and then the hours value changes to yellow, indicating to the user that this is the value currently being modified.

The value of the hours is then modified by the user using the 'up' and 'down' buttons on the remote control until 'enter' is pressed. The hours then turn white again and the code repeats for the minutes.

### 3.8.4 Displaying date and Time

The SPIN code below shows a method which shows the time on the screen. Two variables are passed into the method which set the X, Y position where the time is to be shown. The Minute and Hour values are obtained using the "Getminute" and "Gethour" methods and displayed using the "dec()" method in the "Tv_text" object.

```
PRI showtime(x, y) | m, h
{method shows the time at X, Y position on the screen}

  m := getminute              'get the minute value
  h := gethour                'get the hour value
  text.xy(x, y)               'move text cursor to X, Y
  text.dec(h>>4 & $03)        'display the 10s h
  text.dec(h & $0F)           'display the units of h
  text.str(string(":"))
  text.dec(m>>4 & $07)        'display the 10s of m
  text.dec(m & $0F)           'display the units of m
```

The date and time are displayed in the same place on television on each menu with the exception of the set time and date screens. As the menus are often waiting for a button on the remote control to be pressed, it was practical to write a method to check the time and also the status of the remote control at the same time.
The method below returns the RC5 command when a button is pressed on the remote control. While there is nothing being pressed, the program waits for 4000000 clock cycles (50mS) and checks the RC5 status. This repeats 20 times, taking one second and then checking the minute value to see if there has been a change. If so, the date and time update and the program loops back to the previous loop. The purpose of the delay is so the minute value isn't being read constantly as this momentarily halts the PCF8563, making it run slower and becoming less accurate. The reason for the 50ms delay is in a loop rather than a single one second delay is so that if a button is pressed, the loop ends. This makes the system seem more responsive to the user.

```
PRI wait : c | m, i
{{waits for key press while updating time}}

  rc.flush                        'flush RC5
  m := getminute                  'get minute value

  repeat until rc.getcommand      'repeat until button is pressed
    repeat i from 0 to 19         'repeat 20 times
      waitcnt(cnt + 4_000_000)    'wait 50ms
      if rc.getcommand            'if button is pressed
        i := 19                   'exit repeat

    if m <> getminute             'if minute value has changed
      showtime(35, 0)             'show time
      showdate(0, 0)              'show date
      m := getminute              'update minute
  c := rc.getcommand                      'return RC5 code
  rc.flush                        'flush RC5 code
```

### 3.9 Infra-red Remote Controller

Infra-red (IR) remote controls are commonly used to control home entertainment systems as they are simple to implement, very cheap to manufacture and make it easier for the consumer to use devices as they can be operated from a distance. The main disadvantage of infra-red is that the signal cannot penetrate opaque objects.

Remote controls use an IR LED to emit pulses of IR light. This is then received by an IR receiver which decodes the signal and executes the required function. To prevent noise from interfering with the IR receiver from the sun or other sources of IR noise, the signal is modulated onto a carrier signal, typically with a frequency of 36kHz. This means that a burst of IR signals at 36kHz represents a logic '1' and no signal represents a logic 0.

The protocol used in this project is the RC5 protocol. An RC5 code consists of 14 bits, including 2 start bits which are always logic '1', a toggle bit which toggles every time the same button is pressed, 5 address bits and 6 command bits [38].

An RC5 transmission encodes the data in Manchester code. This method of encoding ensures that there is a clock transition in the middle of each bit period; A low to high transition is a logic 1 and a high to low transition is a logic 0 [39]. The bit period is 1.778ms long with a transition in the middle (after 889µs). The transition on every clock pulse ensures that the receiver can synchronize with the transmitter more easily [40]. A standard RC5 code is shown in Figure 3.9. It is transmitting the command value of 17. This is the 'decrease volume' button.

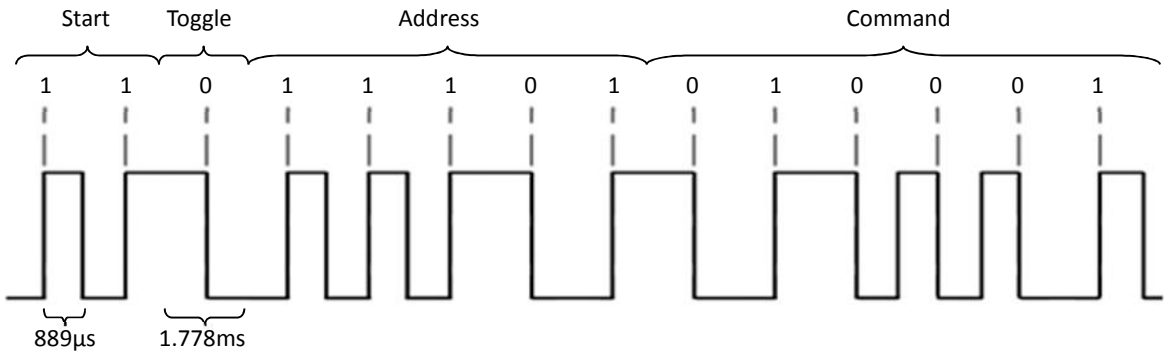


Figure 3.9. A Typical RC5 Transmission

### 3.9.1 IR receiver

The Infra-Red receiver circuit is based on a Sharp GP1UX301QS IR receiver. The device is tuned to detect bursts of IR signals at 40KHz which is capable of detecting 36KHz signals.

Figure 3.9.1 shows the circuit diagram for the IR receiver. The output of the receiver is the demodulated IR signal and is connected to pin 2 of the Propeller chip where it is decoded by the program.

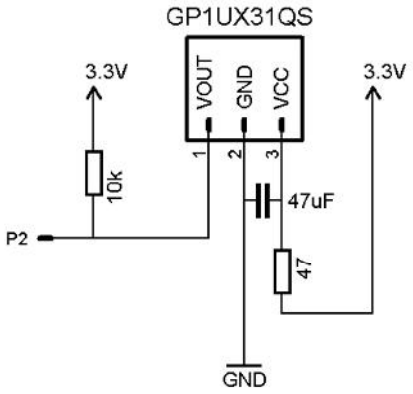


Figure 3.9.1. Circuit Diagram showing the Infra-Red Receiver

The signal from the IR receiver is inverted from the theoretical waveform so a high to low transition is a logic '1' and a low to high transition is a logic '0'.

The code below shows the initialization of the RC5 object. It is initialized by calculating the number of clock cycles in a full bit period, three quarters of a bit period, half of a bit period and a quarter of a bit period, based on the clock frequency. The frequency of the RC5 codes is 1/1.778ms which is approximately 562. Multiplying this by the frequency of the Propeller's clock gives the number of clock pulses between each bit period.

The code then begins the RC5 cog.

```
PUB init (pin)
{{Launch cog to detect RC5 codes}}
  rxmask := |< pin            'mask input pin
  full := (clkfreq/562)       'calculate bit times
  thalf := full >> 1
  quart := thalf >> 1
  tquart := (thalf + quart)
  cognew(@go, @rc5)           'begin RC5 cog
```

The code below is written in PASM and decodes RC5 codes received at the Propellers input pin.

```
DAT

        org 0
go      mov addr, PAR
        mov outmask, dira               'backup dira in outmask
        xor outmask, rxmask             'xor rx pin with outmask
        and dira, outmask               'and' 'xor'ed outmask with dira
detect          mov rxcode, #0
        mov count, cnt
        add count, delay
        waitcnt count, 0
        waitpne rxmask, rxmask          'Wait for start bit
        mov start, cnt                  'time start bit pulse
        waitpeq rxmask, rxmask
        mov half, cnt
        sub half, start                  'half - start = start pulse length, should be about
71120 (80MHz * 889us)
        cmp lh, half                     wz, wc         'check that the half value is between
the upper and lower limits
if_nc    jmp #detect
        cmp uh, half                    wz, wc
if_z_or_c jmp #detect

Sb2             mov count, cnt                  'wait 3T/4 to detect second start bit
        add count, tquart
        waitcnt count, 0
        test rxmask, ina                wz      'check second start bit
if_nz   jmp #detect                             'jump to detect if zero not detected

        mov _l, #12                             'set up loop iteration
        mov count, cnt                          'begin receiving rc5 code
        add count, full                         'wait full time period

loop    waitcnt count, full                     'wait full time period
        test rxmask, ina                wc      'test rx
if_nc   add rxcode, #1                          'add 1 if rx is low
        shl rxcode, #1                          'shift rxcode left
        djnz _l, #loop                          'loop until full code is received
        shr rxcode, #1
        wrlong rxcode, addr                     'write rxcode to hub

        mov time, cnt
        add time, delay
        waitcnt time, 0                         'short delay
        jmp #detect                             'go back to detect next code


rxmask  long 0                   'IR input pin
lh      long 70000               'lower threshold of half bit time
uh      long 75000               'upper threshold of half bit time
thalf   long 0                  ‘calculated half bit time
```

```
quart   long 0                  'calculated 1/4 bit time
tquart  long 0                  'calculated 3/4 bit time
full    long 0          'calculated full bit time
_l      long 0
delay   long 5_000_000
half    long 0                  'timed half bit time
outmask res 1
time    res 1
start   res 1
addr    res 1                   'PAR address
rxcode  res 1
count   res 1
frame   res 1
```

This code is split into four parts, 'go', 'detect', 'sb2', and 'loop'.

The 'go' section initializes the input pin and skips to the 'detect' section. This waits for the line to be pulled low and the Propeller's 'count' value is temporarily backed up. The code waits for the line to be pulled high again and the difference in the 'count' value is compared with the upper and lower threshold for the bit period. If the 'count' value is between 70000 and 75000, the code continues to the 'sb2' section.

The 'sb2' section of the code waits for three quarters of the bit period and samples the second start bit. If the second start bit is not detected, the code jumps back to the detect section.

If the second start bit is present, a loop is entered where the code waits for one bit period and samples the line. This repeats 12 times until all the bits have been received. The code is then written to the hub memory so the rest of the Propeller program can manipulate the code.

## 4 Evaluation of the Information System

To assess the functionality of the system, a series of tests were carried out. The tests included analysing the video signal, the user interface, the remote control input, the internal and external temperature measurements, the communication with the X10 controller, the real time clock and the program's memory usage.

### 4.1 TV Circuit

Figure 4.1 shows a scan line of the PAL TV signal generated by the propeller, including a closer look at the colour burst at the beginning of the scan line.

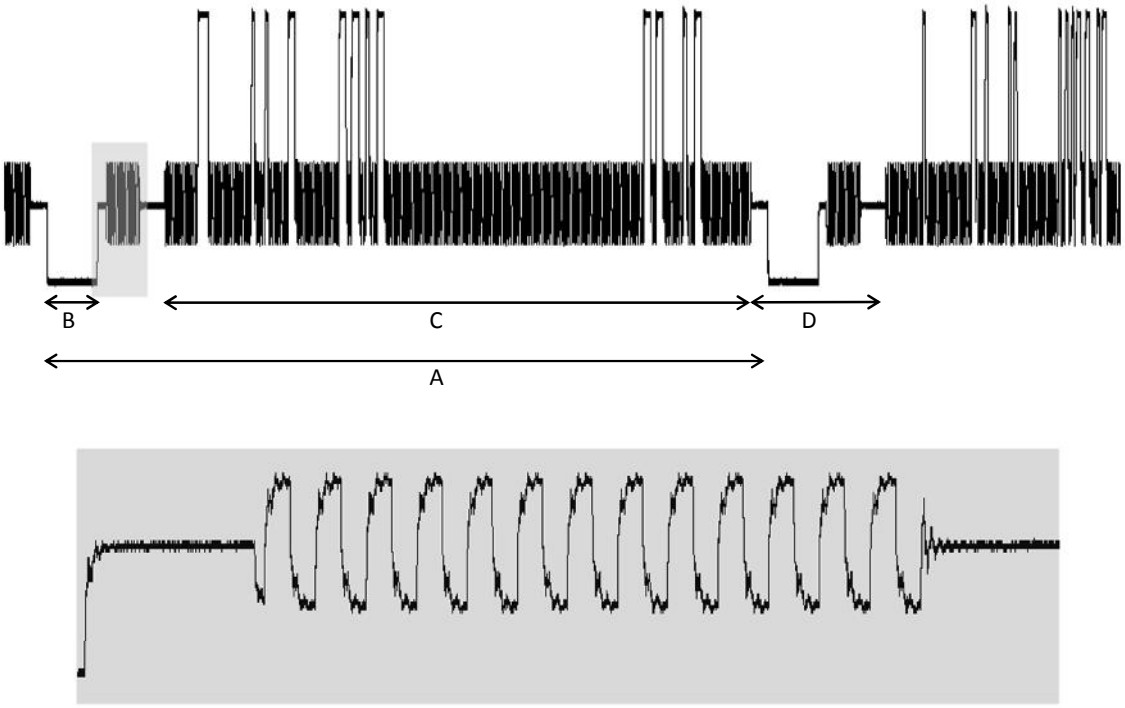


Figure 4.1. Waveform of the PAL TV Signal

The table below shows the corresponding timing measurements for the waveform in figure 4.1. The third column shows the difference from the expected times to the measured times. All the measurements are reasonably low and within the acceptable limits.

| | **Theoretical [41]** | **Practical** | **Error %** |
|---|---|---|---|
| **Scan line (A)** | 64µs | 64µs | 0% |
| **Horizontal Sync (B)** | 4.7µs | 4.52µs | 3.83% |
| **Active Video (C)** | 52µs | 52.16µs | 0.3% |
| **Horizontal Blanking (D)** | 12µs | 11.88µs | 1% |
| **Subcarrier Frequency** | 4.43MHz | 4.42MHz | 0.2% |

### 4.2 The User Interface

Figure 4.2.1 shows a photograph of the main menu screen with the "Settings" item selected. The date can be seen in the top left corner and the time can be seen in the top right corner.


Figure 4.2.1. A Picture of the Main Menu

Figure 4.2.2 shows a picture of the "X10 Setup" item, where the user is able to enter, or edit, the X10 modules in the Information System.

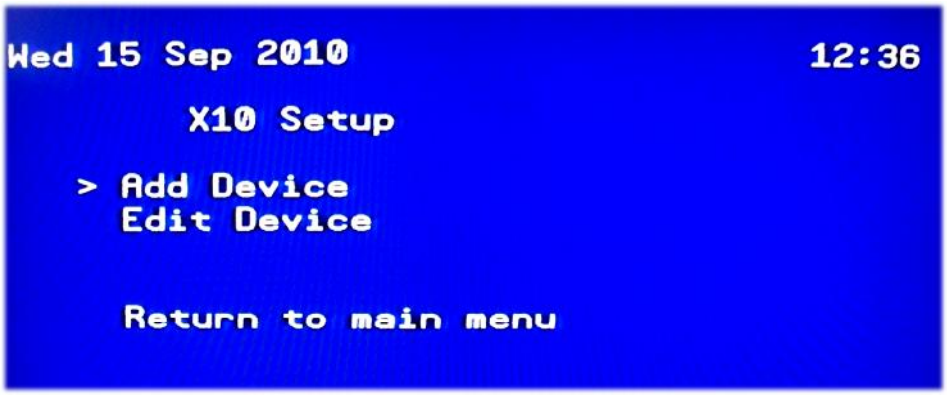


Figure 4.2.2. A Picture of the "X10 Setup" Item

Figure 4.2.3 shows the name of the X10 device being entered as "123". This is done via the remote control as there is no keyboard detected by the Information System.


Figure 4.2.3. A Picture of the Device Name, "123", being Entered

Figure 4.2.4 shows the room of the device being entered. In this example, Dining room is selected.


Figure 4.2.4. A Picture of the Device's Room being Entered

Figure 4.2.5 shows the device type for "123" being entered as "lamp".


Figure 4.2.5. A Picture of the Device Type being Inputted

Figure 4.2.6 shows the X10 address of the device being entered. The address entered was "02".


Figure 4.2.6. A Picture of the X10 Address being Inputted

The "Edit Device" item in the "X10 Setup" item allows the user to view the modules already in the Information System's memory. The user can also modify or delete the modules from this menu. Figure 4.2.7 shows the "Edit Device" menu with the device named "123" along with some previously entered devices.

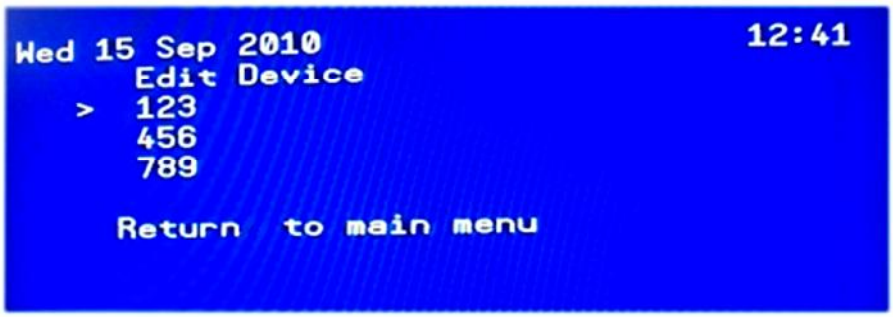


Figure 4.2.7. A Picture of the "Edit Device" Menu

Figure 4.2.8 displays the properties of the device, "123", from the "Edit Device" item. The room is "Dining Room", the type is "Lamp" and the address is "A2". This matches the user inputted data which was entered.


Figure 4.2.8. A Picture of the Properties of the Device, "123"

Figure 4.2.9 shows the "Home Automation" menu. The "All Devices" option displays all the X10 devices and the "List by room" option allows the devices to be filtered by room.


Figure 4.2.9. A Picture of the "Home Automation" Menu

Figure 4.2.10 shows the Information System listing all the devices from the memory.


Figure 4.2.10. A Picture of the Device Listing

Figure 4.2.11 shows the Information System listing the devices which are in the "Dining Room". These are "123" and "789", matching the information which was entered.


Figure 4.2.11. A Picture of the Devices in "Dining Room"

Figure 4.2.12 shows the menu which prompts the user to select the required function for the device, "123". The options are to turn it on, off or dim it.


Figure 4.2.12. A Picture of the Device's Function Menu

Figure 4.2.13 shows the "Settings" menu, giving the user the options to set the time and date.


Figure 4.2.13. A Picture of the "Settings" Item

Figure 4.2.14 shows the interface as the time is being set. In this picture, the hours are yellow, meaning that they are currently being modified.

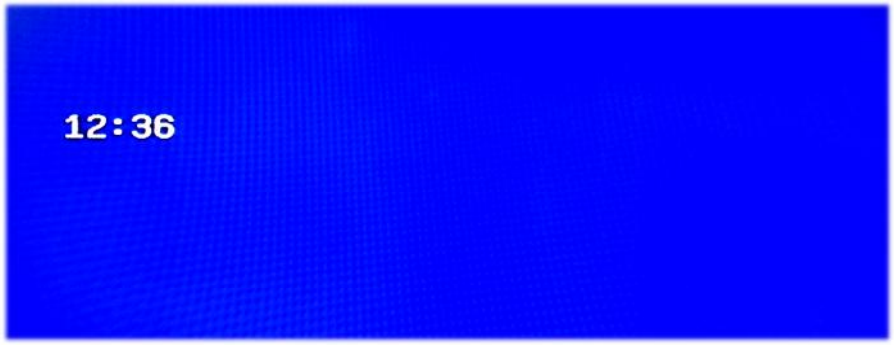


Figure 4.2.14. A Picture of the "Set Time" Item

Figure 4.2.15 shows the interface as the date is being set. In the picture, the day is highlighted in yellow, indicating that this is the part which is currently being edited.


Figure 4.2.15. A Picture of the “Set Date” Item

### 4.3 Infra-red Remote Controller

The waveform in figure 4.3 shows the input to the Propeller from the Infra-Red receiver when the '5' button is pressed on the remote control.

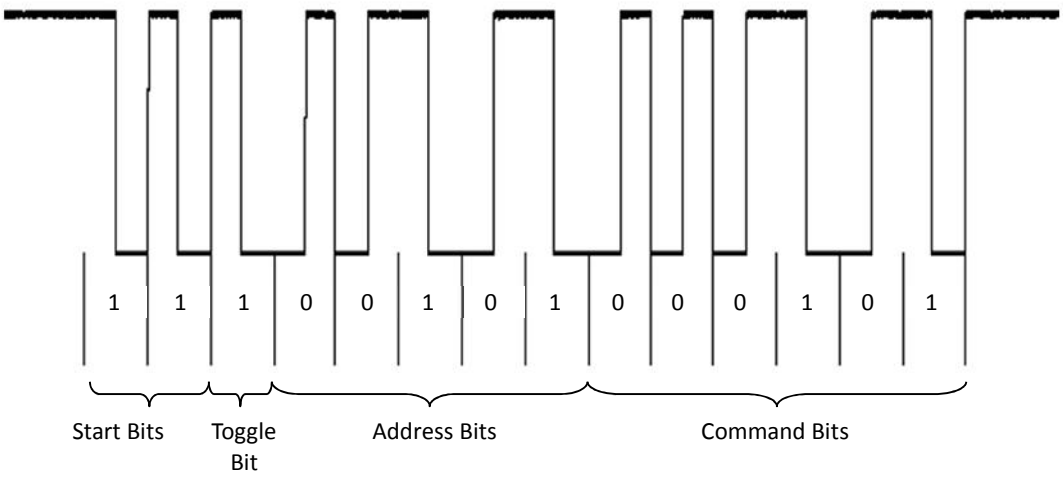


Figure 4.3. Waveform of Input from Infra-Red Receiver

The two start bits are logic '1', as expected and the toggle bit is also logic '1'. This toggles with every new key press. The address bits are equal to '5'. The Information System does not use this information to distinguish between remote controls. The command bits are equal to '5', which is the corresponding value for the '5' button on the remote control.

Each bit period was measured at 1.78ms, just 2µs over the theoretical value. The Propeller seems very responsive at dealing with the input from the remote control as it has a cog dedicated to just this task.

### 4.4 Temperature Measurement

The picture in figure 4.4 shows the temperature menu, displaying both inside and outside temperatures in degrees Celsius.

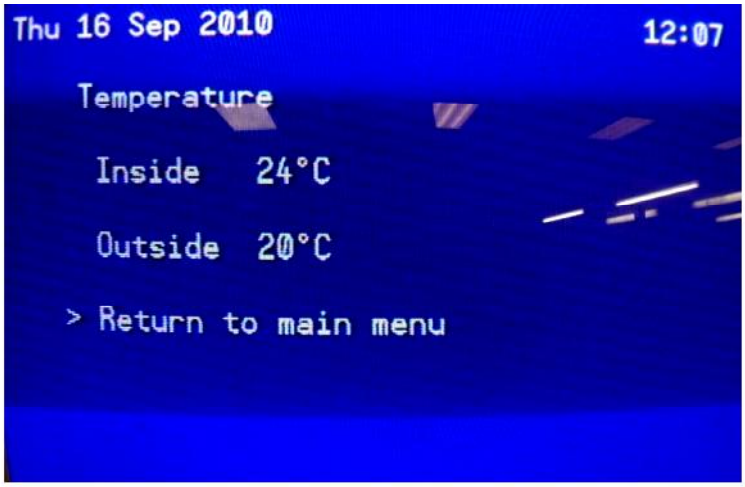


Figure 4.4. Picture of the Temperature Menu with the Internal and External Temperatures Displayed

#### 4.4.1 Internal Temperature

To start the conversion of the temperature on the DS18B20 IC, the Propeller transmits a reset pulse and awaits the response of the DS18B20's presence pulse. The Skip ROM command is transmitted (0xCC) followed by the Convert Temperature command (0x44). The Skip ROM command saves time by skipping the transmission of the 64 bit ROM code. The Convert Temperature command alerts the DS18B20 to begin the temperature conversion. Figure 4.4.1.1 shows the waveform of the transmission of these two bytes from the Propeller. Note that the least significant bit is transmitted first.

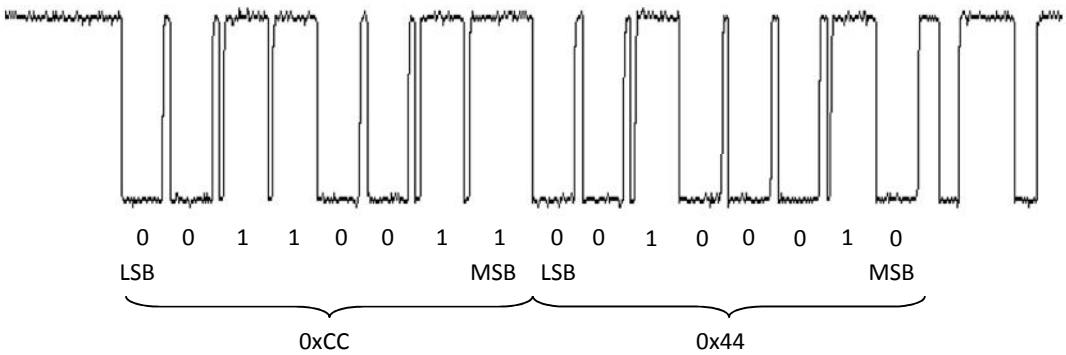


Figure 4.4.1.1. Waveform of the Skip ROM and Convert Temperature commands

The propeller waits until the temperature conversion is complete by issuing read time slots. The DS18B20 responds to this by transmitting '0' until the conversion is complete and it will transmit a '1'. The Propeller then transmits the Skip ROM command again, followed by the Read Scratchpad command. This transmission can be seen in figure 4.4.1.2.

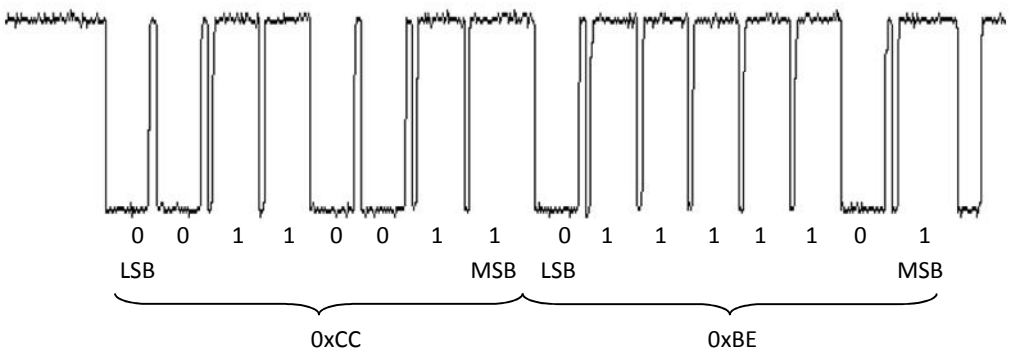


Figure 4.4.1.2. Waveform of the Skip ROM and Read Scratchpad commands

When the DS18B20 receives the Read Scratchpad command, it begins transmitting the contents of the Scratchpad until a reset pulse is transmitted by the Propeller. The temperature value is contained in the first two bytes of the Scratchpad. Figure 4.4.1.3 shows the transmission of the 8 least significant bits of the temperature.

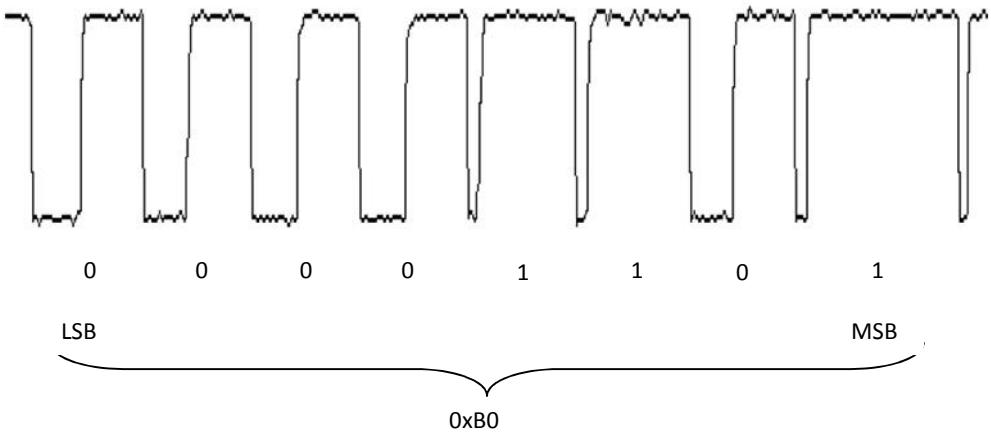


Figure 4.4.1.3. Waveform of the 8 least significant bits of the Scratchpad

Figure 4.4.1.4 shows the transmission of the 8 most significant bits of the temperature. A reset pulse is then sent from the Propeller to inform the DS18B20 that further information is not required.

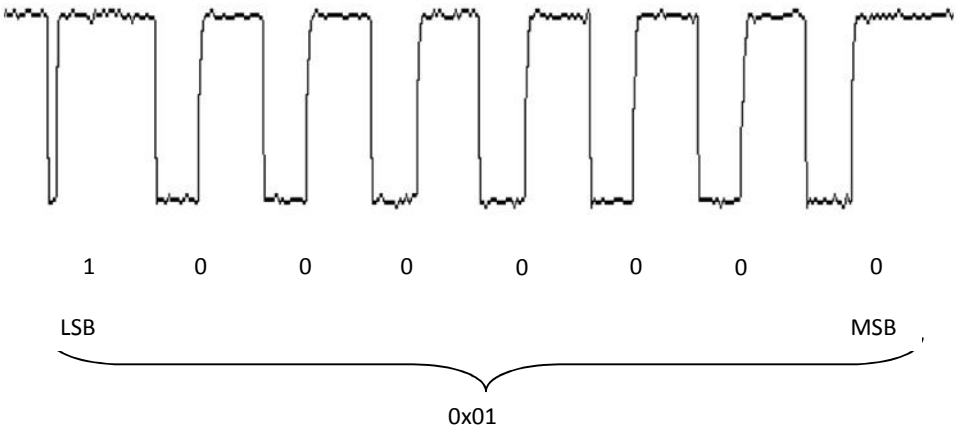


Figure 4.4.1.4. Waveform of the 8 least significant bits of the Scratchpad

The lowest four bits of the temperature can be ignored as they represent the fraction of centigrades which are not needed in the Information System. Putting the two bytes together gives a value of 0x1B, or 27°C.

### 4.4.2 External Temperature

To view an incoming API packet, the Parallax Serial Terminal was used. This allows the Propeller to communicate with the computer and the software shows the data on the screen. Figure 4.4.2 shows a screenshot of the software after the Propeller received an API packet from the XBEE.

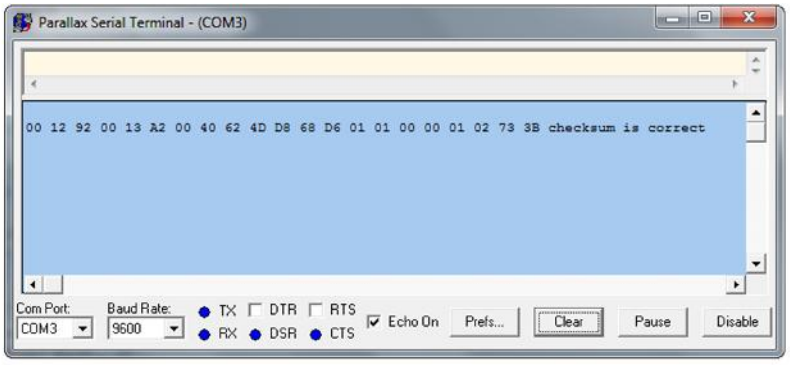


Figure 4.4.2. Screenshot of the Parallax Serial Terminal displaying an incoming API packet

In this example, the first two bytes are 0x00 and 0x12, meaning that there are 18 bytes of information to follow, excluding the checksum.

The 3rd byte is the API identifier. 0x92 notifies the receiver that sampled data is being transmitted.

The next 8 bytes contain the 64 bit address of the remote XBEE, 0x00, 0x13, 0xA2, 0x00, 0x40, 0x62, 0x4D and 0xD8.

Bytes 12 and 13 are the 16 bit network address. In this case they are 0x68 and 0xD6. Byte 14 indicates that the packet was acknowledged and byte 15 specifies how many samples are included in the packet. In this case, just one sample is transmitted.

Bytes 16 and 17 specify the digital channel mask which are both set to 0x00 as there are no digital samples.

The analogue channel mask is indicated by byte 18, which is set to 0x01. This means that the XBEE's input pin 1 is the analogue, sampled data.

Bytes 19 and 20 hold the sample data. In this case, the values are 0x02 and 0x73. The sampled voltage from the temperature sensor is then found using the following formula.

$$AD(mV) = \frac{0x273}{0x3FF} \times 1.2 = \frac{627}{1023} \times 1.2 = 0.7355V \qquad (4.4.2 - 1)$$

This is then converted to a temperature using the following formula

$$T_a = \frac{(0.7355 - 500 \times 10^{-3})}{10 \times 10^{-3}} = 23.55°C \qquad (4.4.2 - 2)$$

The equation used in the information system has been rearranged to avoid performing floating point math. This equation is shown below.

$$T_a = \frac{\frac{ADIO \times 1200}{1023} - 500}{10} = 23.55°C$$

The checksum, 0x3B, is calculated by summing all the bytes after the length bytes, then taking the lowest byte and subtracting it from 0xFF. To verify the checksum, the XBEE datasheet recommends adding all the bytes after the length bytes, including the checksum. If the checksum is correct, the lowest byte of this will equal 0xFF.

$$0X92 + 0x13 + 0xA2 + 0x40 + 0x62 + 0x4D + 0xD8 + 0xD6 \quad (4.4.2 - 3)$$
$$+ 0x68 + 0x01 + x01 + 0x01 + 0x02 + 0x73 + 0x3B = 0x4FF$$

Taking the lowest byte gives 0xFF, as expected, verifying that the information in the API packet is correct.

**4.5 RS232**

Figure 4.5 shows the waveform for an RS232 signal transmitting the value 0x63 at 4800bps. The top waveform shows the signal as it is output from the Propeller, into the MAX3232 IC and the bottom waveform shows the RS232 compatible signal as it outputs the MAX3232 IC.

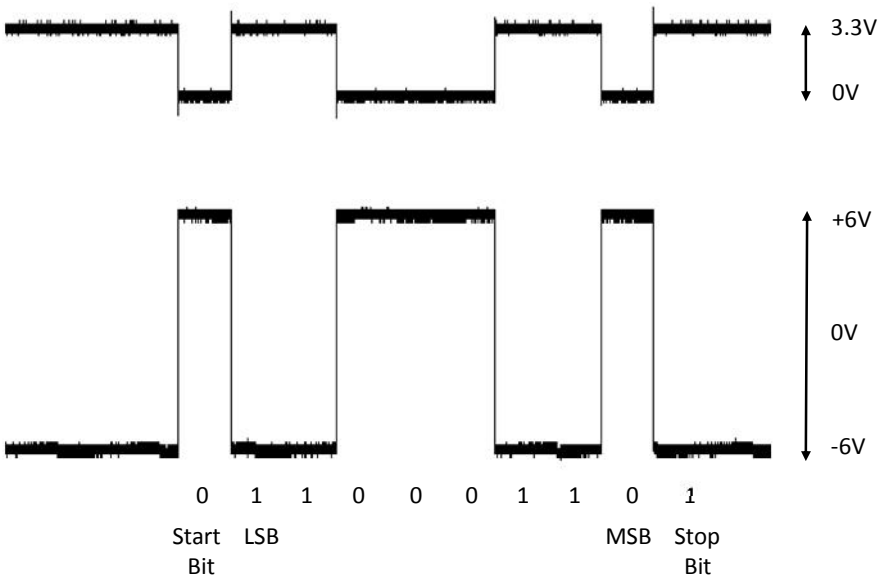


Figure 4.5. Waveform showing the RS232 Signal from the Propeller and MAX3232

To calculate the theoretical bit period, the following equation was used

$$Bit\,Period = \frac{1}{4800} = 208.33\mu s \quad (4.5 - 1)$$

The measured bit period was equal to 210µs, a 0.8% error.

### 4.6 X10

The testing of the X10 commands proved that they were reliable. The modules performed their required functions without a problem although they were quite slow to respond, taking approximately one second to perform the desired function due to the data transfer speed being limited to the 50Hz mains frequency.

### 4.7 I2C

The I2C example in this section show how the Propeller communicates with the PCF8563 IC to read the minute register.

Figure 4.7.1 shows a waveform of the Propeller sending the write address of the PCF8563. The top waveform (SDA) shows the serial data and the bottom waveform (SCK) shows the clock signal.

The waveform begins with the Propeller pulling the SDA line low while SCK remains high. This is the Start condition. The PCF8563 then reads the incoming byte and transmits an acknowledge bit. The Propeller then understands that the PCF8563 is present on the I2C line.

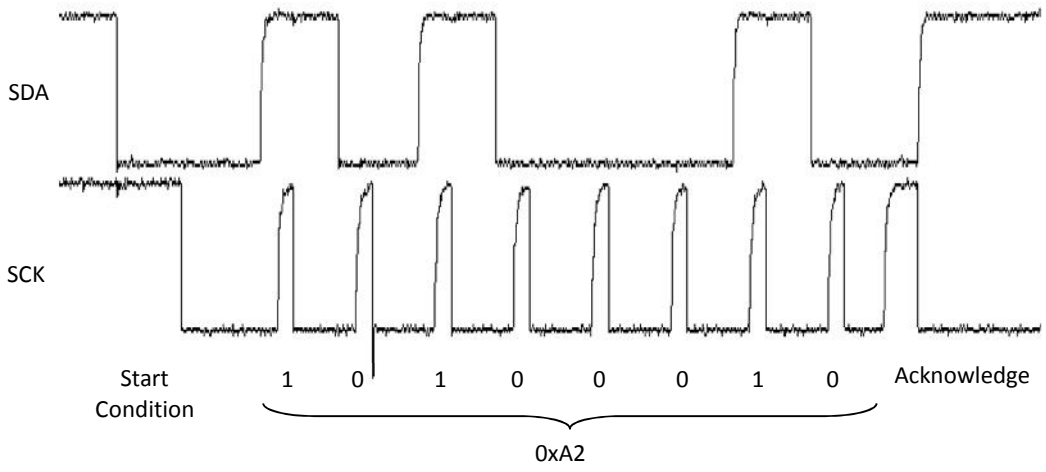


Figure 4.7.1. I2C Waveform Sending the Address 0xA2

Figure 4.7.2 shows the transmission of the register address of the minute register (0x03) being transmitted. This is the register of the value which is required.

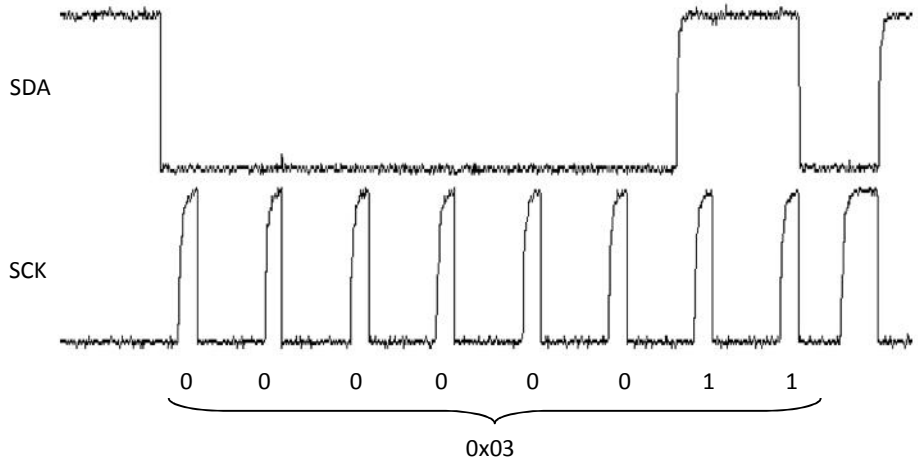


Figure 4.7.2. I2C Waveform Sending the Address 0x03

Figure 4.7.3 shows the transmission of the PCF8563's read address.

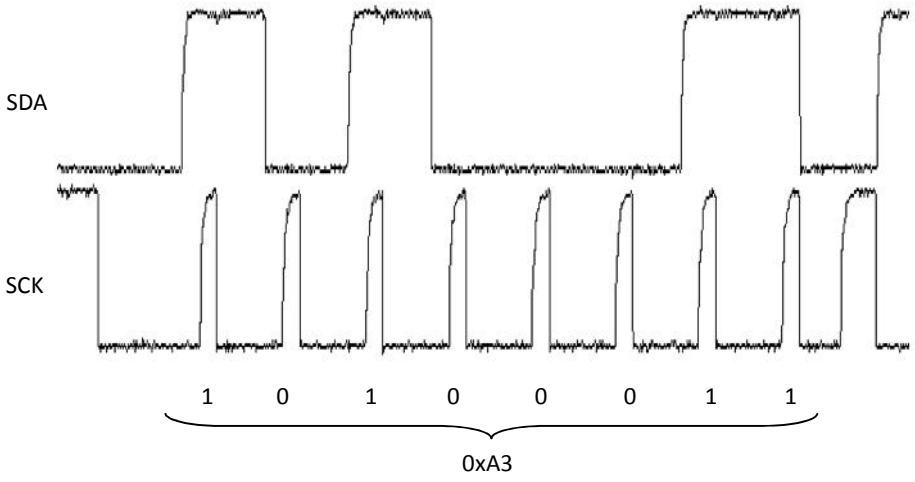


Figure 4.7.3. I2C Waveform Sending the Address 0xA3

Figure 4.7.4shows the transmission of the PCF8563's minute register to the Propeller's input pin. The value of the 7th bit is not used in this register. The value of the register is 0x52, which converted from binary coded decimal is 52 minutes.

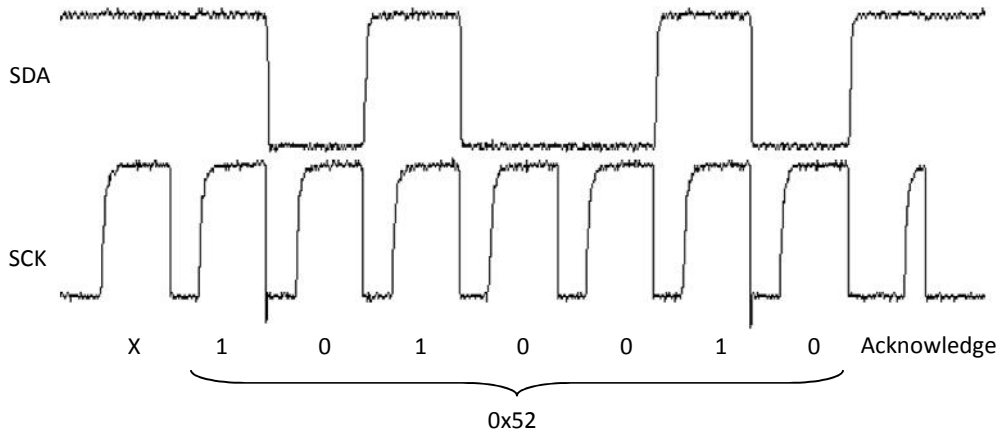


Figure 4.7.4. I2C Waveform Sending the byte 0x52

**4.8 Real Time Clock**

The testing of the clock showed that over long periods of time, the clock seemed to become slightly out of sync by running too slow. After one month from setting the time on the Information system, the time was out by approximately 4 minutes. This could be down to the crystal oscillator running at a slightly slower frequency than specified or it could be the chip itself. This is not thought to be too much of a problem as it is very easy to set the correct time again.

### 4.9 EEPROM Memory Usage

Figure 4.9 shows a screenshot of the program information which can be found by pressing F8 in the Propeller Tool. This data is loaded into the EEPROM and the Propeller loads it into the internal RAM as it boots up. The figure shows that the majority of the Propeller's memory is taken up by program memory but there is still over 35% of memory remaining. This leaves room for adding additional features to the Information System, in the future.

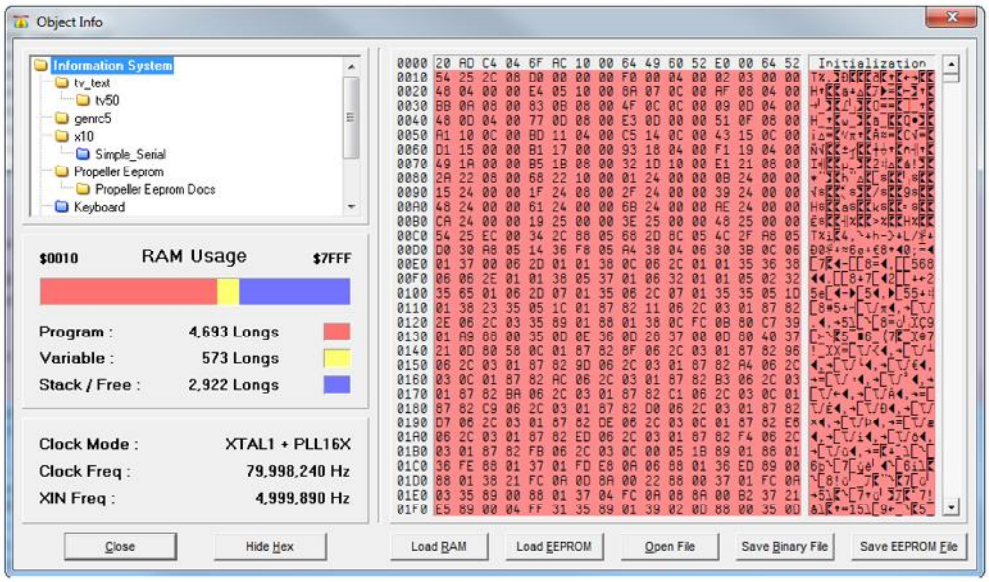


Figure 4.9. Screenshot of the Program Information

## 5 Conclusion

In this report, an Information System has been designed to provide home control, and a prototype has been developed and constructed. It was found that there is a need for this type of device among elderly and disabled people and that it is not an unrealistic situation.

The Information System worked as expected, allowing the user to control X10 modules by adding devices to the device's memory via the remote control or a keyboard. The devices can be sorted into rooms to ensure that the number of devices doesn't complicate the software or overwhelm the user.

### 5.1 Usability of the Information System

The menu was kept simple, with just 4 items it is always easy to return to the main menu. This keeps the software straight forward and prevents the user getting 'lost' in the menus.

The navigation of the menu is simple, needing the use of just three buttons; up, down, and enter. This meets the needs of elderly people who may not be comfortable at getting to grips with more complicated software. The software becomes a little more complicated when adding X10 devices, needing numbers to be pressed but this should be something which most users are familiar with, having used remote controls before.

The response from the remote control was impressive, taking into account that when a key was pressed, the cursor had to be redrawn or a completely different menu was to be shown. Any delay from the user pressing a button to the screen refreshing is not noticeable to the human eye.

The screen resolution of the Information System's interface is low, with a character grid of 40 columns by 13 rows. This makes the characters large and easy to read for visually impaired users. The white text on a dark blue background is also easy on the eye.

### 5.2 Installation/Implementation

To install the Information System into a home, it should be placed near a television in a similar way to a VCR or set-top box. The power is provided by an external, centre positive, 5V AC adapter. The television is connected via a phono lead.

When the System is switched on for the first time, the user will be required to set the time and date by navigating to the settings menu.
X10 modules can be added to the System by entering the desired name on a keyboard if one is present. Otherwise, the name must consist of numbers, entered by the remote control.

This installation requires very little effort and time, allowing for stress-free implementation.

### 5.3 Cost

The table below shows the projected cost of the components of the Information system at £110.87. This price includes all the specialised parts, including the XBEE modules, the AC adapter, the CM12 X10 controller and the relevant ICs. The price does not include any X10 modules which can be purchased for around £20 [42].

This cost seems reasonable as the closest realistic competitor, the "EasyTouch Panel10", has a retail price of £179.00 [43].

The two XBEE modules, at around £20 each, account for approximately 40% of the overall cost of the Information System. Finding another method of reading the temperature could greatly decrease the price of manufacturing the Information System.

Another method of decreasing the cost is by transmitting X10 codes directly through the mains from the Information System, rather than to the CM12. This is due to the cost of the CM12 X10 controller making up approximately 30% of the whole cost.

| Item | Price | Quantity | Net Cost | Source |
|---|---|---|---|---|
| Propeller IC | £5.90 | 1 | £5.90 | http://www.active-robots.com/products/parallax/propeller-chip-40-pin-dip.shtml |
| 24LC256 | £1.48 | 1 | £1.48 | http://uk.farnell.com/microchip/24lc256-e-sn/eeprom-256k-i2c-2-5v-soic8/dp/1579573 |
| AC Adapter | £6.26 | 1 | £6.26 | http://uk.farnell.com/stontronics/ad-05100rbs2-1/adaptor-5vdc-1a-uk/dp/1279510 |
| XBEE 10 pin headers | £0.30 | 4 | £1.20 | http://proto-pic.co.uk/products/2mm-10-Pin-Xbee-Socket.html |
| XBEE | £22.21 | 2 | £44.42 | http://uk.farnell.com/digi-international/xb24-buit-004/module-zigbee-xbee-znet-2-5-1mw/dp/1546392 |
| MCP9700A | £0.34 | 1 | £0.34 | http://uk.farnell.com/microchip/mcp9700a-e-to/thermistor-linear-10mv-c-to-92/dp/1332164 |
| DS18B20 | £3.27 | 1 | £3.27 | http://uk.farnell.com/maxim-integrated-products/ds18b20/programmable-digital-t-mom-18b20/dp/1187948 |
| GP1UX301QS | £0.93 | 1 | £0.93 | http://uk.farnell.com/sharp/gp1ux301qs/photodiode-ir-detector/dp/1243866 |
| MAX3232 | £3.24 | 1 | £3.24 | http://uk.farnell.com/maxim-integrated-products/max3232cpe/transceiver-3232-dip16/dp/9724494 |
| LD1117V33 | £0.78 | 2 | £1.56 | http://uk.farnell.com/stmicroelectronics/ld1117v33/v-reg-ldo-3-3v-1117-to-220-3/dp/9755837 |
| PCF8563 | £1.52 | 1 | £1.52 | http://uk.farnell.com/nxp/pcf8563p-f4-112/rtc-calendar-8563-dip8/dp/8906122 |
| coin cell holder | £0.69 | 1 | £0.69 | http://uk.farnell.com/keystone/1066/battery-holder/dp/1020673 |
| CR2032 | £0.77 | 1 | £0.77 | http://uk.farnell.com/renata/cr-2032-mfr-1bl/cell-button-3v/dp/1823479 |
| 4xAA battery holder | £1.34 | 1 | £1.34 | http://uk.farnell.com/keystone/2477/battery-holder-pcb/dp/1650684 |
| CM12 X10 | £33.91 | 1 | £33.91 | http://www.uk-automation.co.uk/x10-computer-interface-serial-usb-cm1112-p-997.html |
| Remote control | £4.04 | 1 | £4.04 | http://www.shoppingbank.com/sb/pia/d/giant+universal+remote/pid/8570523/?afid=105749 |
| | | **Total** | £110.87 | |

### 5.4 Further Work

To save battery life in the remote temperature sensor, the XBEE could be interrupt driven, so that while the temperature is not needed, the remote XBEE can go into sleep mode. When the "Temperature" item is displayed, the remote XBEE can be woken up by the coordinator and begin sampling and transmitting the temperature.

Further work could include adding internet facilities. This could be used as an alternative method of checking the outside temperature. It could do this by retrieving the local weather based on a postcode entered by the user. It could also give the weather in more detail, by retrieving the maximum and minimum temperature of the day, the wind speed and the weather. This would reduce the cost of manufacturing the Information System by removing the need for the two XBEE modules, but an internet service would be required.

The internet could also be used for reading the latest news or sending and receiving emails. This would ensure that the user does not feel isolated by easily keeping in touch with friends and family.

Another addition to the Information System could be to add sound. This could consist of a notification sound when a button is pressed, ensuring the user knows that the press was acknowledged by the system.

A useful addition to the program would be to add scheduling of tasks. This would allow the user to input an X10 task, and a date and time, and the Information System would perform the task at the inputted time. An example of this would be to turn all the lights off at a certain time.

The control of central heating could be added using the internal temperature and the time to set the heating. This would increase the complexity of installing the Information System but the functionality would greatly increase.

X10 controllers can also be bought which are specifically designed to open and close curtains on an electric rail [44]. This would require the installation of the electric curtain rail [45].

Further work could allow the user to customise the X10 module information. This would include an option to change the X10 housecode for the devices and to add and remove rooms, adapting the Information System for the specific user's home.

## 6 References

[1] George Demiris PhD, Marilytn J. Rantz PhD, RN, Marjorie Skubic PhD, Myra A. Aud PhD, RNn Harry W. Tyrer Jr PhD, "Home Based Assistive Technologies for Elderly: Attitudes and Perceptions", Department of Health Management and Informatics, University of Missouri-Columbia, Columbia, MO, Sinclair School of Nursing, University of Missouri – Columbia, School of Engineering, University of Missouri-Columbia, 2005.

[2] Ofcom, 2006, "Media Literacy Audit: Report on Media Literacy amongst Older People", [Online], available: http://stakeholders.ofcom.org.uk/binaries/research/media-literacy/older.pdf

[3] Debra Kain, 2010, "Older adults watch more TV than younger people, enjoy it less", [Online], Available: http://www.eurekalert.org/pub_releases/2010-06/uoc--oaw062810.php, [Accessed 16th July 2010]

[4] Helen Hoenig, MD, Donald H. Taylor Jr, PhD, and Frank A. Sloan, PhD, "Does assistive technology Substitute for Personal Assistance Among the Disabled Elderly", American Journal of Public Health, Vol 93, No. 2, 2003.

[5] Frank G. Miskelly, "Assistive technology in elderly care", British Geriatrics society, Age and Ageing; 30: pp. 455 – 458, 2001.

[6] James Gerhart, "Home Automation and Wiring", McGraw-Hill, 1999, page 2.

[7] Smart Home USA, "The X10 Story", [Online], Available: http://www.smarthomeusa.com/info/x10story, [Accessed 16th July 2010]

[8] Vesternet, "X10 vs Z-Wave", [Online], Available: http://www.vesternet.com/X10-vs-z-wave, [Accessed 20th July 2010]

[9] Peter, 2010, January, X10, "Insteon and Z-Wave Price Comparison", [Online], Available: http://www.liveautomatic.com/reviews/home-automation/83-x10-insteon-and-z-wave-price-comparison, [Accessed 20th July 2010]

[10] Peter, 2010, February, "X10 Home Automation Review", [Online], Available: http://www.liveautomatic.com/reviews/home-automation/81-x10-home-automation-review, [Accessed 20th July 2010]

[11]HomeSeer Wiki contributors, 2010, August, "HomeSeer HS2", [Online] Available: http://www.homeseer.com/wiki/index.php/HomeSeer_HS2, [Accessed 20th July 2010]

[12]Let's Automate, "HomeSeer Home Automation Software", [Online], Available: http://www.letsautomate.com/12014.cfm?CFID=157731&CFTOKEN=F31719C8-D40D-4761-B802CD6EC1EE1B4F, [Accessed 20th July 2010]

[13]Mark McCall, 2009, October, "Marmitek EasyTouch Panel10 - Inexpensive X-10 Touchscreen Control", [Online] Available: http://www.automatedhome.co.uk/New-Products/Marmitek-EasyTouch-Panel10-Inexpensive-X-10-Touchscreen-Control.html, [Accessed 20th July 2010]

[14]Bernd Blazynski, "Easy Icon 10RF Universal Remote Control - X10 – Marmitek", [Online], Available: http://www.comm4all.com/en/x10-home-automation/transmitters/p1067_easy-icon-10rf-universal-remote-control-x10-marmitek.html, [Accessed 20th July 2010]

[15]Marmitek, "Easy Icon 10RF", [Online], Available: http://www.marmitek.com/en/manual/9837.pdf, [Accessed 20th July 2010]

[16] Maxim Integrated Products, 2008, "MAX7456
Single-Channel Monochrome On-Screen Display with Integrated EEPROM", [Online], Available: http://www.maxim-ic.com/datasheet/index.mvp/id/5516, [Accessed 20th July 2010]

[17] James O. Hamblem, "Using Large CPLDs and FPGAs for Prototyping and VGA Video Display Generation in Computer Architecture Design Laboratories", [Online], Available: http://tab.computer.org/tcca/NEWS/feb99/hamblem.pdf, [Accessed 10th June 2010]

[18] Parallax Inc, 2009, "Propeller datasheet", Version 1.2, [Online], Available: http://www.parallax.com/Portals/0/Downloads/docs/prod/prop/PropellerDatasheet-v1.2.pdf, [Accessed 11th June 2010]

[19] Christopher Diamond, 2005, "ZigBee vs Bluetooth", [Online], Available: http://homepage.uab.edu/cdiamond/ZigBee%20vs%20Bluetooth.htm, [Accessed 15th June 2010]

[20] Digi International, "XBee® & XBee-PRO® ZB ZigBee® PRO RF Modules", [Online], Available: http://www.digi.com/products/wireless/zigbee-mesh/xbee-zb-module.jsp#overview, [Accessed 15th June 2010]

[21]Jeff Martin, "Propeller Manual", Version 1.1, [Online], Available: http://www.parallax.com/Portals/0/Downloads/docs/prod/prop/WebPM-v1.1.pdf, [Accessed 11th June 2010]

[22] Parallax Inc, "Parallax Software Policy Memo", [Online], Available: http://www.parallax.com/Portals/0/Downloads/docs/plcy/ParallaxSoftwarePolicyv1.0.pdf, [Accessed 11th June 2010]

[23]Parallax Inc, "Propeller Object Exchange Library", [Online], Available: http://obex.parallax.com/, [Accessed 11th June 2010]

[24]Parallax Inc, "Propeller Object License Information", [Online], Available: http://obex.parallax.com/license/, [Accessed 11th June 2010]

[25] Maxim Integrated Products, 2002, May, "Video Basics", [Online], Available: http://www.maxim-ic.com/app-notes/index.mvp/id/734, [Accessed 15th June 2010]

[26] Keith Jack, "Video Demystified", 4th Edition, USA, Newnes, Elsevier, 2005, page 327

[27] Michael Tancock, "Broadcast Television Fundamentals", London, Pentech Press, 1991, page 26

[28] X10 Europe, "About X10 Europe", [Online], Available: http://www.x10europe.com/general/about.htm, [Accessed 16th June 2010]

[29] Smith, Jack R., "Programming the PIC Microcontroller with MPBasic", Elsevier/Newnes, 2005, page 455

[30] Ibrahim, Dogan, "Pic Basics: Programming and Projects", Newnes, 2001, page 165

[31] Jan Axelson, "Serial Port Complete", USA, Lakeview Research, 2000, page 12

[32] Marmitek, "Interface Communication Protocol", [Online], Available: http://www.marmitek.com/en/download/software/cm11_x10_protocol_en/114/2, [Accessed 16th June 2010]

[33] Smart Home Systems Inc, "X10 Theory", [Online], Available: http://www.smarthomeusa.com/info/x10theory/#theory, [Accessed 16th June 2010]

[34] Dallas Semiconductor, "DS18B20 datasheet" [Online], Available: http://datasheets.maxim-ic.com/en/ds/DS18B20.pdf, [Accessed 17th June 2010]

[35] Digi International, 2008, February, "XBee™ ZNet 2.5/XBee-PRO™ ZNet 2.5 OEM RF Modules Product Manual", [Online], Available: http://ftp1.digi.com/support/documentation/90000866_C.pdf, [Accessed 15th June 2010]

[36] Microchip, 2009, August, "MCP9700A datasheet", [Online], Available: http://ww1.microchip.com/downloads/en/DeviceDoc/21942e.pdf, [Accessed 17th June 2010]

[37] NXP Semiconductors, 2010, July, "PCF8563 datasheet", [Online], Available: http://www.nxp.com/documents/data_sheet/PCF8563.pdf, [Accessed 17th June 2010]

[38] Stallings, William, "Data and Computer Communications", Prentice-Hall, 2004, page 137

[39] e-CHIP INFOTEK (P) LTD, "RC5 for Remote Control", [Online], Available: http://www.echip.co.in/rc5.html, [Accessed 19th June 2010]

[40] Tanenbaum, Andrew S, "Computer Networtks", Prentice-Hall, 2003, page 274

[41] Michael Tancock, "Broadcast Television Fundamentals", London, Pentech Press, 1991, page 26

[42] UK Automation, Marmitek, "X10 Appliance Module AM12U", [Online], Available: http://www.uk-automation.co.uk/marmitek-x10-appliance-module-am12u-p-1.html, [Accessed 19th June 2010]

[43] UK Automation, "Marmitek Easyouch Panel10", [Online], Available: http://www.uk-automation.co.uk/marmitek-easytouch-panel10-p-1401.html, [Accessed 19th June 2010]

[44] Mark McCall, 2003, October, "X10 Curtain Controller for Autoglide and Swish", [Online], Available: http://www.automatedhome.co.uk/New-Products/X10-Curtain-Controller-for-Autoglide-and-Swish.html, [Accessed 19th June 2010]

[45] Mark McCall, 2002, May, "Swish Autoglide (K400) – Review", [Online], Available: http://www.automatedhome.co.uk/Reviews/Swish-Autoglide-K400-Review.html, [Accessed 19th June 2010]

## 7 Information System Source Code

**Tv_Text_Demo**

```
CON

  _clkmode = xtal1 + pll16x
  _xinfreq = 5_000_000- 110

  hA = %0110
  hB = %1110
  hC = %0010
  hD = %1010
  hE = %0001
  hF = %1001
  hG = %0101
  hH = %1101
  hI = %0111
  hJ = %1111
  hK = %0011
  hL = %1011
  hM = %0000
  hN = %1000
  hO = %0100
  hP = %1100

  d1  = %0110           '6
  d2  = %1110           '14
  d3  = %0010           '2
  d4  = %1010           '10
  d5  = %0001           '1
  d6  = %1001           '9
  d7  = %0101           '5
  d8  = %1101           '13
  d9  = %0111           '7
  d10 = %1111           '15
  d11 = %0011           '3
  d12 = %1011           '11
  d13 = %0000           '0
  d14 = %1000           '8
  d15 = %0100           '4
  d16 = %1100           '12

  Units_Off       = %0000    'All units off
  Lights_On       = %0001    'All lights on
  On              = %0010
  Off             = %0011
  Dim             = %0100
  Bright          = %0101
  Lights_Off      = %0110    'All lights off
  Ex_Code         = %0111    'Extended code
  Hail_Request    = %1000
  Hail_Acknowledge  = %1001
  Pre_set_Dim1    = %1010
  Pre_set_Dim2    = %1011
  Ex_Data_Transfer  = %1100    'Extended data transfer
  Status_On       = %1101
  Status_Off      = %1110
  Status_Request    = %1111
```

```
  up    = 33                'RC5 code "channel up"
  down  = 32                  'RC5 code "channel down"
  enter = 12                 'RC5 code "stand-by"


  RD_ROM     = $33                              ' 1W ROM commands
  MATCH_ROM  = $55
  SKIP_ROM   = $CC
  SRCH_ROM   = $F0
  ALARM_SRCH = $EC

  CVRT_TEMP  = $44                               ' DS1822 commands
  WR_SPAD    = $4E
  RD_SPAD    = $BE
  COPY_SPAD  = $48
  RD_EE      = $B8
  RD_POWER   = $B4


VAR

  byte sel
  byte code
  byte name[160]
  byte dvc[16]
  byte addr[16]
  byte room[16]
  byte type[16]
  byte dvcnum
  byte rcflag
  byte snum[8]

OBJ

  text    : "tv_text"
  rc      : "genrc5"
  x10     : "x10"
  eeprom  : "Propeller Eeprom"
  in      : "keyboard"
  ow      : "jm_1-wire"
  i2c     : "jm_i2c"
  xb      : "XBee_Object"

PUB start
{{Method initializes objects}}

  rc.init(2)
  text.start(12)
  x10.init(0, 1, ha)
  'in.start(26, 27)
  i2c.init(5, 4)
  main

PRI main | i
{{Method displays Main Menu}}
```

```
  i:=0
  repeat
    rc.flush
    text.out($00)
    showdate(0, 0)
    showtime(35, 0)
    text.str(string(13,"        Information System"))
    text.str(string(13,13,13,"       .  Home Automation",13,13,"       .  Temperature",13,13,"       .  X10
Setup",13,13,"       .  Settings"))
    code := 0
    repeat until (code == enter)
      case sel
        0 : text.str(string($0A,7,$0B,5,"> "))
            text.str(string($0A,7,$0B,7,"2."))
            text.str(string($0A,7,$0B,9,"3."))
            text.str(string($0A,7,$0B,11,"4. "))
        1 : text.str(string($0A,7,$0B,5,"1."))
            text.str(string($0A,7,$0B,7,"> "))
            text.str(string($0A,7,$0B,9,"3."))
            text.str(string($0A,7,$0B,11,"4. "))
        2 : text.str(string($0A,7,$0B,5,"1."))
            text.str(string($0A,7,$0B,7,"2."))
            text.str(string($0A,7,$0B,9,"> "))
            text.str(string($0A,7,$0B,11,"4. "))
        3 : text.str(string($0A,7,$0B,5,"1."))
            text.str(string($0A,7,$0B,7,"2."))
            text.str(string($0A,7,$0B,9,"3 "))
            text.str(string($0A,7,$0B,11,"> "))
      code := wait
      if ((code => 1) & (code =< 4))
        sel := code - 1
      if (code == up)
        ++sel
        if sel == 4
          sel := 0
      if (code == down)
        sel := --sel <# 3

    code := 0
    case sel
      0 : HomeAutomation
      1 : Temperature
      2 : X10Setup
      3 : Settings

PRI HomeAutomation
  text.out($00)
  sel := 0
  text.out($00)
  showdate(0, 0)
  showtime(35, 0)
  text.str(string(13,"     Home Automation", 13,13,"     All Devices",13,13,"     List by room",13,13,"     Return"))
  code := 0
  repeat until (code == enter)
    case sel
      0 : text.str(string($0A,5,$0B,4,">"))
          text.str(string($0A,5,$0B,6," "))
```

```
        text.str(string($0A,5,$0B,8," "))
    1 : text.str(string($0A,5,$0B,4," "))
        text.str(string($0A,5,$0B,6,">"))
        text.str(string($0A,5,$0B,8," "))
    2 : text.str(string($0A,5,$0B,4," "))
        text.str(string($0A,5,$0B,6," "))
        text.str(string($0A,5,$0B,8,">"))
  code := wait
  if code == up
    ++sel
    if sel == 3
      sel := 0
  if code == down
    sel := --sel <#  2

 case sel
  0 : AllDevices
  1 : ChooseRoom
  2 : sel := 0
PRI ChooseRoom

 text.out($00)
 sel := 0
 text.out($00)
 showdate(0, 0)
 showtime(35, 0)
 text.str(string("     Home Automation", 13,13,"     Bedroom",13,"     Living Room",13,"     Dining Room",13,"
Kitchen"))
 code := 0
 repeat until (code == enter)
  case sel
   0 : text.str(string($0A,5,$0B,3,">"))
       text.str(string($0A,5,$0B,4," "))
       text.str(string($0A,5,$0B,5," "))
       text.str(string($0A,5,$0B,6," "))
   1 : text.str(string($0A,5,$0B,3," "))
       text.str(string($0A,5,$0B,4,">"))
       text.str(string($0A,5,$0B,5," "))
       text.str(string($0A,5,$0B,6," "))
   2 : text.str(string($0A,5,$0B,3," "))
       text.str(string($0A,5,$0B,4," "))
       text.str(string($0A,5,$0B,5,">"))
       text.str(string($0A,5,$0B,6," "))
   3 : text.str(string($0A,5,$0B,3," "))
       text.str(string($0A,5,$0B,4," "))
       text.str(string($0A,5,$0B,5," "))
       text.str(string($0A,5,$0B,6,">"))
  code := wait
  if code == up
    ++sel
    if sel == 4
      sel := 0
  if code == down
    sel := --sel <#  3

 DisplayRoom(sel)
```

```
PRI DisplayRoom(r) | dp, j, i, n
{{still needs work}}
  text.out($00)
  showdate(0, 0)
  showtime(35, 0)
  dp := 0
  case r
    0 : text.str(string(13,"      Bedroom",13))
    1 : text.str(string(13,"      Living Room",13))
    2 : text.str(string(13,"      Dining Room",13))
    3 : text.str(string(13,"      Kitchen",13))

  n := 0
  repeat j from 0 to (dvcnum - 1)
    if room[dp] == r
      text.str(string(13,"      "))
      repeat i from (dp*10) to ((dp*10) + 9)
        if name[i]
          text.out(name[i])
        else
          i := (dp * 10) + 9
      n++
    dp++

  sel := 0
  code := 0

  repeat until code == enter

    text.xy(2, 4)
    text.str(string("  "))

    text.xy(2, n + 3)
    text.str(string("  "))

    text.xy(2, sel+3)
    text.str(string("  "))

    text.xy(2, sel+4)
    text.str(string(" >"))

    text.xy(2, sel+5)
    text.str(string("  "))

    code := wait
    if code == up
      ++sel
      if sel == n
        sel := 0
    if code == down
      sel := --sel <#  n - 1

  dp := 0
  repeat j from 0 to (dvcnum - 1)
    if room[dp] == r & j == sel
      j := dvcnum - 1
    dp++
```

```
  dp--

  devcom(dp)

PRI AllDevices | dp, j, i
{{Method displays all devices}}

  sel := 0
  dp := 0
  text.out($00)
  showdate(0, 0)
  showtime(35, 0)
  text.str(string("     Home Automation", 13))
  repeat j from 0 to (dvcnum - 1)
   text.str(string(13,"     "))
   repeat i from (dp*10) to ((dp*10) + 9)
    if name[i]
     text.out(name[i])
    else
     i := (dp * 10) + 9
   dp++

  sel := 0
  code := 0
  repeat until (code == enter)

   text.xy(2, 3)
   text.str(string("  "))

   text.xy(2, dvcnum + 2)
   text.str(string("  "))

   text.xy(2, sel+2)
   text.str(string("  "))

   text.xy(2, sel+3)
   text.str(string(" >"))

   text.xy(2, sel+4)
   text.str(string("  "))
   code := wait
   if code == up
    ++sel
    if sel == dvcnum
     sel := 0
   if code == down
    sel := --sel <#  dvcnum - 1
  devcom(sel)
  sel := 0

PRI devcom(dp) | i

  repeat until (sel == i+1)
   sel := 0
   code:= 0
   rc.flush
   text.out($00)
```

```
    showdate(0, 0)
    showtime(35, 0)
    text.str(string("    "))
    repeat i from (dp*10) to ((dp*10) + 9)
     if name[i]
      text.out(name[i])
     else
      i := (dp * 10) + 9
    text.str(string(13,13,"    On"))
    text.str(string(13,"    Off"))
    if type[dp]
     text.str(string(13,"    Dim"))
     i:=2
    else
     i:= 1
    text.str(string($0A,4,$0B,6,"Return"))
    repeat until (code == enter) & ((sel == 2) | (sel == 3))
     case sel
      0 : text.str(string($0A,3,$0B,3,">"))
          text.str(string($0A,3,$0B,4," "))
          text.str(string($0A,3,$0B,5," "))
          text.str(string($0A,3,$0B,6," "))
      1 : text.str(string($0A,3,$0B,3," "))
          text.str(string($0A,3,$0B,4,">"))
          text.str(string($0A,3,$0B,5," "))
          text.str(string($0A,3,$0B,6," "))
      i : text.str(string($0A,3,$0B,3," "))
          text.str(string($0A,3,$0B,4," "))
          text.str(string($0A,3,$0B,5,">"))
          text.str(string($0A,3,$0B,6," "))
      i+1:text.str(string($0A,3,$0B,3," "))
          text.str(string($0A,3,$0B,4," "))
          text.str(string($0A,3,$0B,5," "))
          text.str(string($0A,3,$0B,6,">"))
     code := wait
     if (code == up)
      ++sel
      if sel == i+2
       sel := 0
     if (code == down)
      sel := --sel <# i+1


     if (code == enter)
      case sel
       0 : x10.do(addr[dp],on)
       1 : x10.do(addr[dp],off)
       i : dim_in(dp)
           code := enter

PRI dim_in(dp) | i, j

  text.out($00)
  showdate(0, 0)
  showtime(35, 0)
  text.str(string(13,"     Dim : "))
  repeat i from 0 to 2
```

```
    rc.flush
    rcflag := rc.gettoggle
    repeat until (rc.gettoggle <> rcflag) & ((rc.getcommand =>0) & (rc.getcommand =<9)) | (rc.getcommand ==
enter)
    if ((rc.getcommand =>0) & (rc.getcommand =<9))
      code := rc.getcommand
      if i == 0
        j := code
      else
        j := ((j*10) + code) <# 100
      text.dec(code)
    if rc.getcommand == enter
      i := 2
  text.str(string("   "))
  text.dec(((j*22)/100))
  x10.dim(addr[dp],((j*22)/100))
  waitcnt(cnt + 50_000_000)

PRI Temperature | m, i
{{Method shows Temperature menu}}
  text.out($00)
  showdate(0, 0)
  showtime(35, 0)
  XB.start(6,7,0,9600)          ' XBee Comms - RX,TX, Mode, Baud
  text.str(string(13,"    Temperature",13,13,"     Inside",13,13,"     Outside",13,13,"   > Return to main menu"))
  rc.flush
  repeat until (rc.getcommand == enter)
    text.str(string($0A,14,$0B,4))
    tin
    text.str(string($0A,14,$0B,6))
    repeat i from 0 to 39
      waitcnt(cnt + 4_000_000)
      if rc.getcommand
        i := 39
    if m <> getminute
      showtime(35, 0)
      showdate(0, 0)
      m := getminute
  XB.stop


PRI tin | status, temp, crc

  ow.init(3)                                    'pin 3 input

  'waitcnt(clkfreq / 10 + cnt)
  status := ow.reset                             ' check for device

  if (status == %10)                            ' good 1W reset
    crc := ow.crc8(@snum, 7)                          ' calculate CRC
    if (crc <> snum[7])                         ' compare with CRC in SN
      text.str(string("    "))
      text.hex(crc, 2)
      text.str(string(" - bad CRC"))
      repeat

    else
```

```
    temp := readtc                          ' read the temperature
    showc(temp)                             ' display in °C

  else
   case status
    %00 : text.str(string("-- Bus short"))
    %01 : text.str(string("-- Bus interference"))
    %11 : text.str(string("-- No device"))

  waitcnt(cnt +100_000_000)
  ow.finalize

PRI readtc | tc

'' Reads temperature from DS1820
'' -- returns degrees C in 0.1 degree units

  ow.reset
  ow.write(SKIP_ROM)
  ow.write(CVRT_TEMP)
  repeat                                  ' let conversion finish
   tc := ow.rdbit
  until (tc == 1)
  ow.reset
  ow.write(SKIP_ROM)
  ow.write(RD_SPAD)
  tc := ow.read                            ' lsb of temp
  tc |= ow.read << 8                          ' msb of temp
  ow.reset

  'tc := ~~tc * 5                          ' extend sign, 0.1° units

  return tc

PRI showc (tc) | t', dp
  '' Show the temperature
  tc <<= 16
  if (tc < 0)
    text.str(string("-"))
    ||tc
  tc >>= 16
  'dp := (tc & $0F) * 625
  t := tc >> 4
  text.dec(t)
  'text.str(string("."))
  'if dp == 625 AND (tc << 16) < 0
  '  text.str(string("0"))
  'text.dec(dp))
  text.out($B0)
  text.out($43)

PRI tout : tempout | bit
 {{reserved for outside temperature}}


  XB.API_Rxtime(100)
  if xb.rxident <> $FF
```

```
   tempout := 0
   repeat bit from 18 to 19
      tempout += XB.dtset(bit)
      if bit == 18
        tempout <<= 8
   if (tempout < 10000)
    text.dec((((tempout*1200)/1023)-500)/10)

PRI X10Setup
{{Method shows X10 setup Menu}}

  repeat until ((code == enter) & (sel == 2))
   rc.flush
   in.start(26, 27)
   sel := 0
   code := 0
   text.out($00)
   showdate(0, 0)
   showtime(35, 0)
   text.str(string(13,"       X10 Setup",13,13,"     Add Device",13,"     Edit Device",13,13,13,"     Return to main
menu"))
   repeat until (code == enter)
    case sel
      0 : text.str(string($0A,3,$0B,4,"> "))
          text.str(string($0A,3,$0B,5,"  "))
          text.str(string($0A,3,$0B,8,"  "))
      1 : text.str(string($0A,3,$0B,4,"  "))
          text.str(string($0A,3,$0B,5,"> "))
          text.str(string($0A,3,$0B,8,"  "))
      2 : text.str(string($0A,3,$0B,4,"  "))
          text.str(string($0A,3,$0B,5,"  "))
          text.str(string($0A,3,$0B,8,"> "))
    code := wait

    if (code == up)
      ++sel
      if sel == 3
        sel := 0
    if (code == down)
      sel := --sel <# 2

    if (code == enter)
      case sel
        0 : AddDevice
        1 : EditDevice


  sel := 2

PRI AddDevice | i, j          ' 'i' is used to point to the byte in 'name', 'j' points to the byte of each device variable
{{Method allows user to add X10 devices}}
  text.out($00)
  j := dvcnum
  i := dvcnum * 10


  if in.present
```

```
   keyName_in(i)
  else
   rcName_in(i)


  Room_in(j)

  Type_in(j)

  Address_in(j)

  waitcnt (cnt + 50_000_000)


{{test input}}

  text.out($00)
  showdate(0, 0)
  showtime(35, 0)
  text.str(string(13,13,"      "))
  repeat i from (j*10) to ((j*10) + 9)
   if name[i]
    text.out(name[i])
   else
    i := ((j*10) + 9)

  text.str(string(13,"      "))
  case room[j]
   0 : text.str(string("Bedroom"))
   1 : text.str(string("Living Room"))
   2 : text.str(string("Dining Room"))
   3 : text.str(string("Kitchen"))

  text.str(string(13,"      "))
  case type[j]
   0 : text.str(string("Appliance"))
   1 : text.str(string("Lamp"))

  text.str(string(13,"      A"))
  text.dec(dvc[j])

  waitcnt(cnt + 150_000_000)
  dvcnum++
  backup

PRI EditDevice | dp, i, j ''dp' - device pointer, 'i' points to the byte of 'name' to display, 'j' limits the repeat for
the amount of devices
{{Method allows user to remove device}}
  dp := 0
  text.out($00)
  showdate(0, 0)
  showtime(35, 0)
  text.str(string("      Edit Device"))
  repeat j from 0 to (dvcnum - 1)
   text.str(string(13,"      "))
   repeat i from (dp*10) to ((dp*10) + 9)
    if name[i]
```

```
      text.out(name[i])
    else
      i := (dp * 10) + 9
  dp++

 rc.flush
 sel := 0
 code := 0
 repeat until (code == enter)

  text.xy(2, 2)
  text.str(string("  "))

  text.xy(2, dvcnum + 1)
  text.str(string("  "))

  text.xy(2, sel+1)
  text.str(string("  "))

  text.xy(2, sel+2)
  text.str(string(" >"))

  text.xy(2, sel+3)
  text.str(string("  "))
  code := wait
  if code == up
    ++sel
    if sel == dvcnum
      sel := 0
  if code == down
    sel := --sel <#  dvcnum - 1

 Edit(sel)

PRI Edit(dp) | i

 sel := 0
 text.out($00)
 showdate(0, 0)
 showtime(35, 0)
 text.str(string("     Name : "))
 repeat i from (dp*10) to ((dp*10) + 9)
  if name[i]
    text.out(name[i])
  else
    i := (dp * 10) + 9
 text.str(string(13,"     Room : "))
 case room[dp]
  0 : text.str(string("Bedroom"))
  1 : text.str(string("Living Room"))
  2 : text.str(string("Dining Room"))
  3 : text.str(string("Kitchen"))
 text.str(string(13,"     Type : "))
 case type[dp]
  0 : text.str(string("Appliance"))
  1 : text.str(string("Lamp"))
 text.str(string(13,"  Address : A"))
```

```
  text.dec(dvc[dp])
  text.str(string(13,"   Delete"))
  code := 0
  repeat until code == enter
   case sel
    0 : text.str(string($0A,1,$0B,1,">"))
        text.str(string($0A,1,$0B,2," "))
        text.str(string($0A,1,$0B,3," "))
        text.str(string($0A,1,$0B,4," "))
        text.str(string($0A,1,$0B,5," "))
    1 : text.str(string($0A,1,$0B,1," "))
        text.str(string($0A,1,$0B,2,">"))
        text.str(string($0A,1,$0B,3," "))
        text.str(string($0A,1,$0B,4," "))
        text.str(string($0A,1,$0B,5," "))
    2 : text.str(string($0A,1,$0B,1," "))
        text.str(string($0A,1,$0B,2," "))
        text.str(string($0A,1,$0B,3,">"))
        text.str(string($0A,1,$0B,4," "))
        text.str(string($0A,1,$0B,5," "))
    3 : text.str(string($0A,1,$0B,1," "))
        text.str(string($0A,1,$0B,2," "))
        text.str(string($0A,1,$0B,3," "))
        text.str(string($0A,1,$0B,4,">"))
        text.str(string($0A,1,$0B,5," "))
    4 : text.str(string($0A,1,$0B,1," "))
        text.str(string($0A,1,$0B,2," "))
        text.str(string($0A,1,$0B,3," "))
        text.str(string($0A,1,$0B,4," "))
        text.str(string($0A,1,$0B,5,">"))
   code := wait
   if code == up
    ++sel
    if sel == 5
     sel := 0
   if code == down
    sel := --sel <#  4

  case sel
   0 : rcname_in(dp*10)
   1 : room_in(dp)
   2 : Type_in(dp)
   3 : Address_in(dp)
   4 : delete(dp)
  backup

PRI keyname_in(i) | j, e, t

  t := i
  repeat j from i to i+9
   name[j]~
  i := t
  text.out($00)
  showdate(0, 0)
  showtime(35, 0)
  text.str(string(13,"     Add Device (Keyboard)",13,13,"     Name: "))
  repeat until (e == $0D) | (i == t+9)
```

```
    'rc.flush
    e := in.newkey
    if e <> $0D
      text.out(e)
      name[i] := e
      i++
  in.stop

PRI rcname_in(i) | j, t, x

  code := 0
  t := i
  repeat j from i to i+9
    name[j]~
  i := t
  text.out($00)
  showdate(0, 0)
  showtime(35, 0)
  text.str(string(13,"     Add Device (Remote)",13,13,"     Name: "))
  x := 12
  repeat until ((code == enter) OR (i == t+9))
    code := wait
    if (code =< 9)
      text.xy(x, 4)
      text.dec(code)
      name[i] := code + 48
      i++
      x++

PRI room_in(j)

  text.out($00)
  showdate(0, 0)
  showtime(35, 0)
  text.str(string(13,"     Add Device",13,13,"     Room:  > Bedroom",13,"             Living Room",13,"
Dining Room",13,"             Kitchen"))
  code := 0
  sel := room[j]
  repeat until ((code == enter))
    case sel
      0 : text.str(string($0A,13,$0B,4,"> "))
          text.str(string($0A,13,$0B,5,"  "))
          text.str(string($0A,13,$0B,6,"  "))
          text.str(string($0A,13,$0B,7,"  "))
          room[j] := sel
      1 : text.str(string($0A,13,$0B,4,"  "))
          text.str(string($0A,13,$0B,5,"> "))
          text.str(string($0A,13,$0B,6,"  "))
          text.str(string($0A,13,$0B,7,"  "))
          room[j] := sel
      2 : text.str(string($0A,13,$0B,4,"  "))
          text.str(string($0A,13,$0B,5,"  "))
          text.str(string($0A,13,$0B,6,"> "))
          text.str(string($0A,13,$0B,7,"  "))
          room[j] := sel
      3 : text.str(string($0A,13,$0B,4,"  "))
          text.str(string($0A,13,$0B,5,"  "))
```

```
        text.str(string($0A,13,$0B,6,"  "))
        text.str(string($0A,13,$0B,7,"> "))
        room[j] := sel

    code := wait

    if (code == up)
      ++sel
      if sel == 4
        sel := 0
    if (code == down)
      sel := --sel <# 3

  sel := 0

PRI Type_in(j)

  sel := 0
  code := 0
  text.out($00)
  showdate(0, 0)
  showtime(35, 0)
  text.str(string(13,"     Add Device",13,13,"     Type:  > Appliance",13,"            Lamp"))
  repeat until ((code == enter))
    case sel
      0 : text.str(string($0A,13,$0B,4,"> "))
          text.str(string($0A,13,$0B,5,"  "))
          type[j] := sel
      1 : text.str(string($0A,13,$0B,4,"  "))
          text.str(string($0A,13,$0B,5,"> "))
          type[j] := sel
    code := wait

    if (code == up)
      ++sel
      if sel == 2
        sel := 0
    if (code == down)
      sel := --sel <# 1

PRI Address_in(j) | i

  text.out($00)
  showdate(0, 0)
  showtime(35, 0)
  dvc[j] := 0
  text.str(string(13,"     Add Device",13,13,"     Addr: "))
  repeat i from 0 to 1
    rc.flush
    rcflag := rc.gettoggle
    repeat until ((rcflag <> rc.gettoggle) AND ((rc.getcommand => 0) AND (rc.getcommand =< 9)))
    rcflag := rc.gettoggle
    if i == 0
      dvc[j] := rc.getcommand * 10
    else
      dvc[j] := dvc[j] + rc.getcommand
    text.dec(rc.getcommand)
```

```
  case dvc[j]
    1  : addr[j] := d1
    2  : addr[j] := d2
    3  : addr[j] := d3
    4  : addr[j] := d4
    5  : addr[j] := d5
    6  : addr[j] := d6
    7  : addr[j] := d7
    8  : addr[j] := d8
    9  : addr[j] := d9
    10 : addr[j] := d10
    11 : addr[j] := d11
    12 : addr[j] := d12
    13 : addr[j] := d13
    14 : addr[j] := d14
    15 : addr[j] := d15
    16 : addr[j] := d16

PRI delete(dp) | i

  repeat i from dp*10 to ((dvcnum-1) * 10) + 9
    name[i] := name[i+10]
  repeat i from dp to dvcnum - 1
    room[i] := room[i+1]
    type[i] := type[i+1]
    dvc[i] := dvc[i+1]
    addr[i] := addr[i+1]
  dvcnum--

PRI Settings

  rc.flush
  text.out($00)
  showdate(0, 0)
  showtime(35, 0)
  text.str(string(13,"        Settings"))
  text.str(string(13,13,13,"       .  Set Time",13,13,"       .  Set Date",13,13,"          Return"))
  code := 0
  sel := 0
  repeat until (code == enter)
    case sel
      0 : text.str(string($0A,7,$0B,5,"> "))
          text.str(string($0A,7,$0B,7,"2."))
          text.str(string($0A,7,$0B,9,"  "))
      1 : text.str(string($0A,7,$0B,5,"1."))
          text.str(string($0A,7,$0B,7,"> "))
          text.str(string($0A,7,$0B,9,"  "))
      2 : text.str(string($0A,7,$0B,5,"1."))
          text.str(string($0A,7,$0B,7,"2."))
          text.str(string($0A,7,$0B,9,"> "))
    code := wait
    if ((code => 1) & (code =< 3))
      sel := code - 1
    if (code == up)
      ++sel
      if sel == 3
        sel := 0
```

```
  if (code == down)
    sel := --sel <# 2

 code := 0
 case sel
  0 : sett
  1 : setd
 sel := 3

PRI sett | h, m
{{shows the set time menu}}

 text.out($00)              'clear display
 showtime(3, 3)             'show current time at X, Y

 h := gethour               'get hour and minute
 m := getminute

 text.colour(2)             'change text colour to yellow
 repeat until code == enter    'repeat until enter is pressed
  text.xy(3, 3)
  text.dec(h>>4 & $03)       'show hour (BCD)
  text.dec(h & $0F)
  repeat until rc.getcommand  'wait until a button is pressed
  code := rc.getcommand
  rc.flush
  if code == down            'increment hour
    ++h
    if (h & $0F) => 10
      h &= $30
      h += 16
    if (h & $3F) => $24      'loop hour
      h := 0
  if code == up              'decrement hour
    if (h & $0F) > 1
      --h
    elseif (h & $30) > 0
      h -= 16
      h |= $09
    else                     'loop hour
      h := $23

 text.xy(3, 3)              'show set hour in white
 text.colour(0)
 text.dec(h>>4 & $03)
 text.dec(h & $0F)

 text.str(string(":"))

 code := 0
 text.colour(2)
 repeat until code == enter    'repeat until enter is pressed
  text.xy(6, 3)
  text.dec(m>>4 & $07)       'show minutes in yellow text (BCD)
  text.dec(m & $0F)
  repeat until rc.getcommand  'wait until enter is pressed
  code := rc.getcommand
```

```
  rc.flush
  if code == down           'increment minute
    ++m
    if (m & $0F) => 10
      m &= $70
      m += 16
    if (m & $7F) => $60      'loop minute
      m := 0
  if code == up             'decrement minute
    if (m & $0F) > 1
      --m
    elseif (m & $70) > 0
      m -= 16
      m |= $09
    else                    'loop minute
      m := $59

  text.xy(6, 3)             'show set minute in white text
  text.colour(0)
  text.dec(m>>4 & $07)
  text.dec(m & $0F)

  settimebcd(h, m, 0)         'save changes of time

PRI setd | date, day, m, y
{{shows the set date menu}}


  text.out($00)               'clear Display
  showdate(3, 3)              'show current date at X, Y
  code := 0                   'empty code (RC5)
  date := getdate             'get date
  day := getday               'get day
  m := getmonth               'get month
  y := getyear                'get year

  text.colour(2)              'change text colour to yellow
  repeat until code == enter    'repeat until the enter button is pressed
    text.xy(3, 3)             'move text cursor to X, Y
    case day                  'case statement shows day in yellow text
      0 : text.str(string("Sun"))
      1 : text.str(string("Mon"))
      2 : text.str(string("Tue"))
      3 : text.str(string("Wed"))
      4 : text.str(string("Thu"))
      5 : text.str(string("Fri"))
      OTHER : text.str(string("Sat"))  'if day is out of range, set to 6
            day := 6
    repeat until rc.getcommand   'wait until a button is pressed
    code := rc.getcommand       'code is equal to the RC5 command
    rc.flush                  'flush the RC5 value
    if code == down             'if down was pressed, day is incremented
      ++day
      if day == 7             'loop if day overflows
        day := 0
    if code == up               'if up was pressed, day is decremented
      --day
```

```
text.xy(3, 3)                'move text cursor to X, Y
text.colour(0)               'set text colour to white
case day                     'show set day in white
 0 : text.str(string("Sun"))
 1 : text.str(string("Mon"))
 2 : text.str(string("Tue"))
 3 : text.str(string("Wed"))
 4 : text.str(string("Thu"))
 5 : text.str(string("Fri"))
 6 : text.str(string("Sat"))



code := 0                    'empty code (RC5)
text.colour(2)               'set yellow text colour
repeat until code == enter   'repeat unitl enter is pressed
 text.xy(7, 3)               'move text cursor to X, Y
 text.dec(date>>4 & $03)     'show date (BCD)
 text.dec(date & $0F)
 repeat until rc.getcommand  'wait until button is pressed
 code := rc.getcommand       'code = RC5 code
 rc.flush                    'flush RC5
 if code == down             'if down is pressed
  ++date                     'increment date
  if (date & $0F) => 10      'if lower 4 bits => 10
   date &= $30               'clear lower bits
   date += 16                'add 1 to upper 3 bits
  if (date & $3F) => $32     'loop date if it reaches 32
   date := 1

 if code == up               'if up is pressed
  if (date & $0F) > 1        'if lower 4 bits > 1
   --date                    'decrement date
  elseif (date & $30) > 0    'if lower 4 bits = 0 and upper 3 > 0
   date -= 16                '- 1 from upper 3 bits
   date |= $09               'make lower 4 bits = 9
  else                       'if date = 0
   date := $31               'loop

text.xy(7, 3)                'move text cursor to X, Y
text.colour(0)               'set text colour to white
text.dec(date>>4 & $03)      'display the set date
text.dec(date & $0F)

code := 0                    'empty code (RC5)
text.colour(2)               'set text colour to yellow
repeat until code == enter   'repeat until enter is pressed
 text.xy(10, 3)              'move text cursor to X, Y
 case m                      'display month
  1 : text.str(string("Jan"))
  2 : text.str(string("Feb"))
  3 : text.str(string("Mar"))
  4 : text.str(string("Apr"))
  5 : text.str(string("May"))
  6 : text.str(string("Jun"))
  7 : text.str(string("Jul"))
```

```
    8 : text.str(string("Aug"))
    9 : text.str(string("Sep"))
    10 : text.str(string("Oct"))
    11 : text.str(string("Nov"))
    OTHER : text.str(string("Dec"))  'if m is out of range, set to 12
          m := 12
  repeat until rc.getcommand    'wait until a button is pressed
  code := rc.getcommand        'code = RC5 code
  rc.flush                    'flush RC5
  if code == down              'increment month
    ++m
    if m == 13                'loop month
      m := 1
  if code == up                'decrement month
    --m

text.xy(10, 3)                 'show set date in white
text.colour(0)
case m
  1 : text.str(string("Jan"))
  2 : text.str(string("Feb"))
  3 : text.str(string("Mar"))
  4 : text.str(string("Apr"))
  5 : text.str(string("May"))
  6 : text.str(string("Jun"))
  7 : text.str(string("Jul"))
  8 : text.str(string("Aug"))
  9 : text.str(string("Sep"))
  10 : text.str(string("Oct"))
  11 : text.str(string("Nov"))
  12 : text.str(string("Dec"))


code := 0
text.colour(2)
text.str(string($20, "20"))     'show year
repeat until code == enter
  text.xy(16, 3)               'show year (BCD)
  text.dec(y>>4 & $0F)
  text.dec(y & $0F)
  repeat until rc.getcommand    'wait until button is pressed
  code := rc.getcommand
  rc.flush
  if code == down              'increment year
    ++y
    if (y & $0F) => 10
      y &= $F0
      y += 16
    if y => $9A                'loop year
      y := 0
  if code == up                'decrement year
    if (y & $0F) > 0
      --y
    elseif (y & $F0) > 0
      y -= 16
      y |= $09
    else
```

```
      y := $99

  text.xy(16, 3)               'show set year in white
  text.colour(0)
  text.dec(y>>4 & $0F)
  text.dec(y & $0F)

  setdatebcd(date, day, m, y)     'save changes of date


PRI wait : c | m, i
{{waits for key press while updating time}}

  rc.flush                   'flush RC5
  m := getminute             'get minute value
  repeat until rc.getcommand   'repeat until button is pressed
    repeat i from 0 to 19      'repeat 20 times
      waitcnt(cnt + 4_000_000) 'wait 50ms
      if rc.getcommand         'if button is pressed
        i := 20                'exit repeat
    if m <> getminute          'if minute value has changed
      showtime(35, 0)          'show time
      showdate(0, 0)           'show date
      m := getminute           'update minute
  c := rc.getcommand           'return RC5 code
  rc.flush                   'flush RC5 code

PRI showtime(x, y) | m, h
{Object shows the time at X, Y position on the screen}

  m := getminute           'get the minute value
  h := gethour             'get the hour value
  text.xy(x, y)            'move text cursor to X, Y
  text.dec(h>>4 & $03)     'display the 10s h
  text.dec(h & $0F)        'display the units of h
  text.str(string(":"))
  text.dec(m>>4 & $07)     'display the 10s of m
  text.dec(m & $0F)        'display the units of m

PRI showdate(x, y) | date, day, m, yr
{Object shows the date and X, Y position on the screen}

  date := getdate          'get the date value
  day := getday            'get the day value
  m := getmonth            'get the month value
  yr := getyear            'get the year value

  text.xy(x, y)            'move text cursor to X, Y

  case day                 'Display the day
    0 : text.str(string("Sun"))
    1 : text.str(string("Mon"))
    2 : text.str(string("Tue"))
    3 : text.str(string("Wed"))
    4 : text.str(string("Thu"))
    5 : text.str(string("Fri"))
    6 : text.str(string("Sat"))
```

```
  text.out($20)           'space

  text.dec(date>>4 & $03)   'display the 10s of date
  text.dec(date & $0F)     'display the units of date
  text.out($20)           'space

  case m                  'display the month
    1 : text.str(string("Jan"))
    2 : text.str(string("Feb"))
    3 : text.str(string("Mar"))
    4 : text.str(string("Apr"))
    5 : text.str(string("May"))
    6 : text.str(string("Jun"))
    7 : text.str(string("Jul"))
    8 : text.str(string("Aug"))
    9 : text.str(string("Sep"))
    10 : text.str(string("Oct"))
    11 : text.str(string("Nov"))
    12 : text.str(string("Dec"))
  text.out($20)           'sapce
  text.str(string("20"))
  text.dec(yr>>4 & $0F)    'display the 10s of yr
  text.dec(yr & $0F)       'display the units of yr

PRI getseconds : s

  s := i2c.getbyte($A3, $02)

PRI getminute : m

  m := i2c.getbyte($A3, $03)

PRI gethour : h

  h := i2c.getbyte($A3, $04)

PRI waitm | m, w

  w := getminute
  repeat until w <> m
    m := getminute

PRI getdate : d

  d := i2c.getbyte($A3, $05)

PRI getday : d

  d := i2c.getbyte($A3, $06)
  d := d & $07

PRI getmonth : m

  m := i2c.getbyte($A3, $07)
  m := ((m>>4 & $01)*10) + (m & $0F)
```

```
PRI getyear : y

  y := i2c.getbyte($A3, $08)

PRI settime (h, m, s)

  h := ((h/10) << 4) + h//10
  m := ((m/10) << 4) + m//10
  s := ((s/10) << 4) + s//10

  i2c.putbyte($A2, $04, h)
  i2c.putbyte($A2, $03, m)
  i2c.putbyte($A2, $02, s)

PRI settimebcd(h, m, s)

  i2c.putbyte($A2, $04, h)
  i2c.putbyte($A2, $03, m)
  i2c.putbyte($A2, $02, s)

PRI setdate (date, day, month, year)

  date := ((date/10) << 4) + date//10
  month := ((month/10) << 4) + month//10 | $80
  year := (((year/10) << 4) + year//10)

  i2c.putbyte($A2, $05, date)
  i2c.putbyte($A2, $06, day)
  i2c.putbyte($A2, $07, month)
  i2c.putbyte($A2, $08, year)

PRI setdatebcd (date, day, month, year)

  i2c.putbyte($A2, $05, date)
  i2c.putbyte($A2, $06, day)
  i2c.putbyte($A2, $07, month)
  i2c.putbyte($A2, $08, year)

PRI backup
{{backs up user entered x10 devices}}
  eeprom.VarBackup(@name, @dvcnum)
PRI restore
{{restores backed-up user entries}}
  eeprom.VarRestore(@name, @dvcnum)
```

**Genrc5**

```
''IR-Receiver (RC5)

VAR
  long  rc5        'rc5 code received
  'byte  flag

PUB init (pin)
{{Launch cog to detect RC5 codes}}
  rxmask := |< pin           'mask input pin
  full := (clkfreq/562)      'calculate bit times
  thalf := full >> 1
  quart := thalf >> 1
  tquart := (thalf + quart)
  cognew(@go, @rc5)          'begin RC5 cog

PUB getcommand : c
{returns command bits}
c := rc5 & $3F             'bit masking puts lowest 6 bits of rc5 into c

PUB getaddress : a
{returns address bits}
a := (rc5 & $7C0) >> 6     'rc5 is shifted right until bits 10 down to 6 are the LSBs, bit masking puts these bits into
a

PUB gettoggle : t
{returns toggle bit}
t := rc5 >> 11             'rc5 is shifted right until toggle bit is LSB

PUB retrc5 : rx
{returns full rc5 code}
  rx := rc5                'returns current code in rc5

PUB getrc5 : rx
{waits for new rc5 code}
  repeat until retrc5 <> rc5
  rx := rc5

PUB flush
  rc5 := rc5 & $8000
  'flag := 0
DAT

       org 0
go     mov addr, PAR
       mov outmask, dira           'backup dira in outmask
       xor outmask, rxmask          'xor rx pin with outmask
       and dira, outmask           'and' 'xor'ed outmask with dira
detect  mov rxcode, #0
       mov count, cnt
       add count, delay
       waitcnt count, 0
       waitpne rxmask, rxmask         'Wait for start bit
       mov start, cnt                 'time start bit pulse
       waitpeq rxmask, rxmask
       mov half, cnt
```

```
      sub half, start              'half - start = start pulse length, should be about 71120 (80MHz * 889us)
      cmp lh, half          wz, wc  'check that the half value is between the upper and lower limits
if_nc   jmp #detect
      cmp uh, half          wz, wc
if_z_or_c jmp #detect


Sb2     mov count, cnt              'wait 3T/4 to detect second start bit
      add count, tquart
      waitcnt count, 0
      test rxmask, ina        wz      'check second start bit
if_nz   jmp #detect                 'jump to detect if zero not detected

      mov _l, #12                 'set up loop iteration
      mov count, cnt                'begin receiving rc5 code
      add count, full               'wait full time period

loop    waitcnt count, full           'wait full time period
      test rxmask, ina        wc      'test rx
if_nc   add rxcode, #1              'add 1 if rx is low
      shl rxcode, #1                'shift rxcode left
      djnz _l, #loop                'loop until full code is received
      shr rxcode, #1
      wrlong rxcode, addr           'write rxcode to hub

      mov time, cnt
      add time, delay
      waitcnt time, 0               'wait a bit
      jmp #detect                   'go back to detect next code


rxmask  long 0                    'IR input pin
lh      long 70000                'lower threshold of half bit time
uh      long 75000                 'upper threshold of half bit time
thalf   long 0                   'calculated half bit time
quart   long 0                    'calculated 1/4 bit time
tquart  long 0                    'calculated 3/4 bit time
full    long 0                   'calculated full bit time
_l      long 0
delay   long 5_000_000
half    long 0                    'timed half bit time
outmask res 1
time    res 1
start   res 1
addr    res 1                      'PAR address
rxcode  res 1
count   res 1
frame   res 1
```

**X10**

```
{{
  hA = %0110
  hB = %1110
  hC = %0010
  hD = %1010
  hE = %0001
  hF = %1001
  hG = %0101
  hH = %1101
  hI = %0111
  hJ = %1111
  hK = %0011
  hL = %1011
  hM = %0000
  hN = %1000
  hO = %0100
  hP = %1100

  d1  = %0110
  d2  = %1110
  d3  = %0010
  d4  = %1010
  d5  = %0001
  d6  = %1001
  d7  = %0101
  d8  = %1101
  d9  = %0111
  d10 = %1111
  d11 = %0011
  d12 = %1011
  d13 = %0000
  d14 = %1000
  d15 = %0100
  d16 = %1100

  Units_Off        = %0000    'All units off
  Lights_On        = %0001    'All lights on
  On               = %0010
  Off              = %0011
  Dim              = %0100
  Bright           = %0101
  Lights_Off       = %0110    'All lights off
  Ex_Code          = %0111    'Extended code
  Hail_Request     = %1000
  Hail_Acknowledge  = %1001
  Pre_set_Dim1     = %1010
  Pre_set_Dim2     = %1011
  Ex_Data_Transfer  = %1100    'Extended data transfer
  Status_On        = %1101
  Status_Off       = %1110
  Status_Request    = %1111}}

CON

  Baud    = 4800   'Baud rate for cm11 computer interface
```

```
VAR
  byte housecode
  byte code

OBJ
  serial : "simple_Serial"

PUB init(rx, tx, house)
{initialise X10 house code and serial communications}
  housecode := house << 4
  serial.init(rx, tx, 4800)

PUB do(device, function) | rx
{method to perform a function on X10 devices}
  repeat until rx == (($04 + (housecode + device))& $FF)        'repeat until checksum received
    serial.tx($04)                                  'transmit header
    serial.tx(housecode + device)                           'transmit code
    rx := serial.rx                                 'receive checksum
    if rx == $A5                                    'if controller is polling system
      serial.tx($C3)                                'respond to polling

  serial.tx($00)                                    'send acknowledgement of checksum
  repeat until rx == $55                                'wait until 'ready to receive' is received
    rx := serial.rx



  repeat until rx == (($06 + (housecode + function))& $FF)       'repeat until checksum received
    serial.tx($06)                                  'transmit code
    serial.tx(housecode + function)                         'transmit code
    rx := serial.rx                                 'receive checksum

  serial.tx($00)                                    'send acknowledgement of checksum
  repeat until rx := $55                                'wait until 'ready to receive' is received
    rx := serial.rx

PUB dim(device, d) | rx
  d := (d << 3)

  repeat until rx == (($04 + (housecode + device))& $FF)        'repeat until checksum received
    serial.tx($04)                                  'transmit header
    serial.tx(housecode + device)                           'transmit code
    rx := serial.rx                                 'receive checksum
    if rx == $A5                                    'if controller is polling system
      serial.tx($C3)                                'respond to polling
  serial.tx($00)                                    'send acknowledgement of checksum
  repeat until rx == $55                                'wait until 'ready to receive' is received
    rx := serial.rx

  repeat until rx == ((($06 + d) + (housecode + %0100))& $FF)    'repeat until checksum received
    serial.tx($06 + d)                              'transmit code
    serial.tx(housecode + %0100)                            'transmit code
    rx := serial.rx                                 'receive checksum
  serial.tx($00)                                    'send acknowledgement of checksum
  repeat until rx := $55                                'wait until 'ready to receive' is received
    rx := serial.rx
```