
Abstract

For children less than three years old, febrile fits are a very common problem. Febrile fits are convulsions that the child may experience, if the child has fever, and if the child is not treated quickly. In this project, a “Febrile Fits Monitor” was developed. It is a setup that involves wireless monitoring of the child’s temperature. If the child starts to have a high temperature, the remote monitoring device will alert the parent, to attend to the child. In this way, febrile fits are stopped before they happen. The setup consists of 2 small devices, the Child’s Unit and the Parent’s Unit. They can operate independently of the PC. In addition, if the parents wish to monitor the temperature of their child while they are at work, they can connect the Parent’s Unit to the PC at home and run a program to have the data constantly sent to a website. They will be able to view a chart of their child’s temperature over the internet.

Acknowledgments

There are times when the problems we face seem insurmountable. Things just do not work as they should be. It is times like this that we should just put our trust in God. He will lead you to people who can help you and advise you. He will lead you to answers.

I would like to thank my friends and family members for giving me support and encouragement throughout my project. David Tan has helped me to make my internet temperature monitoring possible. Rush, Amanda, Fazli, and Kim Piau are some of the people who have provided advice to help me get started quickly, and to overcome difficulties. Thanks also to Jalil who made my PCB's. Lively exchanges over lunch were made with Gim Hoe and Yaneng; for that and their friendship I am deeply grateful.

I am greatly indebted to my supervisor, Prof. Ko Chi Chung, and examiner, Prof. Srinivasan Vikram, for giving numerous criticisms and suggestions for improvement. It has indeed helped to make my system more robust.

To all my loved ones, thanks for all the support.

Contents

Abstract	i
Acknowledgments.....	ii
List of Figures	vii
List of Tables	ix
Chapter 1 Introduction	1
1.1 Febrile Fits in Children	1
1.2 Febrile Fits Monitor	2
Chapter 2 Child's Unit.....	3
Chapter 3 Parent's Unit (Portable device)	5
Chapter 4 Choice of Microcontrollers	6
4.1 PIC Microcontrollers.....	6
4.2 BasicX Microcontrollers	8
Chapter 5 BasicX.....	10
5.1 Making a BasicX Program.....	11
5.2 Execution of the Program	12
5.3 Troubleshooting Programs	12
5.4 Software Switch Debouncing.....	13
5.5 Power Requirements of the BX-24	13
Chapter 6 BasicX-24 Microprocessor	14
6.1 BasicX Processor	14

6.2	SPI EEPROM Chip	15
6.3	Serial Port.....	15
6.4	Low Voltage Monitor.....	15
6.5	Analogue to Digital Converter	16
6.6	BX-24 Technical Specifications	16
Chapter 7	Digital Thermometer.....	18
7.1	DS1620.....	18
7.2	Pin Assignment	18
7.3	3–Wire Communications	18
Chapter 8	Thermistor	20
8.1	Advantages of a Thermistor	20
8.2	Properties of Thermistors.....	20
8.3	Steinhart-Hart Thermistor Equation:	21
8.4	Thermistor Used.....	24
8.5	Calculations.....	26
Chapter 9	AM Transmitter and Receiver.....	28
9.1	TWS-434.....	28
9.2	RWS-434.....	29
9.3	Disadvantages	30
Chapter 10	FM Transmitter and Receiver	31
10.1	Choice of Transmitter and Receiver	31
10.2	Description	32
10.3	Functional Properties	33
10.4	Troubleshooting Automatic Gain Control	36
Chapter 11	Web Management Proposal	38

11.1	Introduction	38
11.2	Operation at Local PC	39
11.3	User Registration from Local PC	39
11.4	User Login from Local PC	42
11.5	User Login at remote PC	43
11.6	Server Temperature Databases	43
11.7	Difficulty Faced	44
Chapter 12	Monitoring over the Internet	45
12.1	Excel Visual Basic	45
12.2	Viewing from a Web Browser	47
12.3	Troubleshooting TimerModule	49
12.4	Troubleshooting FMain	50
Chapter 13	Assembly	51
13.1	Mock-up Boards	51
13.2	Selecting the Boxes	54
13.3	Machining the Boxes	54
13.4	PCB Design	55
13.5	Troubleshooting	57
13.6	Final Assembly	58
Chapter 14	Conclusions	61
Appendix A	Source Codes for Microcontrollers	63
A.1	Code for Transmitter	63
A.2	Code for Receiver	66
A.3	Code for LCD Display	72
A.4	Code for DS1620 Temperature Sensor	74

Appendix B	Source Codes for Excel Visual Basic	77
B.1	Code for Module: StartForm.....	77
B.2	Code for Form: FMain	78
B.3	Code for Module: TimerModule.....	82
Appendix C	HTML Code	85
Appendix D	Circuit Diagrams	86

List of Figures

Figure 1: Block Diagram for Child's Unit	3
Figure 2: Sketch of Child's Unit Implementation.....	4
Figure 3: Block Diagram for Parent's Unit.....	5
Figure 4: Main Components of BX-24	14
Figure 5: R-T Graph for an NTC Thermistor	24
Figure 6: Thermistor 3K3A340I	25
Figure 7: Thermistor Circuit	26
Figure 8: TWS-434 Transmitter.....	28
Figure 9: TWS-434 Pin Diagram	29
Figure 10: RWS-434 Receiver	29
Figure 11: RWS-434 Pin Diagram.....	29
Figure 12: TX2 Transmitter	33
Figure 13: RX2 Receiver	33
Figure 14: TX2 Schematic	35
Figure 15: RX2 Schematic	36
Figure 16: Designing the Form in Excel Visual Basic.....	45
Figure 17: "TempSens.xls" Program in Operation	47
Figure 18: Real-time Temperature Graph	48
Figure 19: Graph Scales Automatically To Fit Data.....	48
Figure 20: Breadboard Setup for Parent's Unit	52

Figure 21: Breadboard Setup for Child's Unit.....	52
Figure 22: Mock-up Board for Parent's Unit (without LCD).....	53
Figure 23: Mock-up Board for Child's Unit	53
Figure 24: Preparing the Parent's Unit for Machining	54
Figure 25: Preparing the Child's Unit for Machining.....	55
Figure 26: PCB for Parent's Unit.....	56
Figure 27: PCB for Parent's Unit Button Board	56
Figure 28: PCB for Child's Unit	57
Figure 29: Inside the Child's Unit.....	59
Figure 30: Inside the Parent's Unit	59
Figure 31: Child's Unit	60
Figure 32: Parent's Unit.....	60
Figure 33: Initial Circuit Diagram for Child's Unit	86
Figure 34: Initial Circuit Diagram for Parent's Unit.....	87
Figure 35: Final Circuit Diagram for Child's Unit	88
Figure 36: Final Circuit Diagram for Parent's Unit.....	89

List of Tables

Table 1: BX-24 Technical Specifications	17
Table 2. DS1620 Command Set.....	19
Table 3: Resistance-Temperature Table	25
Table 4: TWS-434 Parameters	29
Table 5: RWS-434 Parameters.....	30
Table 6: TX2 Pin Descriptions.....	34
Table 7: RX2 Pin Descriptions	36

Chapter 1 Introduction

1.1 Febrile Fits in Children

Febrile fits are fits caused by fever. This usually occurs in children less than three years old. The percentages of children who develop febrile fits vary for different race and geographic location. In Singapore, around 40% of children suffer from febrile fits.

When febrile convulsions begin, the child loses consciousness. Following that, the body, legs and arms go into spasm. The head is thrown backwards and the legs and arms begin to jerk. The skin goes pale and may even turn blue briefly. The attack ends after a few minutes and the shaking stops. The child goes limp, and then normal colour and consciousness slowly return.

If the febrile fits continue for an extended period of time, there could be a risk of brain damage. The best treatment for febrile fits is to lower the child's body temperature. This is usually done by cold sponging.

However, as the saying goes, "Prevention is the best medicine". When the body temperature exceeds a certain value, sponging of the child can be initiated. Medicine like panadol or valium may also be given.

1.2 Febrile Fits Monitor

To solve the problem of febrile fits, a small convenient device has been designed.

This involves two units. The Child's Unit, which is attached to the child, and the Parent's Unit, which the parent can carry while he or she is in the house. Upon doing a search over the internet, certain projects have been noted, which also involve temperature monitoring. The setup involves a desktop computer which records and analyzes the temperature reading. Graphs can also be plotted.

However, to ensure that this project is useful and practical, the desktop computer is first removed from the setup. The whole system is implemented using small microcontrollers. The microcontroller chosen for this project is the "BasicX" microcontroller. The BasicX chip is very much like a mini computer, with its own CPU, RAM, EEPROM, input and output ports. The code can be readily downloaded into the BasicX chip, via a cable connected to the serial port of the PC.

With wires come the threats of strangulation, so care must be taken to minimize loose wires as far as possible. The setup involves a wireless link between the Child's Unit and the Parent's Unit. For these devices to be portable, small battery sources are utilized.

The Parent's Unit is user-friendly. It has a display indicating the temperature, as well as a means to set the threshold fever temperature. When the child's temperature rises beyond the threshold, the alarm at the Parent's Unit will sound. A message "Fever" is also displayed. The parent can then immediately attend to the child.

Chapter 2 Child's Unit

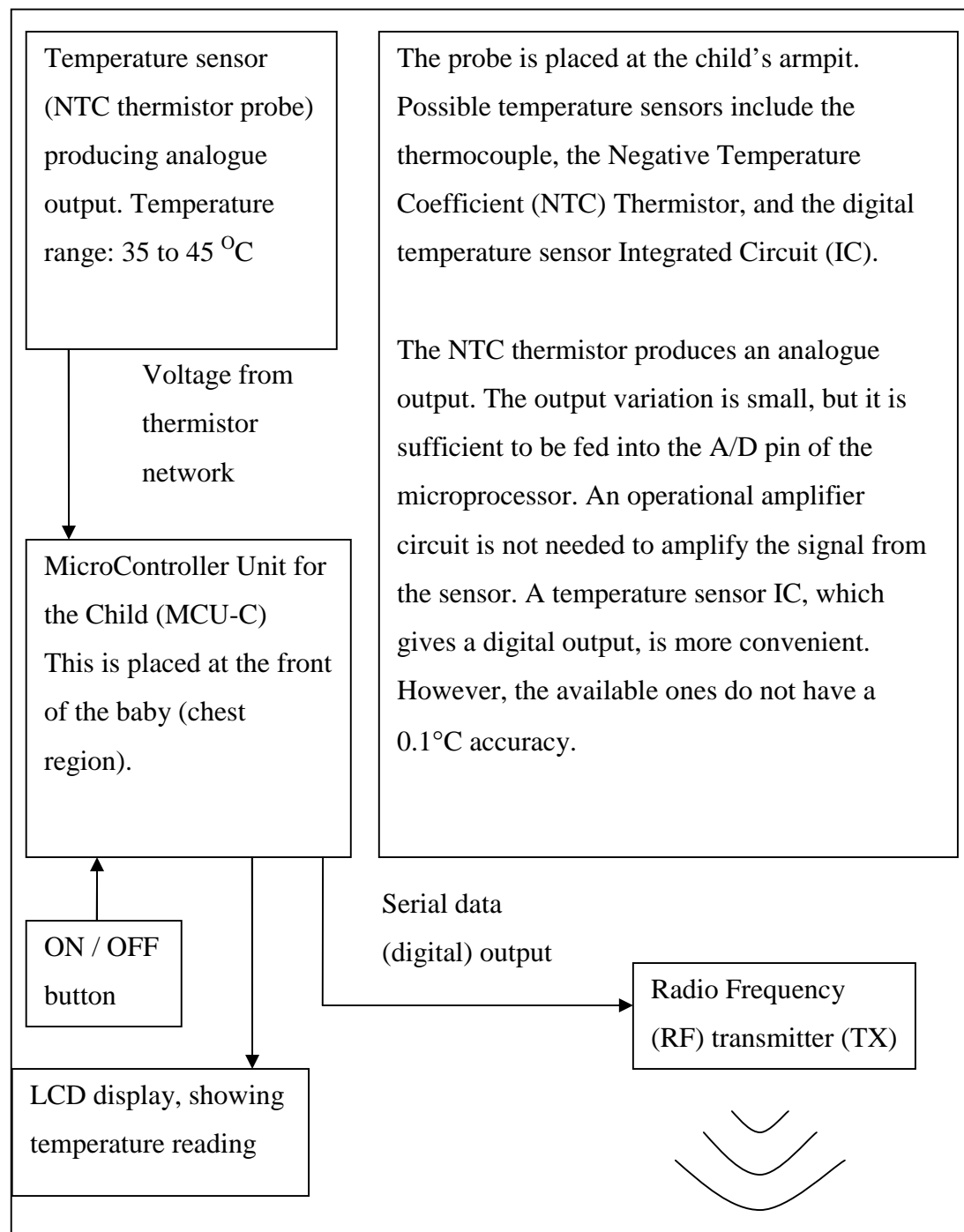


Figure 1: Block Diagram for Child's Unit

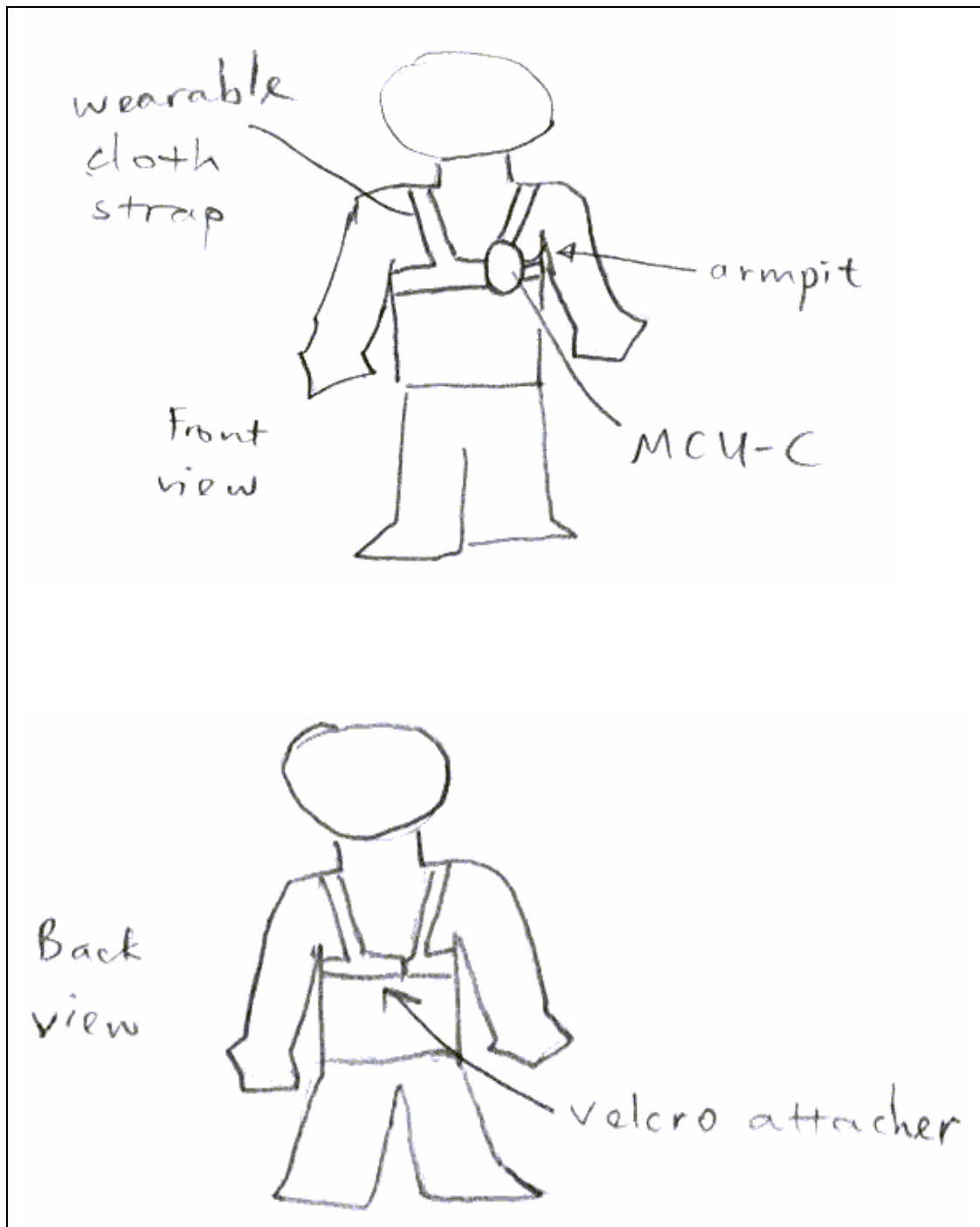


Figure 2: Sketch of Child's Unit Implementation

Chapter 3 Parent's Unit (Portable device)

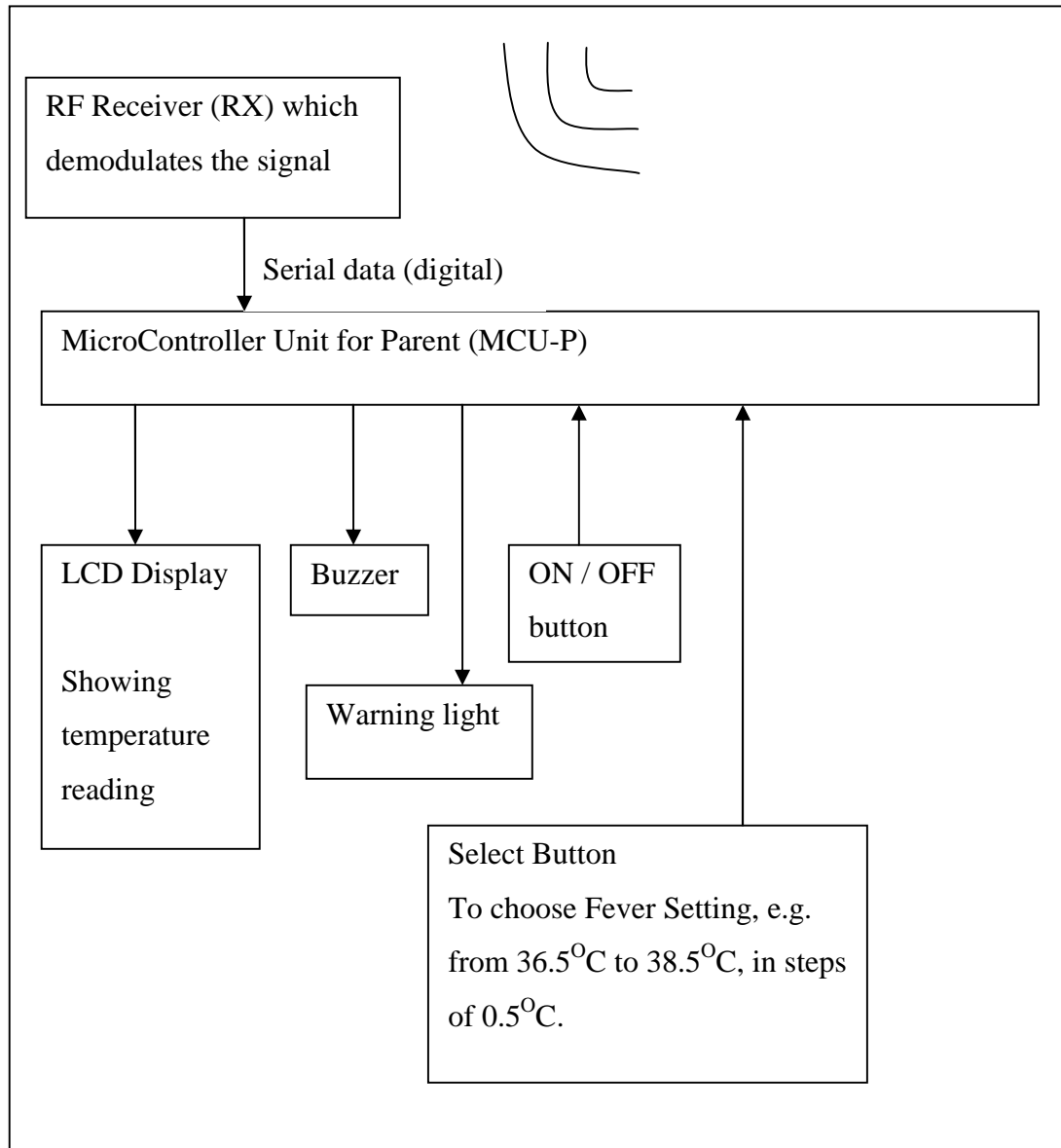


Figure 3: Block Diagram for Parent's Unit

Chapter 4 Choice of Microcontrollers

4.1 PIC Microcontrollers

PIC microcontrollers are readily available in the market. They come in various sizes, with different built-in functionalities. The cost of PIC microcontrollers is relatively low, compared to other microcontrollers. For example, the PIC16F84 costs only US\$7. For one-time-programmable microcontrollers, the cost would be around US\$2 less.

However, the hardware required to program and debug the PIC microcontrollers is not cheap. A programmer board is used to download the program written into the microcontroller. A low-end general purpose programmer board could easily cost US\$200. A high-end one could cost around US\$700. This device is unable to do debugging of the microcontroller. A prototype circuit board is required to do the testing of the microcontroller.

To test the program, Microchip has provided MPLAB ICE (In-Circuit Emulator). MPLAB ICE 2000 costs around US\$1,500, while MPLAB ICE 4000 costs around \$2,500. Additional processor modules are required to allow the system to be configured to emulate different processors. MPLAB ICD (In-Circuit Debugger) is a real-time debugger and programmer for certain PICmicro MCUs.

Microchip provides the program MPLAB, which can be used to download the assembly language code into the PIC microcontroller. This is a free software. Programming in a low level language like assembly language can be very efficient. A set of tasks requiring 1 KByte of memory with assembly language can take up to an equivalent 4 KBytes of memory using the C language. With assembly language, we have more control over the program, and we are able to make more elegant codes.

Unfortunately, programming in this low level language can be extremely tedious. For complicated programs, for example requiring 32 KBytes of memory, the assembler code would be too messy. We may lose a large amount of time debugging the program. A high level language is definitely the better choice for a programmer who wants results fast.

C compilers offer a solution to this problem. The program can be written in C language. The C compiler may integrate with MPLAB. After the C compiler generates the assembler hex code, MPLAB could be used to transfer the hex code into the PIC microcontroller. An example of a good C compiler is ByteCraft's MPC C Compiler for Microchip PIC. This costs more than US\$2000.

On the other hand, there are other inexpensive programmer boards available in the market. One example is the T342 MIG-3 programmer board from MIGadgets, which comes with the T206 MIG C compiler. This costs only S\$88. However, the trouble with these low-end programmer boards is that the hardware does not follow the requirements for the read, write and clocking signals strictly, resulting in a much reduced lifespan for the microcontroller. Furthermore, there was hardly any technical

support for the particular programmer board mentioned earlier. Another example of a low cost programmer board is the K149 from DIY Electronics. More information can be obtained from their website at www.kitsrus.com.

Another option could be to use the PICBasic compiler from MicroEngineering Labs Inc. The PICBasic compiler costs US\$100. An EPIC programming board worth US\$60 is required to download the program into the microcontroller. The PICBasic compiler runs in DOS mode, and uses the Basic language. A crystal, a few capacitors, resistors and a 7805 Voltage Regulator would be required to complete the setup.

4.2 BasicX Microcontrollers

Typical microcontroller applications use C or assembly language. That is why they are expensive to produce and maintain.

NetMedia has produced the BasicX microcontroller. NetMedia has handled the difficult tasks, such as building a multitasking network operating system, language processor, and compiler. The benefit of all this power is not available on most microcontrollers at any price. The BasicX-24 microcontroller is currently being used for this project. It costs just under S\$100.

With this power we can write structured programs in a simple, straightforward language. In fact BasicX's language was modelled after the language used in Microsoft's Visual Basic development system, which is currently a very popular programming language.

Knowledge of Visual Basic is not required to use BasicX. The BasicX language is a subset compatible with the Visual Basic language, and it is possible to write code that will run in both PC and BasicX environments, as long as a common subset is used.

We still need to accommodate differences between operating systems as well as hardware, but it is possible to develop and debug the algorithms in Visual Basic and make use of the same code in BasicX.

NetMedia also provides source code for Visual Basic applications that lets us communicate with the PC from a BasicX application.

Using a Visual Basic development system on the PC side and BasicX as the controller makes a powerful combination. NetMedia recommends Visual Basic 6.0 or higher for codeveloping PC applications and BasicX applications.

Chapter 5 BasicX

BasicX is a complete control system on a chip, combined with a software development environment on a PC-compatible computer running Windows. A BX-24 system combines a BasicX chip with additional devices to make it a standalone computer:

BX-24 Hardware -- In the BX-24 system there is a fast core processor with a ROM to store the BasicX Operating System, 400 bytes of RAM, 32 KBytes of EEPROM, and several I/O devices such as timers, UARTs, ADCs, digital I/O pins, SPI peripheral bus, and more. The BX-24 uses an Atmel AT90S8535 as its core processor.

BasicX Operating System (BOS) -- The BasicX Operating System on-chip provides the multitasking environment that makes the BasicX Chip very powerful. The operating system also contains a high speed BasicX execution engine.

BasicX Development Environment -- BasicX programs are developed on an IBM-PC compatible computer under Windows 95/98/NT/2000. The BasicX Development Environment includes an editor, compiler, various debugging aids, and source code for examples.

The environment incorporates a true 32-bit Windows IDE. There is no reliance on DOS programs hidden behind a Windows shell, which also means there are no hidden 8-character filename limitations.

5.1 Making a BasicX Program

After the program is created, it is compiled. The compiler translates the BasicX source code into an intermediate binary language that the BasicX chip understands, and writes the data to a file (*.BXB). The compiler also takes startup preferences such as pin I/O, RAM configuration information and other important startup parameters and puts them in a preferences file (*.PRF)

```
Source Code --> BasicX Binary file (*.BXB) plus  
                BasicX Preferences (*.PRF)
```

An EXE file on a PC is equivalent to the combination of BXB and PRF files in BasicX.

Once we have these two files, they are the complete representation of the program. These files can be stored on disk, e-mailed, or given away without releasing any source code. This way one could sell BasicX programs without anyone having access to the source code.

The development environment downloads the program directly into the development system or the circuit board.

5.2 Execution of the Program

On a BX-24 microprocessor, once we have a BasicX binary file and preferences file, the code is downloaded into the 32 KB EEPROM. When the BasicX chip starts (after reset), it begins executing instructions from the EEPROM. Since the EEPROM is non-volatile, it is safe from power outages. If the power goes out, the code is still retained in the EEPROM. However, any RAM data that the BasicX chip was working on would inevitably be lost.

5.3 Troubleshooting Programs

When the program was compiled and downloaded into the microprocessor, the processor was not able to function well. It worked sometimes, but usually gave unexpected results. It was found that the size of certain strings exceeded the size allowed by the compiler. The compiler options were then modified to allow 32 character strings.

Furthermore, strings should always be initialized before they are used. If not, the problem comes when using the “Mid” function. A certain part of the string is updated, but the rest of the string is nonsense.

The input buffer at the receiver side must be constantly cleared, while the fever setting is modified. This is so that the microprocessor will not rush through all the accumulated values in the queue.

It was initially intended to send a single byte wirelessly. However, this was not enough due to the larger amount of information when changing from 0.5°C accuracy to 0.1°C accuracy.

5.4 Software Switch Debouncing

Switch debouncing was implemented easily and effectively using software. There was no need for complicated debouncing hardware. Simply, the program checks if a certain push-button is in the down position. After a period of 5ms, the switch bouncing should settle down. The program checks for the down state again. Next, the program waits for the button to be released, before performing the required operations.

5.5 Power Requirements of the BX-24

The BX-24 microprocessor requires a DC power supply in the range of 5.5 V to 15.0 V, which makes it ideal for battery power. Current requirements are 20 mA plus I/O loads, if any.

Chapter 6 BasicX-24 Microprocessor

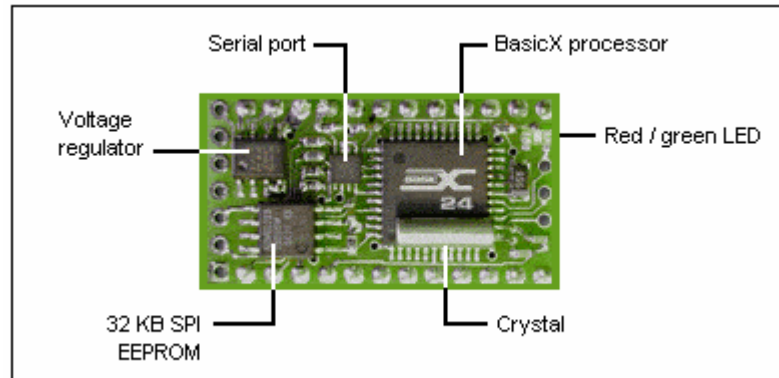


Figure 4: Main Components of BX-24

6.1 BasicX Processor

The BasicX processor is the heart of the BX-24 computer, and is based on an Atmel AT90S8535 chip. This custom-programmed 44 pin chip reads and executes the program stored in the 32 KB EEPROM.

The BX-24 has 16 general purpose I/O lines that are TTL and CMOS compatible. When used for digital I/O, each line can be set to 1 of 4 states -- output high, output low, input tristate (hi-Z) and input with pullup. Up to 8 of the 16 lines can be used alternatively as 10-bit analog to digital converters (ADCs) for sensing analog voltages.

6.2 SPI EEPROM Chip

When a program is written, the SPI (Serial Peripheral Interface) EEPROM chip is where the program is stored. When the BasicX processor is executing, it fetches instructions from this chip. The 32 KByte EEPROM (AT25656) can store approximately 8000 lines of BasicX code, depending on the complexity of the program.

6.3 Serial Port

A high speed 5 volt serial port is provided for connection to modems, PCs, terminals or other controllers. The maximum communication speed is 460 800 baud. A hex inverter (TC7WH04) inverts the serial signals coming to and from the processor's serial port. The hex inverter is also used to isolate the processor's 5 V serial port from the higher voltage levels (typically ± 12 V) present on standard PC serial ports.

The serial port uses 3 wires -- RxData, TxData and DTR. The DTR line is used only for downloading programs. The BasicX Development Environment on the PC has a built-in window that allows 2-way communication with the BasicX serial port.

6.4 Low Voltage Monitor

To prevent the BX-24 from locking up or running erratically during power-on or any other periods of transient or low voltage, the BX-24 employs a low voltage monitor. The monitor is an internal part of the on-board regulator chip.

The monitor constantly checks the system's voltage level. If the BX-24's 5 V supply voltage drops below 4.75 volts, the monitor immediately places the BX-24 in reset until the voltage level rises again.

6.5 Analogue to Digital Converter

The BX-24 includes an 8 channel, 10-bit analogue to digital converter (ADC). The ADC channels are tied to pins 13 to 20, and is an integral part of the processor. All 8 channels can be used either as analogue or digital inputs.

The ADC inputs are 0 V to 5 V level and will not tolerate either higher or negative voltages. For reliable ADC conversions it is recommended that the ground connection of the source voltage (the voltage being measured) share a common ground with the BX-24 ground connections at pin 4 or pin 23.

6.6 BX-24 Technical Specifications

I/O Lines	16 total; 8 digital plus 8 lines that can be ADC or digital
EEPROM for program and data storage	On-board 32 KB EEPROM Largest executable user program size is 32 KBytes
RAM	400 bytes
Analogue to digital converter	8 channels of 10 bit ADC, can also be used as regular digital (TTL level) I/O
ADC sample rate	6 k samples/s maximum
On-chip LEDs	Has a 2-color surface mount LED (red/green), fully user programmable, not counted as I/O line
Program execution speed	60 microseconds per 16 bit integer add/subtract
Serial I/O speed	2400 baud to 460.8 Kbaud on Com1 300 baud to 19 200 baud on any I/O pin (Com3)

Operating voltage range Min/Max	4.8 VDC to 15.0 VDC
Current requirements	20 mA plus I/O loads, if any
I/O output source current	10 mA @ 5 V (I/O pin driven high)
I/O output sink current	20 mA @ 5 V (I/O pin pulled low)
Combined maximum current load allowed across all I/Os	80 mA sink or source
I/O internal pull-up resistors	120 k Ω maximum
Floating point math	Yes
On-chip multitasking	Yes
On-chip clock/calendar	Yes
Built-in SPI interface	Yes
PC programming interface	Parallel or serial downloads
Package type	24 pin PDIP carrier board
Environmental specifications Absolute maximum ratings	Operating temperature: 0 °C to +70 °C Storage temperature: -65 °C to +150 °C

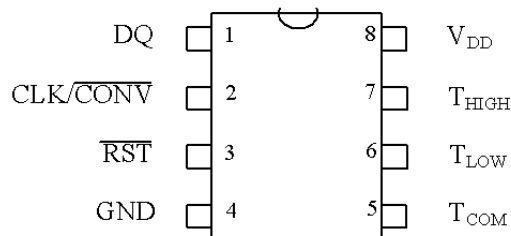
Table 1: BX-24 Technical Specifications

Chapter 7 Digital Thermometer

7.1 DS1620

The DS1620 Digital Thermometer is provided by Dallas Semiconductor. This is an integrated circuit that performs the task of temperature measurement. The DS1620 was initially chosen for this project. It is available from Farnell at \$13.29.

7.2 Pin Assignment



DS1620 8-Pin DIP (300-mil)

7.3 3-Wire Communications

The 3-wire bus is comprised of three signals. These are the $\overline{\text{RST}}$ (reset) signal, the CLK (clock) signal, and the DQ (data) signal. All data transfers are initiated by driving the $\overline{\text{RST}}$ input high. Driving the $\overline{\text{RST}}$ input low terminates communication. A clock cycle is a sequence of a falling edge followed by a rising edge. For data inputs, the data must be valid during the rising edge of a clock cycle. Data bits are output on the falling edge of the clock and remain valid through the rising edge.

When reading data from the DS1620, the DQ pin goes to a high-impedance state while the clock is high. Taking $\overline{\text{RST}}$ low will terminate any communication and cause the DQ pin to go to a high-impedance state.

Data over the 3-wire interface is communicated LSB first. The command set for the 3-wire interface is shown in Table 2.

INSTRUCTION	DESCRIPTION	PROTOCOL	3-WIRE BUS DATA AFTER ISSUING PROTOCOL	NOTES
TEMPERATURE CONVERSION COMMANDS				
Read Temperature	Reads last converted temperature value from temperature register.	AAh	<read data>	
Read Counter	Reads value of count remaining from counter.	A0h	<read data>	
Read Slope	Reads value of the slope accumulator.	A9h	<read data>	
Start Convert T	Initiates temperature conversion.	EEh	Idle	1
Stop Convert T	Halts temperature conversion.	22h	Idle	1
THERMOSTAT COMMANDS				
Write TH	Writes high temperature limit value into TH register.	01h	<write data>	2
Write TL	Writes low temperature limit value into TL register.	02h	<write data>	2
Read TH	Reads stored value of high temperature limit from TH register.	A1h	<read data>	2
Read TL	Reads stored value of low temperature limit from TL register.	A2h	<read data>	2
Write Config	Writes configuration data to configuration register.	0Ch	<write data>	2
Read Config	Reads configuration data from configuration register.	ACh	<read data>	2

Table 2. DS1620 Command Set

Chapter 8 Thermistor

8.1 Advantages of a Thermistor

The DS1620 IC described in the previous section is rather large. It would not be convenient to place it on the child. The large size may cause discomfort to the child. A better alternative would be to use a thermistor. The thermistor probe consists of a small thermistor at the tip, with two wires attached. The thermistor at the tip is only 2mm in diameter. The small size makes it very suitable for temperature monitoring of a human subject.

Furthermore, the DS1620, in normal operation, only gives temperature readings up to a resolution of 0.5 deg C. This is hardly enough for clinical measurement of human body temperature. In addition, it is not a straight-forward process to obtain higher resolutions from the DS1620 IC. Digital temperature sensors with 0.1 deg C accuracy are not readily available in Singapore. This was also another factor in the decision to change from a digital temperature sensor to an analogue temperature sensor.

8.2 Properties of Thermistors

NTC devices are devices whose resistance decreases as their temperature increases. PTC devices are devices whose resistance increases as their temperature increases.

Alpha(α) (Temperature Coefficient), a material characteristic, is defined as the percentage resistance change per degree Centigrade. Alpha is also referred to as the temperature coefficient. For Negative Temperature Coefficient (NTC) Thermistors, typical values of alpha are in the range -3%/°C to -6%/°C.

Beta (β), is the material constant of a NTC thermistor. It is also known as the Sensitivity Index. It provides an indication of the temperature sensitivity of a thermistor over a temperature range in a manner that accounts for the exponential nature of the relationship between the Resistance and Temperature of a thermistor. It is defined as:

$$\beta = \left(\frac{1}{\frac{1}{T_1} - \frac{1}{T_2}} \right) \times \ln \left(\frac{R_1}{R_2} \right)$$

where

- R_1 is the resistance (ohms) of the thermistor at temperature T_1 (Kelvin)
- R_2 is the resistance (ohms) of the thermistor at temperature T_2 (Kelvin),
- \ln is log function to base 'e' (inverse of exponential function)
- β has units of Kelvin.

8.3 Steinhart-Hart Thermistor Equation:

The Steinhart-Hart thermistor equation is named after two oceanographers associated with Woods Hole Oceanographic Institute on Cape Cod, Massachusetts. The first publication of the equation was by I.S. Steinhart & S.R. Hart in "Deep Sea Research" vol. 15 p. 497 (1968).

The equation is derived from mathematical curve-fitting techniques and examination

of the Resistance versus Temperature characteristic of thermistor devices. In particular, using the plot of the natural log of resistance value, $\ln(R)$ versus $(1/T)$ for a thermistor component to consider $(1/T)$ to be a polynomial in $\ln(R)$, an equation of the following form is developed:

$$\frac{1}{T} = A_0 + A_1(\ln(R)) + \dots + A_N(\ln(R))^N$$

(where T is the temperature in Kelvin, and $A_0 \dots A_N$ are polynomial coefficients that are mathematical constants.)

The order of the polynomial to be used to model the relationship between R and T depends on the accuracy of the model that is required and on the non-linearity of the relationship for a particular thermistor. It is generally accepted that use of a third order polynomial gives a very good correlation with measured data, and that the "squared" term is not significant.

The equation then is reduced to a simpler form, and it is generally written as:

$$\frac{1}{T} = A + B(\ln(R)) + C(\ln(R))^3$$

where: A , B , and C are constant factors for the thermistor that is being modelled.

This is the Steinhart-Hart equation, with Temperature as the main variable. The equation is presented explicitly in resistance. Before summarizing the situation, some general points of relevance in understanding the practical issues associated with it are discussed:

The equation is relevant for the complete useful temperature range of a thermistor.

The coefficients A , B , and C are constants for the individual thermistors. Unlike

Alpha and Beta they should not be regarded as material constants. The A, B, and C constants are established for individual thermistors in a particular temperature range as follows:

The equation is considered for three temperature points in the range – usually at the low end, the middle and the high end of the range. This ensures best fit along the full range. (The smaller the temperature range, the better the calculations will match measured data.) The temperature values are usually taken to be 0°C, 25 °C and 70 °C therefore these values are used to illustrate the principle.

Precisely controlled measurements of temperature and associated resistance value of the thermistor are made in a temperature controlled medium at these three calibration points. These accurately measured values of Resistance and Temperature are inserted into the equation to form three simultaneous equations as follows: (note: 0°C = 273.15K)

$$\frac{1}{T_0} = \frac{1}{273.15} = A + B(\ln(R_0)) + C(\ln(R_0))^3$$

$$\frac{1}{T_{25}} = \frac{1}{298.15} = A + B(\ln(R_{25})) + C(\ln(R_{25}))^3$$

$$\frac{1}{T_{70}} = \frac{1}{343.15} = A + B(\ln(R_{70})) + C(\ln(R_{70}))^3$$

Since the resistance values are measured numerical quantities, the equations are a system of three simultaneous equations in three unknowns namely A, B and C. The values for A, B and C can be found by standard mathematical techniques for solving simultaneous equations, or by use of analytical software tools.

This is a brief summary of the origins and techniques used to derive the A, B and C coefficients for thermistor components. These values are sometimes referred to as the “Steinhart Coefficients” for a thermistor. Thermistor manufacturers publish data for these coefficients for their thermistor products.

The Steinhart-Hart equation produces a good approximation to the relationship between T and R for the complete range of a thermistor based on data from just three calibration points. The following figure shows the R-T Graph for BetaTHERM part # 0.1K1A Thermistors. The full Temperature range is from -80 °C to +100 °C. The difference between values calculated from the Steinhart-Hart equation and the actual measurements should typically be less than +/- 0.01 °C.

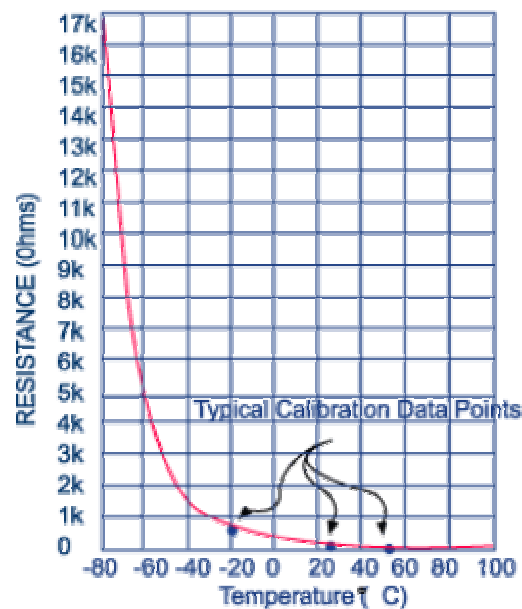


Figure 5: R-T Graph for an NTC Thermistor

8.4 Thermistor Used

The thermistor 3K3A340I from BetaTherm was used. It cost \$11.33, and was obtained from Farnell.

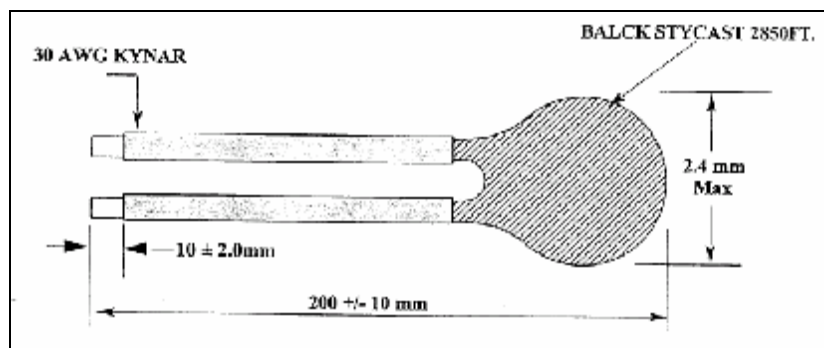


Figure 6: Thermistor 3K3A340I

Electrical Specifications:

Resistance @ 25°C: 3,000 Ohms
Tolerance: $\pm 0.2^\circ\text{C}$ from 0°C to 70°C
Operating Range: -55°C to $+150^\circ\text{C}$
Beta Value 0/50: 3892 $\pm 1.0\%$
Alpha Value @ 25°C: $-4.39\%/^\circ\text{C}$

Curve 3	R @ 25°C :	3000.00
Temp	Multiplier	R nominal
-50	66.7820	200346.00
-45	47.0610	141183.00
-40	33.5668	100700.40
-35	24.2193	72657.90
-30	17.6682	53004.60
-25	13.0242	39072.60
-20	9.8976	29092.81
-15	7.2895	21868.56
-10	5.5298	16589.25
-5	4.2314	12694.32
0	3.2651	9795.29
5	2.5396	7618.69
10	1.9904	5971.09
15	1.5714	4714.21
20	1.2494	3748.13
25	1.0000	3000.00
30	0.8056	2416.79
35	0.6530	1959.06
40	0.5325	1597.51
45	0.4367	1310.06
50	0.3601	1080.29
55	0.2985	895.52
60	0.2487	746.12
65	0.2082	624.70
70	0.1752	525.49
75	0.1480	444.03
80	0.1256	376.85
85	0.1071	321.17
90	0.0916	274.84
95	0.0787	236.10
100	0.0679	203.59
105	0.0587	176.19
110	0.0510	153.02
115	0.0444	133.35
120	0.0389	116.58
125	0.0341	102.25
130	0.0300	89.95
135	0.0265	79.36
140	0.0234	70.22
145	0.0208	62.31
150	0.0185	55.44

Table 3: Resistance-Temperature Table

8.5 Calculations

Two approximate resistance values are as shown:

35°C: 2k Ω

45°C: 1.3k Ω

The thermistor is placed in series with an accurate 1.5k ohm resistor, to get a voltage divider circuit.

High voltage = $2/(2+1.5)*5 = 2.86V$

Low voltage = $1.3/(1.3+1.5)*5 = 2.32V$

Difference = 0.54V

The BasicX chip at the receiver side does A/D conversion to obtain the voltage reading. This is done without amplification. The voltage across the thermistor is calculated. The resistance of the thermistor is then obtained.

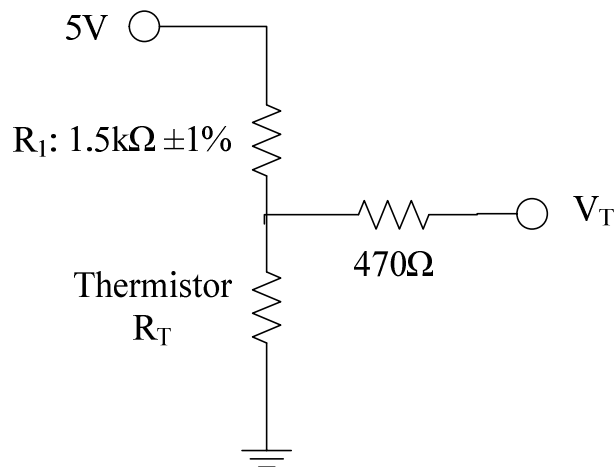


Figure 7: Thermistor Circuit

$$V_T = \frac{R_T}{R_T + R_1} \times 5$$

$$R_T = \frac{V_T R_1}{5 - V_T}$$

Following that, the temperature value is calculated using the Steinhart-Hart Thermistor Equation.

$$\frac{1}{T} = A + B(\ln(R)) + C(\ln(R))^3$$

$$A = 1.405027 \times 10^{-3}$$

$$B = 2.369386 \times 10^{-4}$$

$$C = 1.012660 \times 10^{-7}$$

These constants are based on the three temperature reference points, 0°C, 25°C and 70°C.

Chapter 9 AM Transmitter and Receiver

The transmitter TWS-434 and receiver RWS-434 from Reynolds Electronics are extremely small, and are excellent for applications requiring short-range RF remote control or monitoring. The transmitter module is only 1/3 the size of a standard postage stamp, and can easily be placed inside a small plastic enclosure. These were initially used in the design.

9.1 TWS-434

The transmitter TWS-434 uses Amplitude Modulation (AM). The transmitter output is up to 8mW at 433.92MHz with a range of approximately 120 metres (open area) outdoors. Indoors, the range is approximately 60 metres, and will go through most walls.



Figure 8: TWS-434 Transmitter

The TWS-434 transmitter accepts both linear and digital inputs, can operate from 2 to 12 Volts-DC, and makes building a miniature hand-held RF transmitter very convenient. The TWS-434 is approximately 1/3 the size of a standard postage stamp.

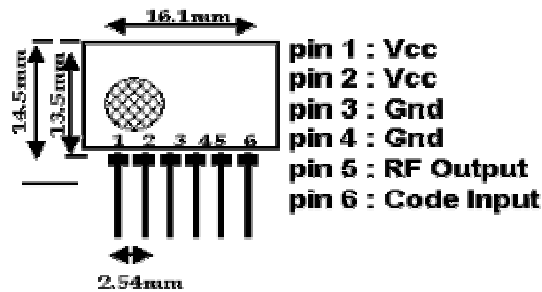


Figure 9: TWS-434 Pin Diagram

Symbol	Parameter	Condition	Min	Typ	Max	Unit
Vcc	Supply Voltage		2.0	-	12.0	V
I _p	Peak Current	2V / 12V	-	1.64 / 19.4	-	mA
V _h	Input High Voltage	I _{data} = 100uA (High)	V _{cc} -0.5	V _{cc}	V _{cc} +0.5	V
V _l	Input Low Voltage	I _{data} = 0 uA (Low)	-	-	0.3	V
F _o	Operating Frequency		433.90	433.92	433.94	MHz
T _r / T _f	Modulation Rise / Fall Time	External Coding	-	-	100 / 100	uS
P _o	RF Output Power – Into 50Ω	V _{cc} = 9 to 12 V V _{cc} = 5 to 6V	-	16 14	-	dBm
D _r	Data Rate	External Coding	-	2.4K	3K	Bps

Table 4: TWS-434 Parameters

9.2 RWS-434

The receiver also operates at 433.92MHz using Amplitude Modulation, and has a sensitivity of 3uV. The RWS-434 receiver operates from 4.5 to 5.5 volts-DC, and has both linear and digital outputs.



Figure 10: RWS-434 Receiver

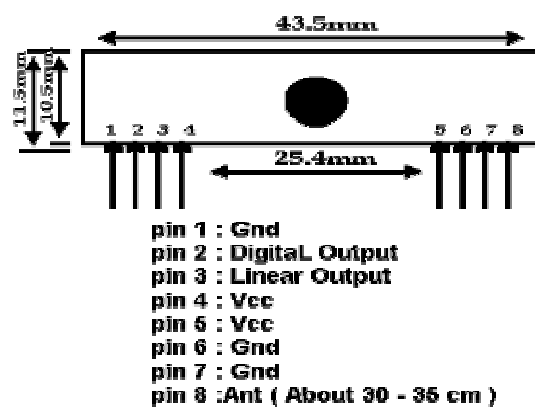


Figure 11: RWS-434 Pin Diagram

For maximum range, the recommended antenna should be approximately 35cm long. On the internet, it was found that certain modules were tested using a 35cm, solid, 24 gauge (0.51mm diameter) hobby type wire, and reached a range of over 120 metres. Actual results for this project may vary depending on the current surroundings.

Symbol	Parameter	Condition	Min	Typ	Max	Unit
Vcc	Supply Voltage		4.5	5	5.5	V
It	Operating Current		-	3.5	4.5	mA
	Channel Width	+ / - 500				kHz
Rd	Data Rate				3k	Bps
Vdat	Data Out	Idata = +200 uA (High)	Vcc-0.5	-	Vcc	V
		Idata = -10 uA (Low)	-	-	0.3	V

Table 5: RWS-434 Parameters

9.3 Disadvantages

The range of these AM modules was tested. The maximum range was found to be around 15 metres, from one room to another room. This is definitely not sufficient for standard home applications. The reason for the short range is that AM modules are much more susceptible to noise, than FM modules. Because of this, the AM radio frequency modules were replaced with FM radio frequency modules.

Chapter 10 FM Transmitter and Receiver

10.1 Choice of Transmitter and Receiver

The Infocomm Development Authority of Singapore (iDA) approves radio equipment and devices in the frequency bands of 433.79-434.79MHz, 866.10-869.00MHz, 924-925MHz, 2,400.00-2,483.50MHz and 24,000-24,250MHz.

For the first 3 bands, the maximum permissible output power emission is 10mW, while in the last 2 bands, the permissible emission is 100mW.

In choosing the radio frequency (RF) modules, care was taken to make sure that the frequency fell in the allowed frequency band, and the transmitter power did not exceed the permissible emission. A long transmission range is indeed advantageous. However, as the hardware will only be used within the house, the range need not be very far. Nevertheless, a large transmission range will buffer the effect of electromagnetic interference or obstacles in the transmission path.

The voltages of the RF modules are also very important. Since the existing hardware components all require 5V for operation, it is much simpler to get RF modules that operate at also 5V.

2nd Generation Radiometrix FM transmitter and receiver were used. These operate at 433MHz and 5V. The transmitter and receiver were purchased from Farnell, and cost \$48.81 and \$100.83 respectively.

10.2 Description

The TX2 and RX2 data link modules are a miniature PCB mounting UHF radio transmitter and receiver pair which enable the simple implementation of a data link at up to 40 Kbit/s at distances up to 75 metres in-building and 300 metres open ground.

The actual range was tested in a building, and was found to be around 50 metres. This should be quite good for a home application.

Available for operation at 433.92 MHz in Europe and Singapore, and 418.00 MHz in the U.K., both modules combine full screening with extensive internal filtering to ensure EMC compliance by minimising spurious radiations and susceptibilities. The TX2 and RX2 modules will suit one-to-one and multi-node wireless links in applications including car and building security, EPOS and inventory tracking, remote industrial process monitoring and computer networking.

Because of their small size and low power requirements, both modules are ideal for use in portable, battery-powered applications such as hand-held terminals.



Figure 12: TX2 Transmitter



Figure 13: RX2 Receiver

10.3 Functional Properties

The TX2 transmitter module is a two stage, SAW controlled FM transmitter operating between 2V and 6V and is available in 433.92MHz and 418.00 MHz versions. The 433.92 MHz unit is type-approved to ETS 300-220 for European use and delivers nominally +9dBm from a 5V supply at 12mA, while the 418.00 MHz unit has MPT 1340 type-approval for U.K. use and delivers -3dBm from a 5V supply at 5mA. Both modules measure 12 x 32 x 3.8 mm.

The RX2 module is a double conversion FM superheterodyne receiver capable of handling data rates of up to 40kbit/s. The SIL style RX2 receiver measures 17.5 x 48 x 4.5 mm. It will operate from a supply of 3-6V and draws 14mA when receiving. A fast-acting carrier detect and a power-up enable time of less than 1ms allows effective duty cycle power saving and a -107 dBm sensitivity. This, combined with a SAW front-end filter results in an excellent RF performance and EMC conformance.

RF GND (pin 1)	RF ground pin, internally connected to the module screen and pin 4 (0V). This pin should be connected to the RF return path (e.g. coax braid, main PCB ground plane etc.)
RF out (pin 2)	50Ω RF output to the antenna, it is DC isolated internally.
Vcc (pin 3)	+ve supply pin. The module will generate RF when the Vcc supply is present. Max ripple content 0.1V _{P-P} . A 100nF de-coupling ceramic capacitor is may be used.
0V (pin 4)	Supply ground connection, connected to pin 1 and screen.
TXD (pin 5)	This DC coupled modulation input will accept either serial digital data (0V to Vcc levels) or high level linear signals. Input impedance is 100kΩ.

Table 6: TX2 Pin Descriptions

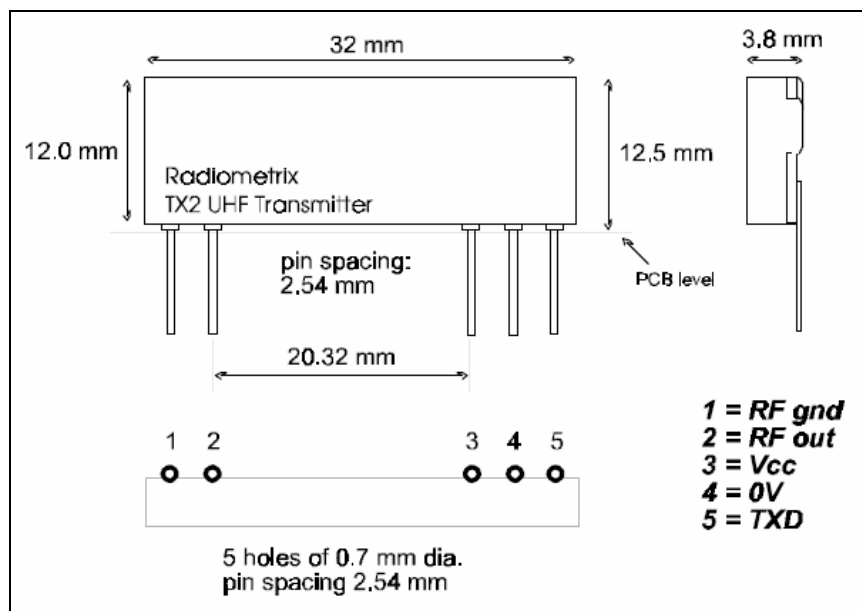


Figure 14: TX2 Schematic

RF in (pin 1)	50Ω RF input from the antenna, it is DC isolated internally.
RF GND (pin 2)	RF ground pin, internally connected to the module screen and pin 4 (0V). This pin should be connected to the RF return path.
CD (pin 3)	The Carrier Detect may be used to drive an external PNP transistor to obtain a logic level carrier detect signal. If not required it should be connected to pin 5 (Vcc).
0volt (pin 4)	Supply ground connection, connected to pin 1 and screen.
Vcc (pin 5)	+ve supply pin. +3.0V to +6.0V @ <17mA . The supply must be clean < 2mV _{P-P} ripple. A 10μF de-coupling capacitor and 10Ω series resistor is recommended if a clean supply is not available.
AF (pin 6)	This is a buffered and filtered analogue output from the FM demodulator. It has a standing DC bias of 1.2V and 400mV _{P-P} base band signal. It is useful as a test point or to drive linear decoders. Load impedance should be > 2kΩ and < 100pF.

RXD (pin 7)	This digital output from the internal data slicer is a squared version of the signal on pin 6 (AF). It may be used to drive external decoders. The data is true data, i.e. as fed to the transmitter. Load impedance should be $> 1k\Omega$ and $< 1nF$
-------------	---

Table 7: RX2 Pin Descriptions

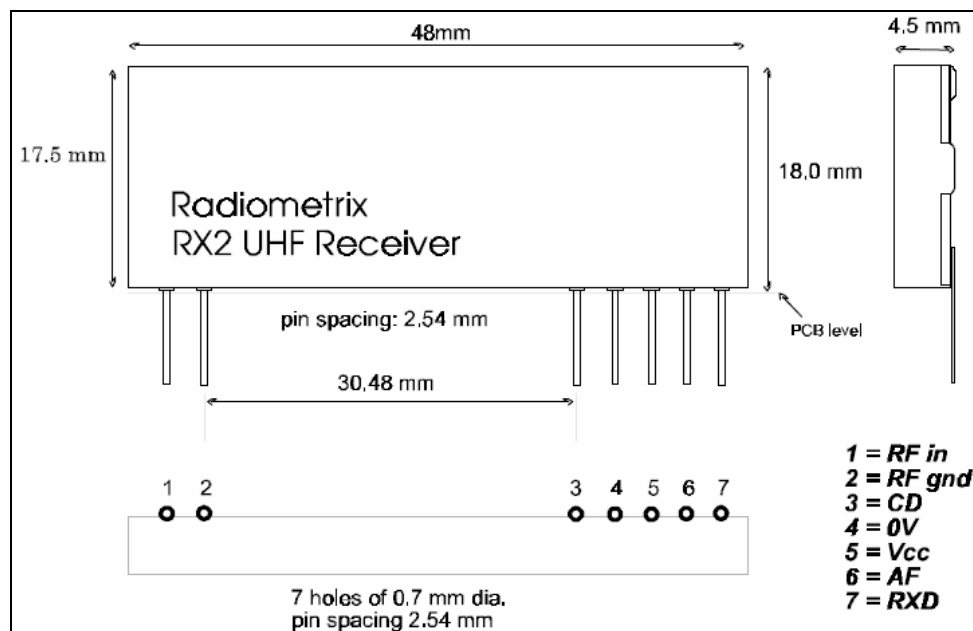


Figure 15: RX2 Schematic

10.4 Troubleshooting Automatic Gain Control

In Automatic Gain Control (AGC), the FM receiver IC will detect the envelope of any signal that it receives. When the FM transmitter IC does not send a signal, the receiver will amplify the received signal. That was the reason why the receiver was outputting nonsense most of the time.

Only when there is a short burst of serial signal transmission, the receiver outputs the correct signal. However, nonsense data was constantly being sent from the receiver

to the microcontroller. This caused the microcontroller to be unable to time correctly, according to the serial interface.

One solution is to use the Carrier Detect signal of the receiver. Unfortunately, the transmitter IC is always on, so there is always a carrier.

The easiest and most effective solution is to send a header, before the actual data. For digital data, it would consist of a sequence of bits. In this case, the microcontroller handles the serial interface, at a higher level. The header would then consist of a sequence of bytes.

However, this still does not solve the problem of the serial timing. It was found that by sending two headers, with a short pause of around 3 bytes duration (300 baud) in between them, the microcontroller would then be able to time correctly, and decode the serial data. When no data is received, the receiver gain is high. Once the first header is detected by the receiver, the receiver will reduce its gain to an appropriate level. The first header cannot be decoded by the microcontroller. The short pause allows the microcontroller on the receiver side to adjust its timing for the serial data. The receiver does not amplify the signal yet, as it takes some time before the Automatic Gain Control starts to take effect. By the time the second header comes in, the microcontroller is able to decode it. The microcontroller will then receive the actual data, for further processing.

Because of the Automatic Gain Control, an analogue signal cannot be transmitted. For this reason, a microcontroller is also needed at the transmitter (Child's Unit).

Chapter 11 Web Management Proposal

11.1 Introduction

The Parent's Unit is able to receive the temperature data from the Child's Unit remotely, from a distance of about 50 metres in-building. This works fine if the parent is also at home with the child. The parent can carry along the Parent's Unit with him, as long as he stays within the house.

However, the problem arises if the parents need to go out to work. They would not be able to take the Parent's unit out of the house. There should be some way for them to know the state of their child at home. The solution is to introduce internet functionality to the device. The Parent's Unit can be connected to a laptop or desktop PC, which has an internet connection. Once the connection to a particular FTP server is established, a program in the PC will then start uploading the temperature data to the FTP server. The parents would be able to view the temperature data and graph over the internet, even when they are at their workplace.

This chapter describes how the project could be developed to a professional and commercially viable state. This proposal caters for the case of multiple users.

On the local PC, the Visual Basic 6 program, **TempSens.exe** will capture data from the serial port, which is connected to the temperature measuring device. The data

will then be stored in an Access database file (Access 97), **TempLocal.mdb**. The incoming data can be uploaded to a server, for viewing on a remote PC.

11.2 Operation at Local PC

The user need not login to the server. The Visual Basic 6 program, **TempSens.exe**, will capture data from the serial port, which is connected to the temperature measuring device. The temperature data will come in every minute.

The program will plot a graph on the local PC. It will save the readings into the Access database file, **TempLocal.mdb**.

TempLocal.mdb

Date (from local PC)	Time (from local PC)	Temperature (from serial port)
2003-12-21	10:00	37
2003-12-21	10:01	37.2
2003-12-21	10:02	37.4
2003-12-21	10:03	37.2
2003-12-21	10:04	37

By default, the program will store up to 7 days of temperature data. The user can choose to delete all the past records, or to set the number of days to store the records.

11.3 User Registration from Local PC

The user has to switch on the device, which will then output the Product Serial Number and the Product Code, to the Visual Basic program. The Product Code is

different from the Product Serial Number. The Product Code is not known to the user.

The user registration is done via internet connection, using the same program, **TempSens.exe**. First, internet connection must be established. The user selects “Register” from the menu. He is shown a dialog box with the following fields:

User Information: Fields that are to be completed: (* Required)

* First Name

* Last Name

* Company

* Product Serial No. (Product Serial No. is stated on the product)

* Email

Country

State

Telephone No.

Shop where product was purchased

The program, **TempSens.exe**, will first check to see if the Product Serial No. that is entered by the user matches with that given by the device. If they match,

TempSens.exe will continue to send data over the internet.

The User Information is sent from the Visual Basic program to the internet server, e.g. http://www.TempSens_website.com. The server will check the Product Serial Number and the Product Code. The server has a database of valid Product Serial Number and Product Code pairs, **Serial_Code.mdb**. This database, and all other

databases, should be protected, so that people cannot access them. Only the web administrator can access them.

Serial_Code.mdb

Product Serial No.	Product Code
TS123001	378FD9VJ
TS123002	JNVKD94H
TS123003	83BV75NT
TS123004	WUR7VND9
TS123005	ZNC4J4O8

If the Visual Basic program has provided the correct information, the server makes a sub-directory, using the Product Serial No. This will then store the temperature measurement data later on.

The user is shown the next dialog box. The user enters the following info:

User Name and Password.

The server checks if the User Name is unique. If so, it is accepted, and the User Name and Password are stored into a database, **Password.mdb**.

Password.mdb

User Name	Password	Product Serial No.
chance	depdep123	TS123001
alextan	cavcav123	TS123002
greenp	teatea123	TS123003

gopalbn	frodosam	TS123004
maggie	simpson	TS123005

11.4 User Login from Local PC

The device can be connected, and running in the beginning. The user runs the program, TempSens.exe, and chooses “Login to website”. The user enters his “User Name” and “Password”. The server will check if they are valid.

If the information is correct, the server allows TempSens.exe in the user’s local PC to upload the temperature measurement data. These data will be stored in a database, **TempRemote.mdb**, on the server, in the user’s folder (named using the Product Serial No.)

TempRemote.mdb – inside folder “TS123003”

Date of Upload (from server)	Date (from local PC)	Time (from local PC)	Temperature
2003-12-21	2003-12-21	10:00	37
2003-12-21	2003-12-21	10:01	37.2
2003-12-21	2003-12-21	10:02	37.4
2003-12-21	2003-12-21	10:03	37.2
2003-12-21	2003-12-21	10:04	37

When TempSens.exe starts its uploading session, it will upload a temperature record, every minute. This event is triggered when the program TempSens.exe receives a data (string) from the serial port.

The internet connection may be lost for some time. TempSens.exe would not be able to upload the data. When the connection is established again, TempSens.exe will update the server database with the past temperature readings.

11.5 User Login at remote PC

The user can access his temperature measurement data from a remote PC, using a web browser. In fact, he can also access it from the local PC itself, for testing purposes.

At the website, http://www.TempSens_website.com, the user will login with his “User Name” and “Password”. When accepted, the server will extract the temperature data from the user’s database, TempRemote.mdb, on the server, in the user’s folder (named using the Product Serial No.)

The server will use these readings to plot a graph, for the user to see on his remote PC. The graph for the current day is shown. The vertical axis is temperature, and the horizontal axis is time. There will be an option for the user to zoom in on selected regions of the graph. The user can also see the raw temperature readings as text.

11.6 Server Temperature Databases

The server will only store up to the past 7 days of temperature data. The 7 days are calculated according to the variable “Date of Upload”, in case the User changes his date setting on his local PC. After 7 days, the temperature data is erased. This is to conserve space on the server.

11.7 Difficulty Faced

It was intended to outsource this portion of the project. Programmers and web designers were approached to handle the web management. The cost of setting up this website would amount to \$6,000. This is quite a large sum of money. This is certainly not appropriate for a project prototype. As a result, I decided to do a simple real-time remote temperature monitoring system, over the internet. This will be explained in the following chapter.

Chapter 12 Monitoring over the Internet

12.1 Excel Visual Basic

As it was not economically feasible to accomplish the web management proposal mentioned in the previous chapter, a simpler version was developed. It is described in this chapter. The source codes are placed in Appendix B.

An Excel 2002 file, “TempSens.xls”, was created for the purpose of updating the FTP server. Excel Visual Basic (VB) was used as the programming language. The following diagram shows the Visual Basic editing environment.

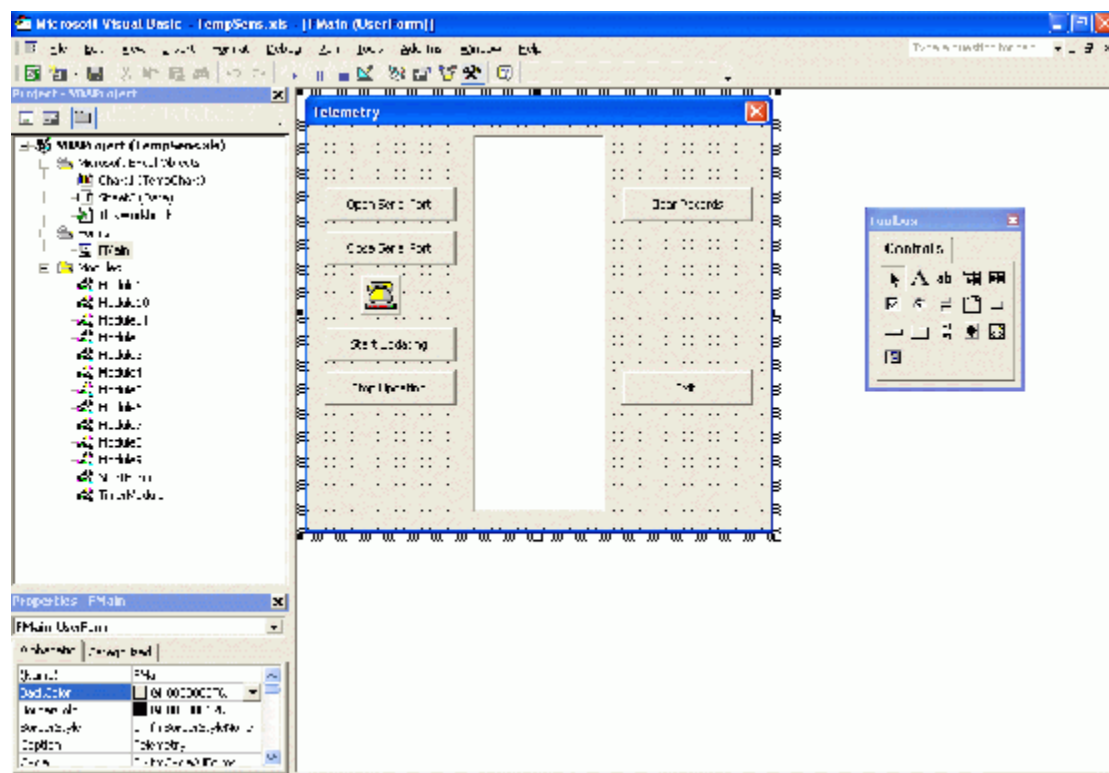


Figure 16: Designing the Form in Excel Visual Basic

The short-cut key “Ctrl+T” was assigned to run the “Telemetry” Visual Basic Macro. After opening the files “TempLog.xls” and “TempSens.xls”, and saving the chart to the FTP server, the connection to the FTP server is established. The connection will remain, as long as the Excel program is kept open. “Ctrl+T” can be pressed to activate the “Telemetry” Macro. On the form that appears, the user can choose “Open Serial Port” and “Start Updating”. This will allow the FTP server to be updated in regular intervals. The update interval can be set by the user, by typing in the appropriate cell (“Update Interval”). The user can choose “Clear Records” to erase the past temperature logs.

The Parent’s Unit is to be connected to the PC via the serial port. When the Parent’s Unit receives a temperature value from the Child’s Unit, it will send this data to the PC. The running “TempSens.xls” would capture the data. The temperature logs will be copied to the file “TempLog.xls”. “TempLog.xls” would be uploaded to the FTP server. The chart in “TempSens.xls” would also be published to the server. Following that, the file “TempSens.xls” would be saved to the local hard drive.

This process is repeated after the duration of the “Update Interval”. The “Update Interval” is the period of free time between successive updates. The repetition period may be longer, due to a slow internet connection, or a slow processor speed.

The following figure shows the “TempSens.xls” Visual Basic program in operation. As can be seen from the figure, the program will also keep a record of the instances when the Child’s Unit is not detected, or when the Parent’s Unit is not detected. A file “Report.txt”, is also created, which records any errors encountered.

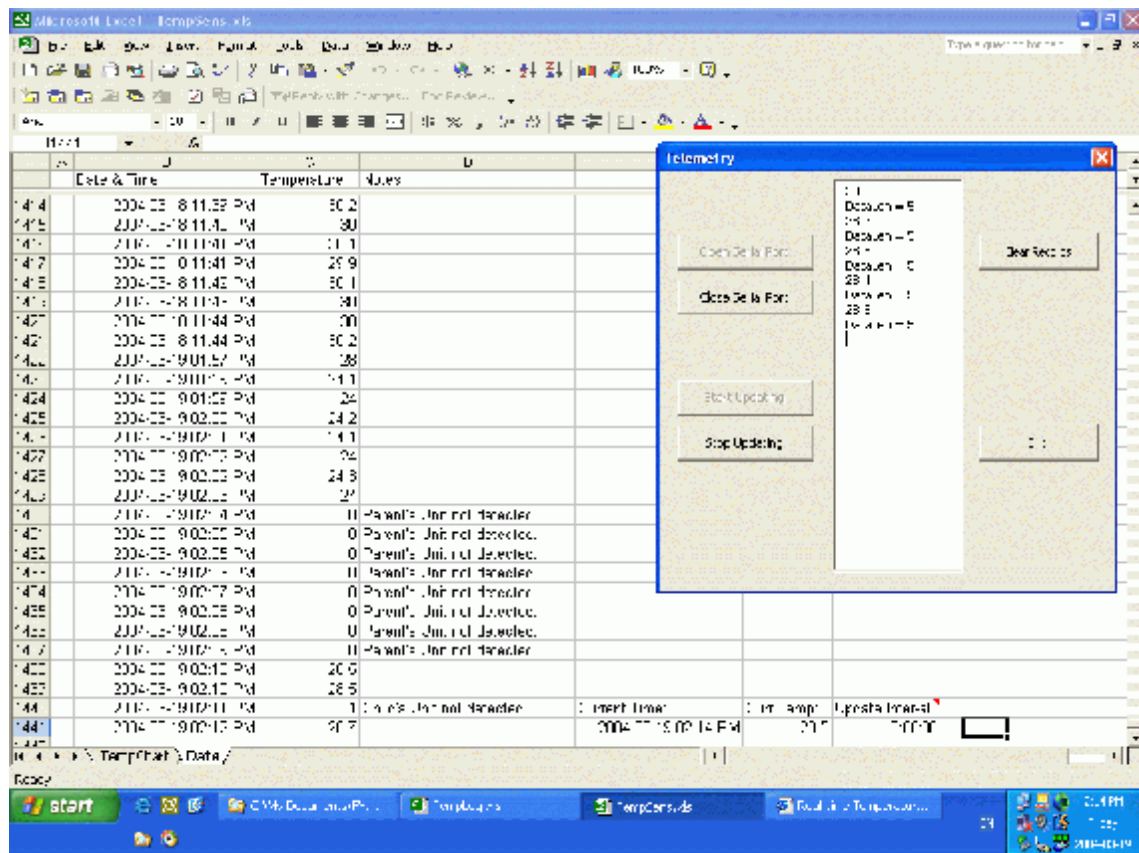


Figure 17: “TempSens.xls” Program in Operation

12.2 Viewing from a Web Browser

Once the program in “TempSens.xls” is running in the home PC, the parents will be able to see their child’s temperature data and graph over the internet, anytime, and anywhere. The website address currently used for this project is

<http://www.innovation-invention.com/Telemetry/>.

Once at the website, the user can click on the link, “View the Real-time Temperature Graph”. The temperature graph will be shown, and will refresh itself automatically, every 15 seconds. It is best viewed with a 1024 by 768 screen resolution. The graph is automatically scaled to fit all the temperature data. The user can also download the file “TempLog.xls”, which contains the temperature data.

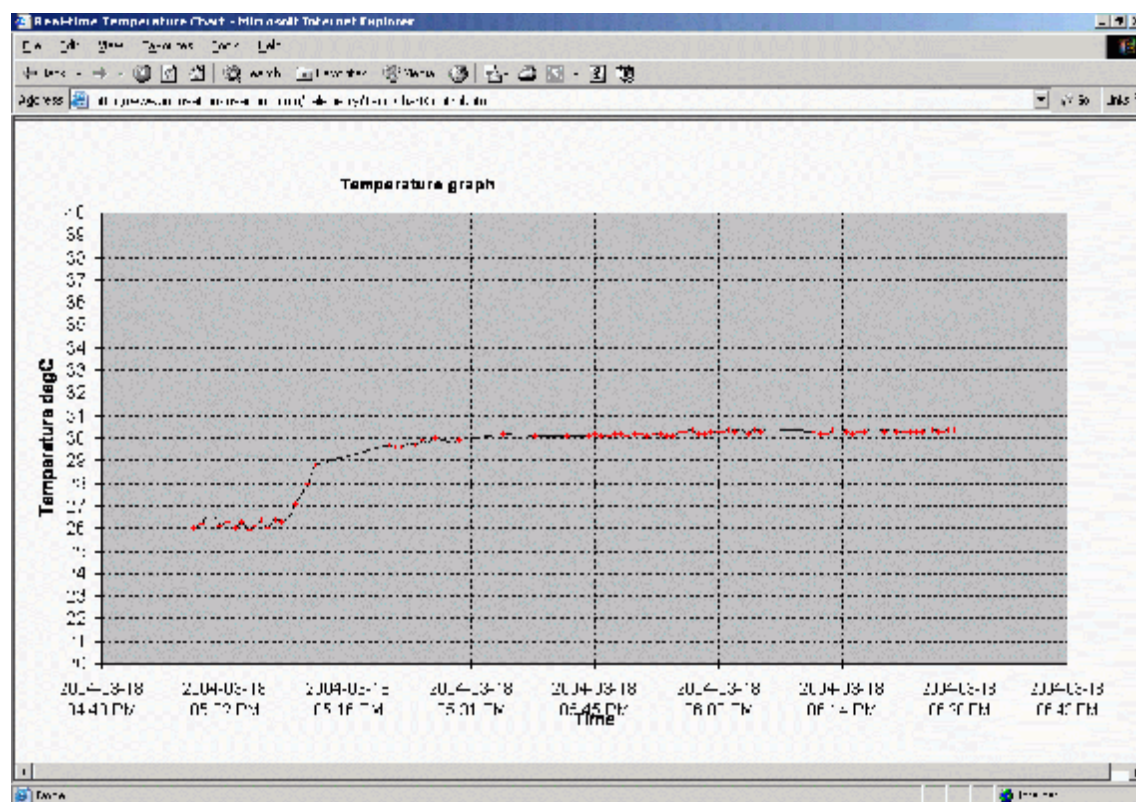


Figure 18: Real-time Temperature Graph

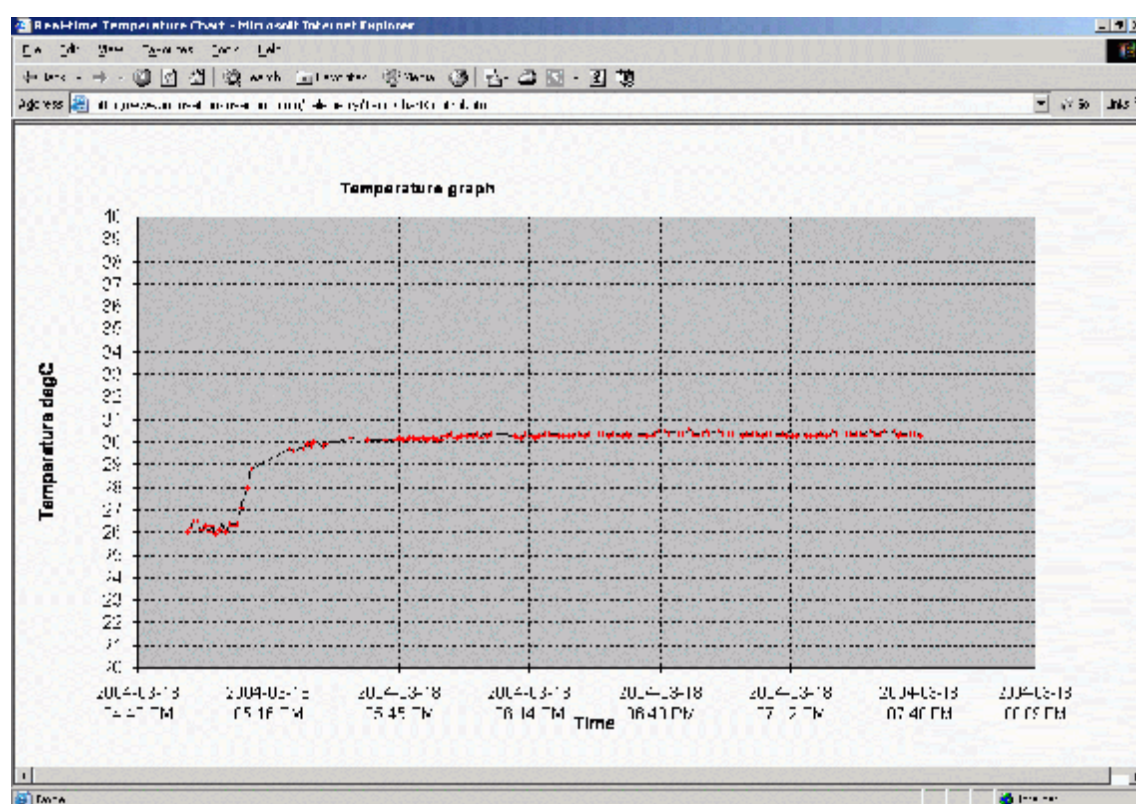


Figure 19: Graph Scales Automatically To Fit Data

The source code for the web page showing the real-time graph is included in Appendix C. It can be seen that the code can be very short and simple. The absolute internet address must be stated, instead of just the relative link. This is to prevent Internet Explorer from showing its cached version of the webpage, which will remain static.

12.3 Troubleshooting TimerModule

The Excel VB command “Application.OnTime” allows the user to schedule a task at a future date. This command is used repeatedly, to schedule the next updating process, once the current process has finished executing.

The repetition time had to be equal or longer than the “Update Interval”. However, it was found that the updating process, after a while, would repeat itself immediately after the current process.

The problem was traced to the control buttons “Stop Updating” and “Start Updating”. On pressing “Stop Updating”, the updating process is stopped by setting a certain flag variable, to halt the next updating process that is scheduled to come. It can only halt the updating at the future scheduled time, not at the current time. By clicking on the “Start Updating” button, a new timer process is initiated, and the flag variable is set to allow future scheduled events. Thus two timer processes operate at the same time, giving continuous updating. To solve this problem, the program keeps track of the number of timer processes. If there are more than 1 timer processes, the program will stop the redundant timer process.

Another problem was encountered. After around half an hour to a few hours, the program could no longer upload to the server. Subsequently, the program reported errors at every scheduled updating process. The “Publish” property was changed to “True”, so that Excel would upload a new file each time, instead of just uploading the additional data. The problem could have been due to a corruption in the data, while it was sent to the server. The uploading commands were modified to include the absolute pathname and filename, instead of just specifying a “Save” at the same previous location. In the event the program fails to upload, the internet connection can be closed and reopened. The program will then continue upload.

Screen savers were always suspected as the cause of certain problems. On the contrary, the problems were found to have different reasons behind them. Thus, screen savers, so far, are not known to affect the program’s operation.

12.4 Troubleshooting FMain

Every now and then, the program encountered an error. When the data is received from the serial port, it will be written to one particular cell in the worksheet. This worked most of the time, but after a few minutes, the program hanged. The cause of the problem was that at the time the serial data was received, Excel was busy saving the file. When Excel is saving the file, data cannot be written to the cells. As a solution, Boolean flag variables were used to indicate that Excel is busy saving, or busy modifying. This ensured that the two processes did not occur simultaneously.

Chapter 13 Assembly

13.1 Mock-up Boards

Due to time constraints, different aspects of the project, like hardware and software, had to be done simultaneously. Boxes were required to house the Parent's Unit and the Child's Unit. The PCB design was just beginning. Boxes of the correct dimensions and shape needed to be selected. Furthermore, machining of the boxes had to be done to incorporate the final circuitry.

Looking at the breadboards consisting of the hardware setups, mock-up boards were created. Mock-up boards are basically full-scale models of the final hardware. A great deal of estimation had to be done, as to the positioning of the different components. Styrofoam boards and basic hardware components were used in the construction of the mock-up boards.

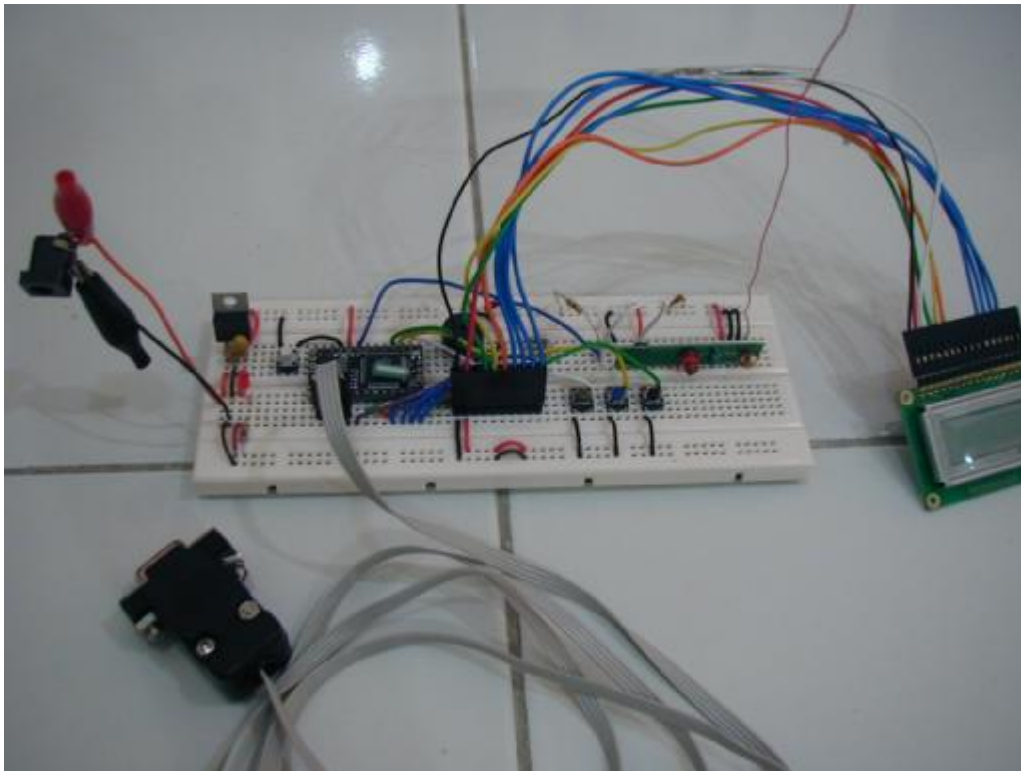


Figure 20: Breadboard Setup for Parent's Unit

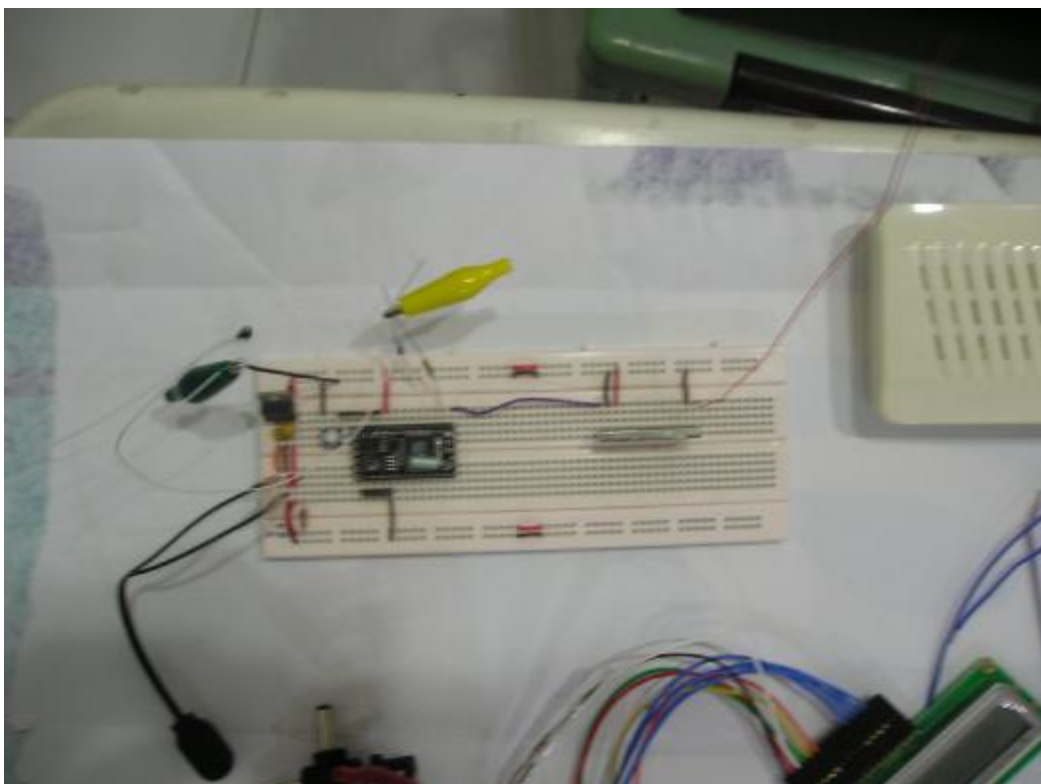


Figure 21: Breadboard Setup for Child's Unit

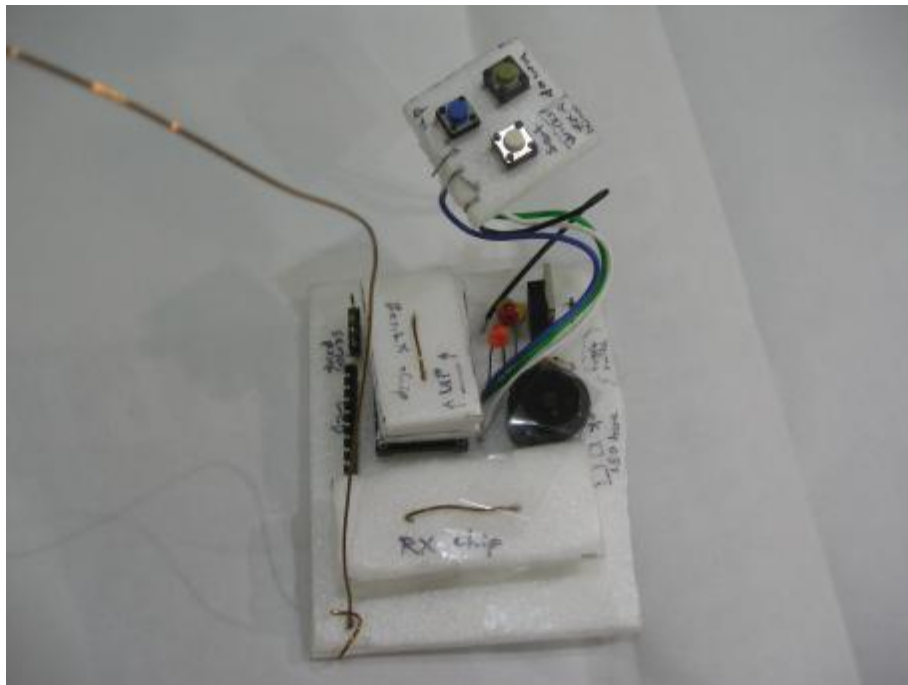


Figure 22: Mock-up Board for Parent's Unit (without LCD)

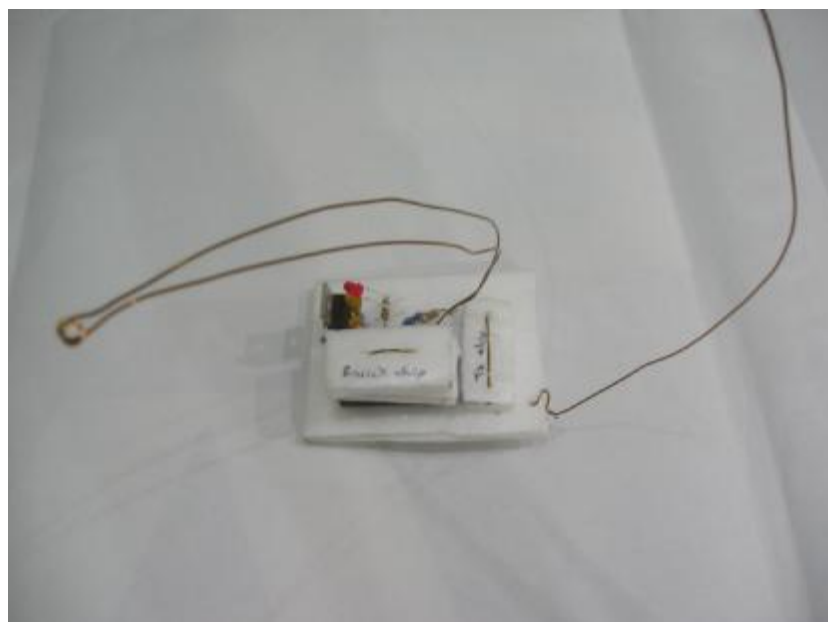


Figure 23: Mock-up Board for Child's Unit

13.2 Selecting the Boxes

Places like Sim Lim Tower and Mustafa Centre (Serangoon Road) were visited.

With the mock-up boards in hand, different products on the shelves were inspected, to see which could be suitable to house the hardware setups. The size of the boxes is of extreme importance. The box should not be too large. On the other hand, too small a box will result in the PCB being unable to fit inside. If the box is too small, one option would be to design a double-sided PCB. This is not so easy, and the NUS PCB Fabrication Lab does not recommend double-sided PCB's.

13.3 Machining the Boxes

According to the models of the mock-up boards, the boxes were marked for machining.



Figure 24: Preparing the Parent's Unit for Machining

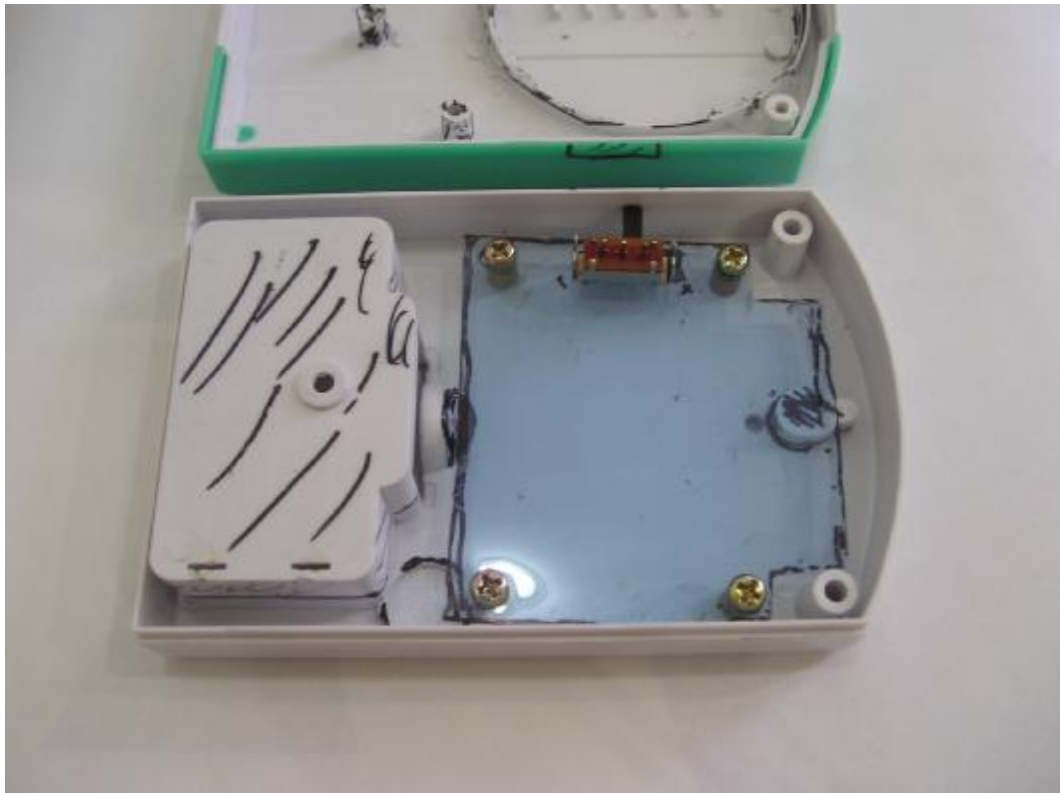


Figure 25: Preparing the Child's Unit for Machining

13.4 PCB Design

According to the mock-up boards, and the box dimensions, Printed Circuit Boards (PCB) were designed for the various components in the Parent's Unit and the Child's Unit. The software used was "Protel Design System Advanced PCB version 2.7.1". It was indeed an advantage having had practice on this software in an earlier module, EE2001.

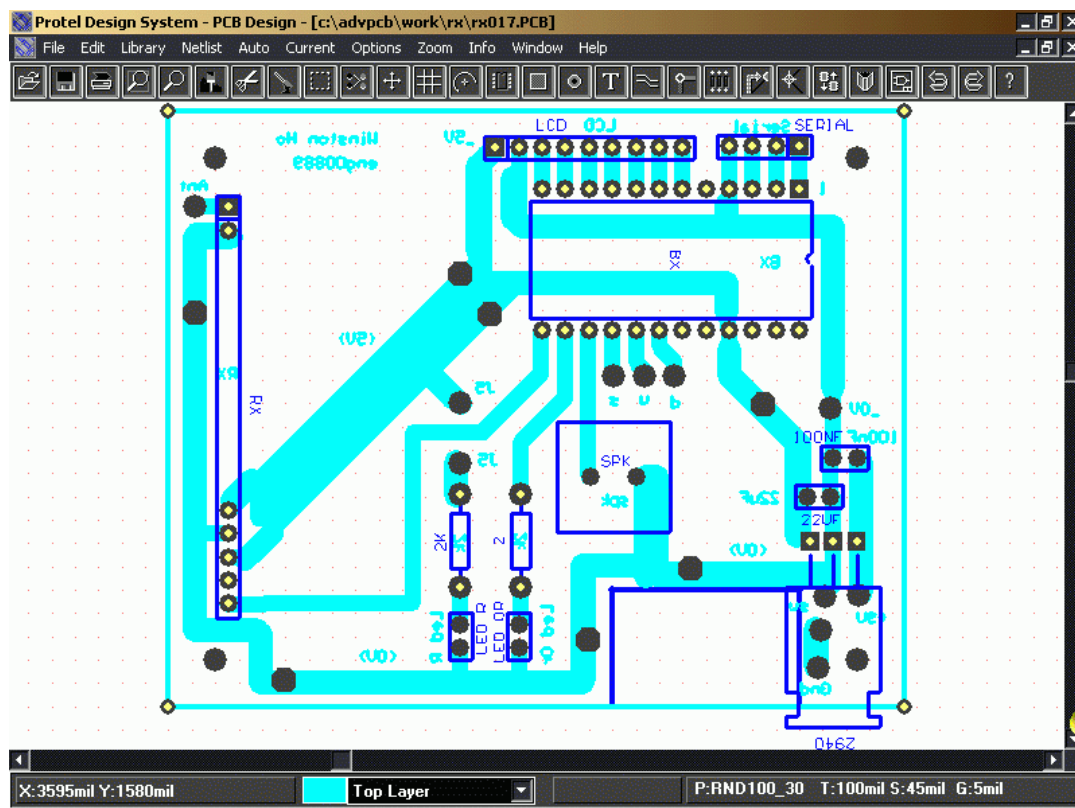


Figure 26: PCB for Parent's Unit

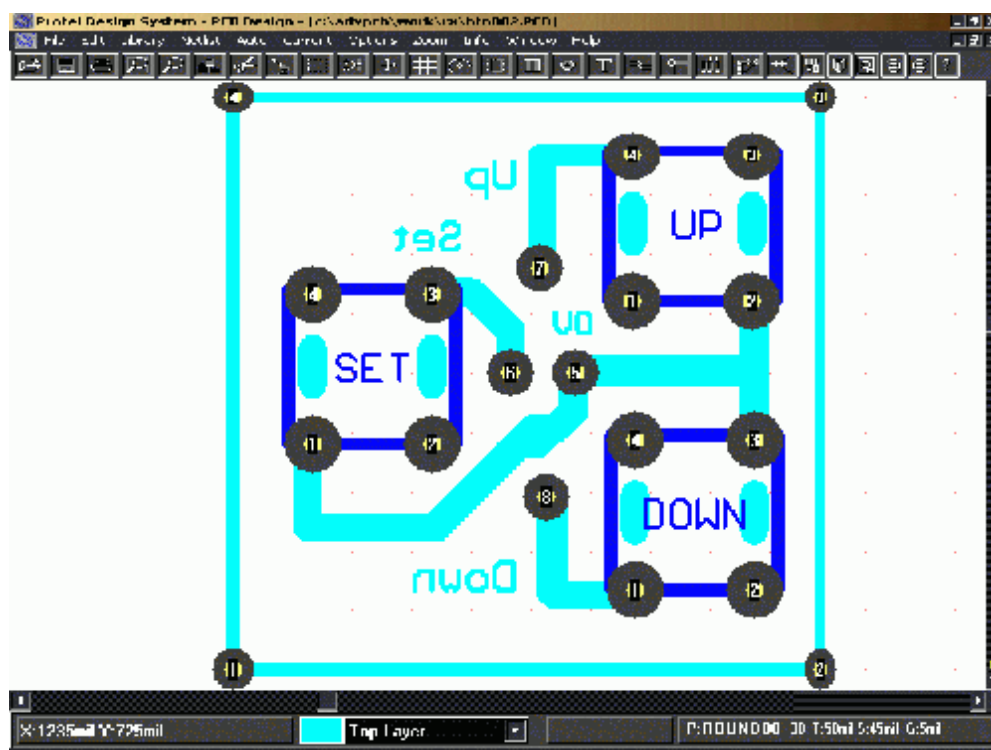


Figure 27: PCB for Parent's Unit Button Board

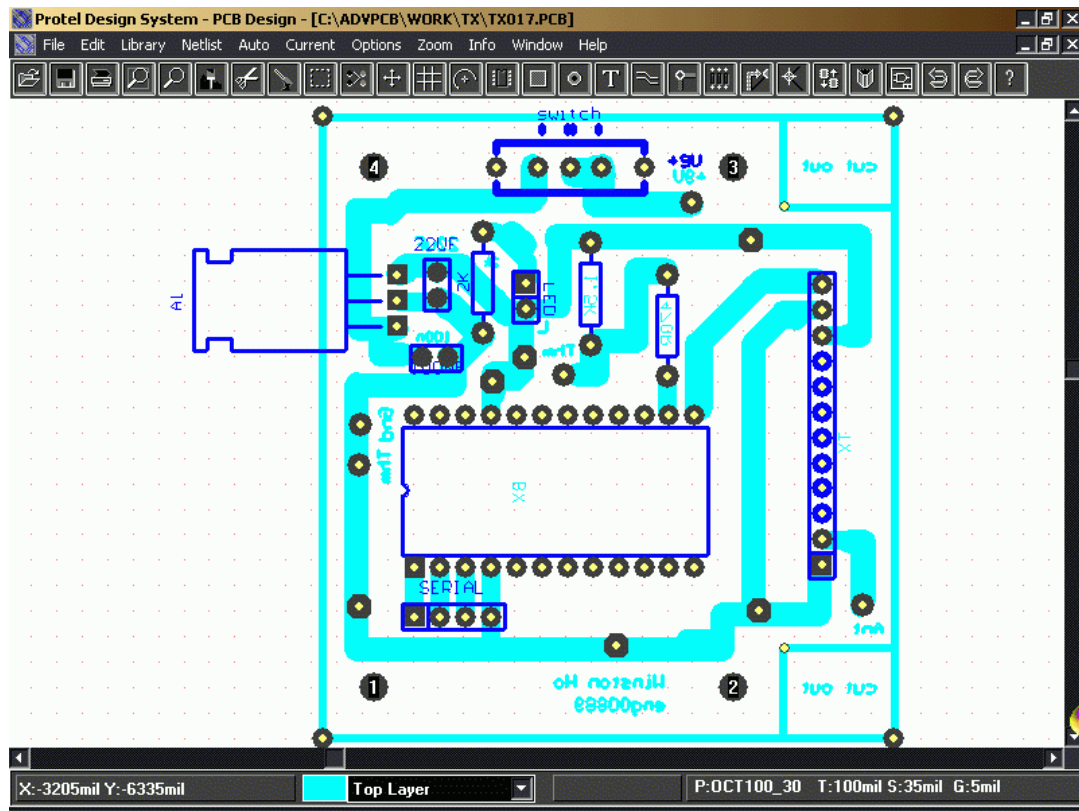


Figure 28: PCB for Child's Unit

13.5 Troubleshooting

The PCB's were fabricated by Jalil in the PCB Fabrication Lab in NUS. After obtaining the PCB's, the devices were soldered on. The PCB's were tested and they were found to be working. Jalil was quite impressed that they worked the first round.

However, it was found that the Child's Unit PCB was not able to operate beyond around 5 minutes. The voltage regulator became extremely hot. The power had to be cut off. There was even a faint burning smell. The voltage regulator was removed from the PCB. There had been initial trouble soldering on this device. Due to extended period of active soldering, the thermal stress on the voltage regulator may

have been too great. Upon removing the voltage regulator, one of the legs were broken from the chip's body. Still, it could not be confirmed that the fault lay in the voltage regulator.

Every device on the PCB was tested thoroughly, using a multimeter. Since they have all been soldered onto the PCB, and electrically connected, estimation had to be done, to account for the effect of the circuit.

It was found that one of the capacitors did not show the correct capacitance. On closer observation, the polarity of the capacitor (22uF) was actually reversed. It had been assumed that capacitors of this particular shape and design did not have a polarity. All these while, during the testing on the breadboards, this capacitor had somehow been placed correctly. The capacitor was promptly replaced.

13.6 Final Assembly

The following diagrams show the final assemblies of the Child's Unit and the Parent's Unit. Both the internal and external views are shown.



Figure 29: Inside the Child's Unit



Figure 30: Inside the Parent's Unit

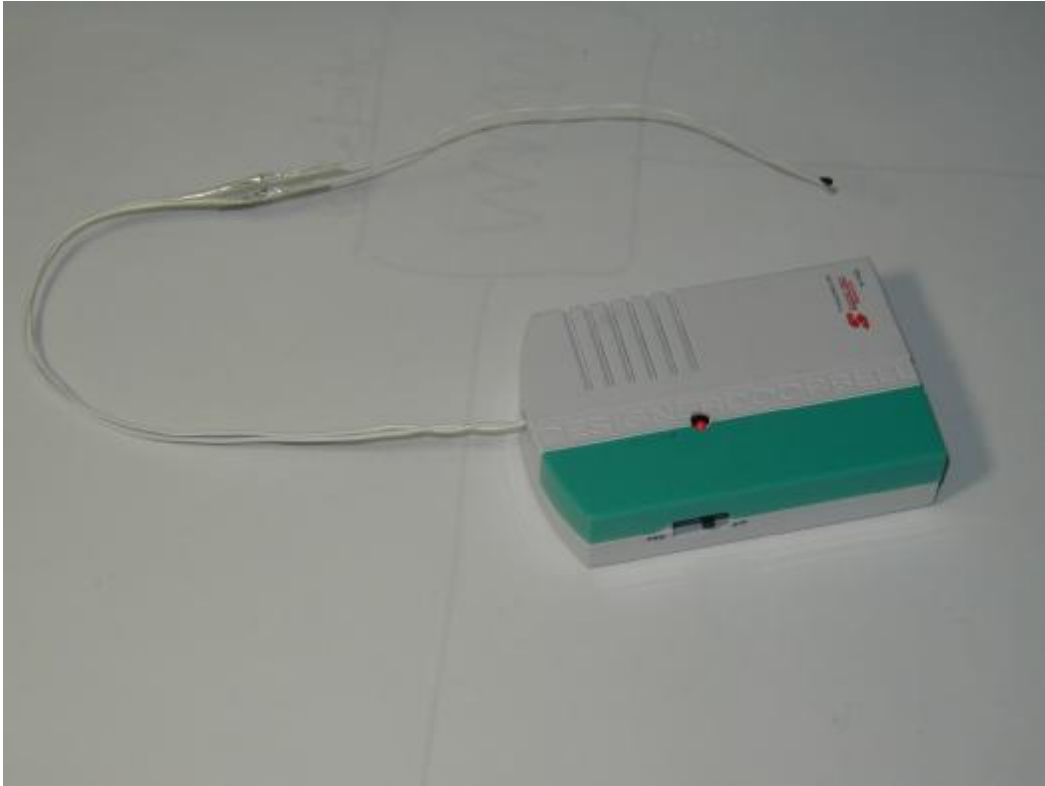


Figure 31: Child's Unit



Figure 32: Parent's Unit

Chapter 14 Conclusions

To solve the problem of febrile fits in children, the “Febrile Fits Monitor” was developed in this project. Although there were technical difficulties encountered during the progress of the project, they were overcome eventually, through detailed troubleshooting and careful planning. This project involved a great deal of integration between hardware and software. These types of projects are not very common in the usual curriculum. This being the case, I took it as a challenge to develop something different, and novel.

When this project was conceived, and got under way, there were no other devices like this found in the market. In the duration of the project, however, a certain similar device was made available commercially. It was “Bebe Sounds”. This also consisted of two units. One unit would be on the child, and would also send the temperature data to the other unit. However, the temperature sensor was flat on the child’s unit, and had to be positioned at the tummy. The temperature readings would tend to be less than actual body temperature.

I had to improve my system, so as not to be simply performing the same function. Firstly, I have an external thermistor probe that can be positioned at the armpit. Secondly, the system was improved to allow the user to view the temperature chart over the internet.

This project does not need to stop here. The system could be improved to have added functionality. For example, the system could also measure other biomedical signals, like heart rate, blood pressure and oxygen saturation. The framework of hardware, communications and software can easily be adapted to suit the additional features.

It must be said that tremendous experience was gained during the development of the “Febrile Fits Monitor”. With careful time management and diligence, the project was brought from its infancy of conceptualization, to a complete and practical working model.

Appendix A Source Codes for Microcontrollers

A.1 Code for Transmitter

```
' TxMain.bas

'Remember to set the compiler option to get 32 character string.

Option Explicit

'Note : Queue buffer overhead = 9 bytes
Private Const InputBufferSize As Integer = 30
Private InputBuffer(1 To InputBufferSize) As Byte

Private Const OutputBufferSize As Integer = 30
Private OutputBuffer(1 To OutputBufferSize) As Byte

'Private Const PCInputBufferSize As Integer = 30
'Private PCInputBuffer(1 To PCInputBufferSize) As Byte
'Private Const PCOutputBufferSize As Integer = 30
' ' If = 10, there's no transfer
'Private PCOutputBuffer(1 To PCOutputBufferSize) As Byte
'Private Const Buffer30Size As Integer = 30
'Private Buffer30(1 To Buffer30Size) As Byte
'Private Queue2(1 To 30) As Byte
'Private Queue3(1 To 30) As Byte
'-----
Dim HeaderStr As String
Dim ByteArr7(1 To 10) As Byte
'Dim ByteArrReady(1 To 10) As Byte
Private Const TXpin As Byte = 13
'Private Const TXpin2 As Byte = 14 'false TX pin
'Private Const Com3Baud As Integer = 300
' ' baud rate for FM TX
' ' Valid range 300 to 19200 baud.
Private Const IdleTime As Single = 0.35 'seconds
' ' Idle time of 10+3 bytes duration.
' ' For UART serial interface timing purposes.
' ' To prevent byte framing error.
'-----
Sub Main()
'Dim RFData As Byte
'dim Temp1620 as single
'dim Temp as byte
'Dim Temp2 As Byte

Dim NonDimVolt As Single
Const ADCPin As Byte = 14
Dim V_T As Single 'Votage across thermistor
Const R1 As Single = 1500.0 'ohms
Dim R_T As Single
Dim T_T As Single
'Steinhart Coefficients for 3K3A:
Const A_T As Single = 1.405027E-3
Const B_T As Single = 2.369386E-4
Const C_T As Single = 1.012660E-7
'Dim T_str As String

'debug.print "Ready"
'Call OpenPCchannel
```

```

Call OpenRFchannel
Call DefineTrueTX
'Call OpenQueue(Buffer30, Buffer30Size)
'Call OpenQueue(Queue2, 30)
'Init the DS1620
'config_1620
'start_convert
'RFData = 0      ' not used

HeaderStr = "Start" ' Header String '5 char
Call InitHeader

Do
    'Temp1620 = Read1620
    'debug.print "DS1620: "; cstr(Temp1620)
    'Temp2 = CByte(Temperature*2.0)

    ' Temp2 = CByte(Temperature*2)
    ' gives an error "Type Mismatch"
    ' 2 is Integer
    ' 2.0 is Single
    'Call PutQueue(OutputBuffer, Temp2, 1)' send data
    'Call PutQueue(OutputBuffer, 123, 1)' send data
    ' Call PutQueue(OutputBuffer, 123+Temp, 1)' send data

    Call GetADC (ADCPin , NonDimVolt)
    V_T = NonDimVolt * 5.0 'Actual voltage in volts

    R_T = (V_T * R1) / (5.0 - V_T)
        'Resistance of thermistor

    T_T = 1.0 / _
        (A_T + B_T*Log(R_T) + C_T*Pow(Log(R_T),3.0)) _
        - 273.15
        'Temperature of thermistor in deg Celcius

    T_T = CSng(CInt(T_T*10.0)) * 1.0 / 10.0
        ' Round down to 1 decimal place

    debug.print "Thermistor: "; cstr(T_T)

    'T_str = CStr(T_T)
    'Call PutQueueStr(OutputBuffer, T_str)' send data, TX
    'debug.print'CR LF --- new line
    'T_T = CSng(CInt(T_T))      ' Testing
    ' Round down to 0 decimal place
    'Temp2 = CByte(T_T*2.0)
    'Call PutQueue(OutputBuffer, Temp2, 1)' send data
    'Pi = 3.14159
    'Call PutQueue(OutputBuffer, Pi, 4)
    'Call Tester
    'Call DefineFalseTX
    'Call BusyFM
    'Call DefineTrueTX
    Call SendData(T_T)

    delay 4.1      'effectively repeating every 5 seconds
Loop
End Sub
'-----
Sub OpenRFchannel()
' Open the RF communications channel.
' Open input and output queues.
Call OpenQueue(InputBuffer, InputBufferSize)
Call OpenQueue(OutputBuffer, OutputBufferSize)
' Configure Com3 to transmit on I/O pin TXpin.
' Not interested in receiving anything,
' so use pin 0 for the receive pin
' (pin 0 is a dummy pin).

End Sub
'-----
Sub DefineTrueTX()
Call DefineCom3(0, TXpin, bx1000_1000)
        ' inverted, no parity, 8 bits.
' Open Com3 at Com3Baud baud.
Call OpenCom(3, 300, InputBuffer, OutputBuffer)

```

```

    Call Sleep(0.5)
End Sub
'-----
Sub DefineFalseTX()
'    Call DefineCom3(0, TXpin2, bx1000_1000)
'    ' inverted, no parity, 8 bits.
'
'    ' Open Com3 at Com3Baud baud.
'    Call OpenCom(3, 300, InputBuffer, OutputBuffer)
'    Call Sleep(0.5)
End Sub
'-----
Sub BusyFM()
'    Dim Count As Integer
'    For Count = 1 To 12
'        Call PutPin(TXpin, bxOutputLow)
'        Call Sleep(0.1)
'        Call PutPin(TXpin, bxOutputHigh)
'    Next
End Sub
'-----
Sub OpenPCchannel()
'    ' Open the serial communications channel.
'    ' Open input and output queues.
'    Call OpenQueue(PCInputBuffer, PCInputBufferSize)
'    Call OpenQueue(PCOutputBuffer, PCOutputBufferSize)
'
'    ' No need to DefineCom3
'
'    Call OpenCom(1, 19200, PCInputBuffer, PCOutputBuffer)
'    Call Sleep(0.5)
End Sub
'-----

Sub SendData(T_T As Single)

    Dim FullStr As String
    Dim ByteArr1(1 To 15) As Byte
    Dim CurrCharStr As String * 1
    Dim Count As Byte

    FullStr = HeaderStr & CStr(T_T) & _
        " " '10 space
    Debug.Print "Check Point : SendData"
    '-----
    ' 5 (HeaderStr) + 10 (space) = 15
    For Count = 1 To 15
        CurrCharStr=Mid(FullStr,Count,1)
        'String expression cannot be
        ' used as function argument.
        ByteArr1(Count) = Asc(CurrCharStr)
        ' debug.print CurrCharStr ; " " ;
    Next
    '-----
    'Serial interface timing purpose:
    Call PutQueue(OutputBuffer, ByteArr7, 10)
    Delay IdleTime
    Call PutQueue(OutputBuffer, ByteArr7, 10)
    Delay IdleTime
    '-----
    'Finally send the header with data
    Call PutQueue(OutputBuffer, ByteArr1, 15)
    debug.print "Sent the data."
End Sub
'-----
Sub InitHeader()
'    Dim ReadyStr As String
'    Dim CurrCharStr As String * 1
'    Dim Count As Byte
'    ReadyStr = "abcdefghij"
'    '-----
'    'Serial interface timing purposes:
    For Count = 1 To 10
        ByteArr7(Count) = 55'Asc("7")
    Next
    'Str7 = "7777777777" '10 7's
    '-----

```

```

End Sub
'-----
'Sub Tester()
'
'   'Dim Pi As Single
'   Dim TestAsc As Byte
'   Dim TestChr As String
'
'   TestAsc = Asc("7")
'   debug.print CStr(TestAsc)
'
'   TestChr = Chr(255)
'   debug.print TestChr
'
'End Sub
'-----

```

A.2 Code for Receiver

```

' RxMain.bas
' Pin 13 receives data from the Receiver chip.
' Pin 15 is connected to the buzzer.
' Pins 6 to 12 are used for the LCD screen.
' Pins 16 to 18 are attached to PushButtons 1 to 3
' Pin 14 is connected to LED for Fever indication.

Option Explicit
'Note: Queue buffers have implicit 9 bytes overhead.
Private Const InputBufferSize As Integer = 30
Private InputBuffer(1 To InputBufferSize) As Byte

Private Const OutputBufferSize As Integer = 30
Private OutputBuffer(1 To OutputBufferSize) As Byte

Private Const PCInputBufferSize As Integer = 30
Private PCInputBuffer(1 To PCInputBufferSize) As Byte

Private Const PCOutputBufferSize As Integer = 30
' If = 10, there's no transfer
Private PCOutputBuffer(1 To PCOutputBufferSize) As Byte

Private Const SpeakerPin As Byte = 15
Private Const RXpin As Byte = 13 'Receive data from RX chip
Private Const PB1Set As Byte = 16 'Set Button
Private Const PB2Up As Byte = 17 'Up button
Private Const PB3Down As Byte = 18 'Down button
'Private Const PB4Clear As Byte = 17 'Clear button
'acknowledge alarm
Private Const LightPin As Byte = 14 'LED pin

'Dim RFData As Byte
'Dim FeverPoint As Single = 37.5 'default 37.5 deg C
'Cannot write the value here.
'Only write the value for Const.
Dim FeverPoint As Single 'Fever Threshold

Dim DumpByte As Byte
'Dim StatusQueueDelay As Single
Dim TimeLastRX As Single
Dim WaitInterval As Single
Dim TimeToCheck As Single
Dim TimeNow As Single
Dim CUMissing As Boolean
Dim TimeLastPC1 As Single
Dim ResendInterval As Single

'-----
Public Sub Main()

    Dim T_Str As String * 5
    Dim CurrByte As Byte
    Dim HeaderStr As String * 5
    Dim RecogChar As String * 1
    Dim MatchIndex As Byte

```

```

'Dim Match7 As Integer
Dim Count As Byte

Dim T_Index As Byte
Dim T_T As Single
Dim T_Success As Boolean

PutPin PB1Set , bxInputPullup
'Input with BasicX's internal 120k pullup resistor
PutPin PB2Up , bxInputPullup
PutPin PB3Down , bxInputPullup
'PutPin PB4Clear , bxInputPullup

FeverPoint = 37.5 'default 37.5 deg C

LCDInit
delay 0.1
LCDClearDisplay

Call OpenPCchannel 'IMPORTANT !!!
Call OpenRFchannel

HeaderStr = "Start"
'StatusQueueDelay = 0.05
'0.03 possible if:
'    Debug.Print Chr(CurrByte)
'    Without this statement,
'    Does not work all the time
'0.02 cannot work
'-----
T_Str = " " 'initialize string.
RecogChar = " "
'-----
TimeLastRX = Timer
WaitInterval = 15.0 'Wait 15 seconds for the Child's Unit ###
CUMissing = False
ResendInterval = 5.0'Wait 5 s before resending 1 to PC

'LCDClearDisplay
LCDMoveCursorPos 1,1
LCDPrint "Temp : " 'spaces to erase previous value.

Do
    'Debug.Print "MatchIndex at start = " ; CStr(MatchIndex)
    MatchIndex = 1
    'Match7 = 0
    'If data is present in the serial queue, get it
    'Do While StatusQueue(InputBuffer)
    'There's always data coming in anyway,
    'due to RX chip giving nonsense because
    'of Automatic Gain Control

    Do While True

        Call GetQueue(InputBuffer, CurrByte, 1)

        'Debug.Print Chr(CurrByte) 'Test
        If CurrByte = 55 Then 'Asc("7") = 55
            Match7 = Match7 + 1 'Byte "7" detected
            'Debug.Print "7 is detected."
        Else
            Match7 = 0
        End If

        If Match7 >= 5 Then ' 5 7's detected
            For Count = 1 To 25
                Call GetQueue(InputBuffer, DumpByte, 1)
            Next
            'Match7 = 0
            Exit Do
        End If

        RecogChar = Mid(HeaderStr,MatchIndex,1)
        If CurrByte = Asc(RecogChar) Then
            MatchIndex = MatchIndex + 1 'Match found
            'Debug.Print "MatchIndex = " ; CStr(MatchIndex)
        Else

```

```

        MatchIndex = 1
    End If

    If MatchIndex >= 6 Then      '"Start" received
        'Debug.Print "Header Recognized."
        'MatchIndex = 1 'original
        Exit Do
    End If

    'Delay StatusQueueDelay
    CheckButton
    CheckTime

Loop

'If (MatchIndex >= 6) Then
' --- Definitely True at this point
' Debug.Print "Check Point : Header detected"

For T_Index = 1 To 5 '100.0 for temperature
    Call GetQueue(InputBuffer, CurrByte, 1)
    'Debug.Print Chr(CurrByte) 'Test
    Mid(T_Str,T_Index,1) = Chr(CurrByte)
Next

    'freqout SpeakerPin,4000,4000,100
    'Will result in temperature "2!" ?
    'Now no more. Nevermind.
    'Delay StatusQueueDelay
'End If

'Delay 1.0 ' - even worse. T_Success always false.
'For Count = 1 To 21 '30-9
'    Call GetQueue(InputBuffer, DumpByte, 1)
'    'Remove from the queue and do nothing
'Next
'Debug.Print T_Str
'Forgot to initialize the string!!!
'Mid function only modifies one section.
'The rest of the string is nonsense.

Call ValueS(T_Str, T_T, T_Success)
'T_T = CSng(T_Str) cannot work.

If T_Success = True Then
    'Debug.Print "Success = True"
    'Debug.Print CStr(T_T)
    Call Action(T_T)
Else
    'Debug.Print "Success = False"
End If

CheckButton
CheckTime

Loop

End Sub

'-----
'Sub BufferClear7()
'End Sub
'-----

Sub CheckButton()

    ' check if button is pressed
    If GetPin(PB1Set) = 0 Then
        Delay 0.005 'bounce settle down
        If GetPin(PB1Set) = 0 Then 'button down

            'Wait for it to go up.
            Do
                ClearInputBuffer
                'So that it won't rush through
                'all the accumulated values in queue.
            Loop Until GetPin(PB1Set) = 1

            FreqOut SpeakerPin,4000,4000,200

```

```

        FreqOut SpeakerPin,8000,8000,200
        SetFeverPoint

    End If
End If

End Sub
'-----
Sub CheckTime()

    If CUMissing = True Then
        TimeNow = Timer
        TimeToCheck = TimeLastPC1 + ResendInterval
        'cross midnight
        If (TimeToCheck - TimeNow) > (ResendInterval + 1.0) Then
            TimeToCheck = TimeToCheck - 86400.0
        End If

        If TimeNow > TimeToCheck Then
            'Send 1 to PC every 5 seconds.
            SendToPC(1.0)
            TimeLastPC1 = Timer
            CUMissingBeep
        End If

        Exit Sub
    End If

    TimeNow = Timer
    TimeToCheck = TimeLastRX + WaitInterval
    'cross midnight
    If (TimeToCheck - TimeNow) > (WaitInterval + 1.0) Then
        TimeToCheck = TimeToCheck - 86400.0
    End If

    If TimeNow > TimeToCheck Then

        LCDClearDisplay
        LCDMoveCursorPos 1,1
        LCDPrint "Child's Unit"
        LCDMoveCursorPos 2,1
        LCDPrint "not detected"

        SendToPC(1.0)
        TimeLastPC1 = Timer
        CUMissingBeep
        CUMissing = True

    End If

End Sub
'-----
Sub SendToPC(ByVal T_T As Single)

    Dim T_Str2 As String * 5

    T_Str2 = CStr(T_T)
    Call PutQueueStr(PCOutputBuffer, T_Str2)' send data to PC
    Delay 1.0

End Sub
'-----
Sub CUMissingBeep()

    'FreqOut SpeakerPin,8000,8000,100
    'Delay 0.2
    FreqOut SpeakerPin,4000,4000,10

End Sub
'-----
Sub ClearInputBuffer()

    If StatusQueue(InputBuffer) Then
        Call GetQueue(InputBuffer, DumpByte, 1)
        'Remove from the queue and do nothing
    End If

End Sub

```

```

End If

End Sub
'-----
Sub SetFeverPoint()

    LCDClearDisplay
    LCDMoveCursorPos 1,1
    LCDPrint "Fever Setting:"
    DisplayFeverSet

    Do
        ' check if button is pressed
        If GetPin(PB2Up) = 0 Then 'Up button
            Delay 0.005 'bounce settle down
            If GetPin(PB2Up) = 0 Then 'button down

                'Wait for it to go up.
                Do
                    ClearInputBuffer
                    'So that it won't rush through
                    'all the accumulated values in queue.
                Loop Until GetPin(PB2Up) = 1

                FeverPoint = FeverPoint + 0.5
                DisplayFeverSet
            End If

        ElseIf GetPin(PB3Down) = 0 Then 'Down Button
            Delay 0.005 'bounce settle down
            If GetPin(PB3Down) = 0 Then 'button down

                'Wait for it to go up.
                Do
                    ClearInputBuffer
                    'So that it won't rush through
                    'all the accumulated values in queue.
                Loop Until GetPin(PB3Down) = 1
                FeverPoint = FeverPoint - 0.5
                DisplayFeverSet
            End If

        ElseIf GetPin(PB1Set) = 0 Then 'Exit Setting
            Delay 0.005 'bounce settle down
            If GetPin(PB1Set) = 0 Then 'button down

                'Wait for it to go up.
                Do
                    ClearInputBuffer
                    'So that it won't rush through
                    'all the accumulated values in queue.
                Loop Until GetPin(PB1Set) = 1

                FreqOut SpeakerPin,8000,8000,200
                'Delay 0.2
                FreqOut SpeakerPin,4000,4000,200
                LCDClearDisplay
                LCDMoveCursorPos 1,1
                LCDPrint "Temp : "
                'spaces to erase previous value.
                TimeLastRX = Timer
                CUMissing = False 'to enable CheckTime
                Exit Sub
            End If
        End If
    Loop
End Sub
'-----
Sub DisplayFeverSet ()

    'Debug.Print "Fever Setting: "; 'to PC
    ' 'for same line
    'LCDMoveCursorPos 1,1

    'LCDPrint "Fever Setting:"
    'Debug.Print CStr(FeverPoint) 'Automatically new line.

```



```

LCDMoveCursorPos 2,1
LCDPrint CStr(FeverPoint)
LCDPrint "      " '5 more spaces to clear 111.1

End Sub
'-----
Sub Action (ByVal T_T As Single)

    Dim T_Str2 As String * 5
    Dim Count As Byte
    Dim Count2 As Byte

    T_Str2 = CStr(T_T)
    'display Temp
    'debug.print "Temp: "; 'to PC
    Call PutQueueStr(PCOutputBuffer, T_Str2) ' send data to P
    Delay 1.0
    'Debug.Print T_Str2

    LCDClearDisplay
    LCDMoveCursorPos 1,1
    LCDPrint "Temp :      " 'spaces to erase previous value.

    'Temp = CSng(RFData) / 2.0
    'RFData is twice the temperature.

    LCDMoveCursorPos 1,8
    LCDPrint T_Str2

    if T_T >= FeverPoint then 'threshold FeverPoint.
        LCDMoveCursorPos 2,1
        LCDPrint "Fever"

        For Count2 = 1 To 2
            For Count = 1 To 2
                Call PutPin(LightPin, bxOutputHigh)
                freqout SpeakerPin,8192,8192,50 '50ms
                Call PutPin(LightPin, bxOutputLow)
                Delay 0.050 ' 0.05s

                Call PutPin(LightPin, bxOutputHigh)
                freqout SpeakerPin,4096,4096,50 '50ms
                Call PutPin(LightPin, bxOutputLow)
                Delay 0.050 ' 0.05s
            Next
            Delay 0.6
            LCDMoveCursorPos 2,1
            LCDPrint "      " 'to erase
        Next

        LCDMoveCursorPos 2,1
        LCDPrint "Fever"
    end if

    'if RFData < 128 then 'integer value for temperature.
    '    debug.print cstr(RFData)
    '    LCDMoveCursorPos 1,8
    '    LCDPrint cstr(RFData)
    '    if RFData >= 31 then'threshold of 31 deg C.
    '        freqout SpeakerPin,4000,4000,100
    '    end if
    'else 'temperature is an integer + 0.5
    '    debug.print cstr(RFData-128);".5"
    '    LCDMoveCursorPos 1,8
    '    LCDPrint cstr(RFData-128)
    '    LCDPrint ".5"
    'end if

    TimeLastRX = Timer
    CUMissing = False

End Sub
'-----
Sub OpenRFchannel()
' Open the RF communications channel.
' Open input and output queues.
Call OpenQueue(InputBuffer, InputBufferSize)

```

```

Call OpenQueue(OutputBuffer, OutputBufferSize)
' BasicX uses Com3 serial port.
' Configure Com3 to receive on I/O pin RXpin.
' Not interested in transmitting anything,
' so use pin 0 for the transmit pin.
' (pin 0 is a dummy pin).
' PutPin RXpin , bxInputTristate      'Added this line
' PutPin RXpin , bxInputPullup
Call DefineCom3(RXpin, 0, bx1000_1000)
' inverted, no parity, 8 bits.
' Call DefineCom3(RXpin, 0, bx1010_0111)
' inverted, odd parity, 7 data bits.
' Open Com3 at 300 or 19200 baud.
Call OpenCom(3, 300, InputBuffer, OutputBuffer)
' Call OpenCom(3, 9600, InputBuffer, OutputBuffer)
Call Sleep(0.5)
End Sub
'-----
Sub OpenPCchannel()
' Open the serial communications channel.
' Open input and output queues.
Call OpenQueue(PCInputBuffer, PCInputBufferSize)
Call OpenQueue(PCOutputBuffer, PCOutputBufferSize)

' No need to DefineCom3
Call OpenCom(1, 19200, PCInputBuffer, PCOutputBuffer)
Call Sleep(0.5)
End Sub
'-----

```

A.3 Code for LCD Display

```

'LCDLib.bas
Option Explicit

public const rs as byte = 6
public const rw as byte = 7
public const en as byte = 8

public const d3 as byte = 12
public const d2 as byte = 11
public const d1 as byte = 10
public const d0 as byte = 9

Public Sub LCDPrint(ByVal Text as string)
dim TextLen as byte
dim i as byte
dim TempStr as string
'Print a string
TextLen = cbyte(len(text))
if TextLen > 0 then
    for i= 1 to TextLen
        TempStr = Mid(Text,i,1)
        LCD4bit Asc(TempStr)
    next
end if
End Sub

Public Sub LCDMoveCursorPos ( _
    ByVal Line as byte, ByVal Pos as byte)
'data : move cursor to specific line and pos
if (Line>0) and (Line<3) and (Pos>0) and (Pos<17) then
    if Line = 1 then
        LCDCmd (bx1000_0000 + Pos -1)
    end if
    if Line = 2 then
        LCDCmd (bx1000_0000 + Pos + &H3F)
    end if
end if
End Sub

Public Sub LCDMoveCursorRight()
'data : move cursor right
LCDCmd bx0001_0100

```

```

End Sub

Public Sub LCDMoveCursorLeft()
    'data : move cursor left
    LCDCmd bx0001_0000
End Sub

Public Sub LCDCursorBlinkOn()
    'data : Set Cursor On with blink
    LCDCmd bx0000_1111
End Sub

Public Sub LCDCursorBlinkOff()
    'data : Set Cursor On - no blink
    LCDCmd bx0000_1110
End Sub

Public Sub LCDCursorOff()
    'data : Set Cursor Off
    LCDCmd bx0000_1100
End Sub

Public Sub LCDCursorOn()
    'data : Set Cursor On
    LCDCmd bx0000_1110
End Sub

Public Sub LCDDisplayOn()
    'data : Set Display On
    LCDCmd bx0000_1100
End Sub

Public Sub LCDDisplayOff()
    'data : Set Display Off
    LCDCmd bx0000_1000
End Sub

Public Sub LCDClearDisplay()
    'data : clr screen
    LCDCmd &H01
End Sub

Public Sub PulseX()
    putpin en,1
    putpin en,0
    delay 0.00001
End Sub

Public Sub LCDCmd(byval data as byte)
    dim tempH as byte
    dim tempL as byte
    putpin rs,0
    putpin rw,0
    delay 0.00001
    LCD4bit data
    putpin rs,1
    putpin rw,0
    delay 0.00001
End Sub

Public Sub LCD4bit(byval LCDdata as byte)
    dim data as byte
    dim i as byte
    data = LCDdata
    for i = 1 to 2
        putpin d3,(data and bx1000_0000)
        data = data * 2
        putpin d2,(data and bx1000_0000)
        data = data * 2
        putpin d1,(data and bx1000_0000)
        data = data * 2
        putpin d0,(data and bx1000_0000)
        data = data * 2
        PulseX
    next
End Sub

```

```

Public Sub LCDInit()
dim i as byte
for i = 1 to 2
    putpin en,0
    putpin rs,0
    putpin rw,0
    'data : H28
    LCD4bit &H28
    'data : H0C
    LCD4bit &H0C
    'data : H03
    LCD4bit &H03
    delay 0.001
    LCD4bit &H03
    delay 0.001
    'data : H01
    LCD4bit &H01
    delay 0.001
    'data : H06
    LCD4bit &H06
    'data : H02
    LCD4bit &H02
    delay 0.001
    putpin rs,1
    putpin rw,0
    delay 0.01
next
End Sub

```

A.4 Code for DS1620 Temperature Sensor

```

' DS1620lib.bas

Private Const DQ As Byte = 5 ' BasicX-24 pins
Private Const CLK As Byte = 6
Private Const RST As Byte = 7

Private o_byte As Byte
Private i_9_bit as integer

'=====
Function Read1620() as single
    measureT
    Read1620 = CSng(i_9_bit) * 0.5
    'i_9_bit is double the temperature
end Function
'=====
Sub config_1620()

    putpin RST,0
    putpin CLK,1
    putpin RST,1

    o_byte = &H0C ' Write Config [0Ch]
    out_byte

    o_byte = &H02
    '
    '          1          -- CPU Use Bit BasicX CPU
    '          0000 0010    is communicating to it
    '          0          -- 1SHOT = 0, so there is continuous
    '                      temperature conversion.
    out_byte
    putpin RST,0
    delay 0.2 ' Writing to E^2 req's 10ms at room temperature.

End Sub
'=====
Sub out_byte()
    dim n as byte
    dim temp as byte

```

```

for n=0 to 7' Send 8 bits into DS1620
    temp = o_byte and &H01 ' &H hexadecimal no. 0000 0001
    putpin DQ,temp ' temp always has a value of 0 or 1.
    putpin CLK,0
    'PutB(temp)
    'NewLine
    putpin CLK,1
    ' For data input to DS1620, data must be valid
    ' during the rising edge of the clock.
    o_byte = o_byte \ 2 ' All bits shift right.
next

End Sub
'-----
Sub start_convert()
    putpin CLK,1
    putpin RST,1
    o_byte = &HEE ' Start Convert [EEh]
    out_byte
    putpin RST,0

End Sub
'-----
Sub stop_convert()
    putpin CLK,1
    putpin RST,1
    o_byte = &H22 ' [22h]
    out_byte
    putpin RST,0

End Sub
'-----
Sub measureT()
    putpin CLK,1
    putpin RST,1' Initiate data transfer
    o_byte = &HAA ' Read Temperature [AAh]
    out_byte
    read_9_bits
    putpin RST,0' Terminate communications

End Sub
'-----
Sub read_9_bits()
    dim n as byte
    dim temp as byte

    i_9_bit = 0

    PutPin DQ , bxInputTristate
    'To receive data from DS1620 chip
    'Still keep the 1k resistor.

    for n=0 to 8
        ' The next 9 clock cycles will output the last
        ' temperature conversion result, LSbit first.
        putpin CLK,0' Data bits are output on the
        ' falling edge of the clock.
        temp = getpin(DQ)
        if temp = 1 then' if that bit = 1
            select case n
                case 0 ' bit position 0
                    ' LSbit (0.5 deg C)
                    i_9_bit = i_9_bit + 1
                case 1 ' bit position 1
                    i_9_bit = i_9_bit + 2
                case 2 ' bit position 2, etc
                    i_9_bit = i_9_bit + 4
                case 3
                    i_9_bit = i_9_bit + 8
                case 4
                    i_9_bit = i_9_bit + 16
                case 5
                    i_9_bit = i_9_bit + 32
                case 6
                    i_9_bit = i_9_bit + 64
                case 7
                    i_9_bit = i_9_bit + 128
            end select
        end if
    next
end sub

```

```
        case 8
            i_9_bit = i_9_bit + 256
        end select
    end if

    putpin CLK,1
next
End Sub
```

Appendix B Source Codes for Excel Visual Basic

B.1 Code for Module: StartForm

Option Explicit

```
Public BusySaving As Boolean
Public BusyModifying As Boolean
'Public StrErr1 As String
Public CarryOn As Boolean

'-----
'Private
'Sub UserForm_Layout()

Sub StartForm1()

    ' Do these first.
    ' Open "TempSens.xls" and "TempLog.xls"
    ' Ensure that Web_Update1 & 2 & 3 are enabled.
    ' For "TempSens.xls", publish TempChart to website,
    ' (without interactivity) as TempChart.htm

    ' Click inside the 'StartForm1' function, then run.
    ' Connect the receiver board anytime,
    ' before or after 'StartForm1'.
    '-----
    Dim StrErr2 As String
    StrErr2 = "Same Line"
    ' ASCII Values 8, 9, 10, and 13 convert to
    ' backspace, tab, linefeed, and
    ' carriage return characters, respectively.
    ' They have no graphical representation but,
    ' depending on the application,
    ' can affect the visual display of text.

    CarryOn = True
    BusySaving = True

    Open "Report.txt" For Append As #77
    'Print #77, "Error Logging has started"; Tab(2); "Same Line"
    ' It won't stay on the same line.
    'Print #77, StrErr1, Tab(2), StrErr2
    ' It won't stay on the same line.
    'StrErr1 = CDate(Now) & Chr(9) & "Error logging has started."
    'Print #77, StrErr1, 'Tab is inserted.
    'Print #77, StrErr1; 'Joined up with the next one.
    'Print #77, StrErr1 'Go to next line
    'Write #77, "Second line"
    'Write #77,
    'Write #77, "3rd line after skipping"
    'Write #77,
    Print #77, CDate(Now),
    Print #77, "Error logging has started."

    Close #77
    Exit Sub
```

```

Call Web_Update1
'Close #77
If CarryOn = False Then Exit Sub

BusySaving = False
BusyModifying = False
'TimerBusy = False
NumTimers = 0

Call FMain.StartForm2

End Sub
'-----

Sub Web_Update1()

    On Error GoTo WU1err

    'Excel.Application.
    Workbooks("TempLog.xls").Activate
    '.Worksheets("TempLog").Select
    'Workbooks("TempLog.xls"). 'Cannot hide it.

    Workbooks("TempLog.xls").SaveAs Filename:= _
        "ftp://www.innovation-invention.com/d:/innovation-
invention/Telemetry/TempLog.xls" _
        , FileFormat:=xlNormal, Password:="", WriteResPassword:="", _
        ReadOnlyRecommended:=False, CreateBackup:=False

    Workbooks("TempSens.xls").Activate
    'Workbooks("TempSens.xls").Worksheets.Select
    Exit Sub

WU1err:

    CarryOn = False

    MsgBox "Make sure internet connection is available." _
        & Chr(13) & "Open TempLog.xls" _
        & Chr(13) & "Open TempSens.xls" _
        & Chr(13) & "Select the chart page." _
        & Chr(13) & "Please republish TempChart.htm to the FTP server:" _
        & Chr(13) & "Save as web page to" _
        & Chr(13) & "innovation-invention.com/Telemetry/TempChart.htm" _
        & Chr(13) & "" _
        & Chr(13) & "Connect all hardware." _
        & Chr(13) & "Finally, run the Telemetry Macro using 'Ctrl+T'" _
        , vbExclamation

    ' The following section does not work.
    Open "Report.txt" For Append As #77 'Close and open too fast? No.
    Print #77, CDate(Now),
    Print #77, "Error in Web_Update1 (1st save)."
    Print #77, CDate(Now),
    Print #77, "Error logging has ended."
    Write #77,
    Close #77

    Exit Sub

End Sub

```

B.2 Code for Form: FMain

```

'Visual Basic 6 - code to draw graph from serial data
'-----
Option Explicit

Dim CurrentTime As Variant
Dim NewTime As Variant
Dim FirstData As Boolean

Sub StartForm2()

```



```

'Init Stuff
'For VB6, it will be Form_Load()
'txtDisp.Visible = False
txtDisp.Text = ""

cmdOpen.Enabled = True
cmdClose.Enabled = False

cmdTimerStart.Enabled = True
cmdTimerStop.Enabled = False

'-----
'COMM STATUS
MSComm1.RThreshold = 1
' Fire Rx Event Every One Bytes

MSComm1.InputLen = 7 'changed from 1
' When Inputting Data, Input 7 Bytes at a time

MSComm1.Settings = "19200,N,8,1"
' 19200 Baud, No Parity, 8 Data Bits, 1 Stop Bit

MSComm1.CommPort = 1
' Open COM1

MSComm1.DTREnable = False
'Disable ACK
'-----
' CurrentTime = Now - TimeSerial(0, 1, 0)
' - 1 min so that Update_Data starts immediately
' when the first Data comes in
'FirstData = True

FMain.Show

End Sub
'-----
Private Sub cmdExit_Click()

Call cmdTimerStop_Click
Call cmdClose_Click

Open "Report.txt" For Append As #77
Print #77, CDate(Now),
Print #77, "Error logging has ended."
Write #77,
Close #77

Unload FMain

End Sub
'-----
Private Sub cmdClear_Click()

'If cmdOpen.Enabled = True And cmdTimerStart.Enabled = True Then
Call cmdTimerStop_Click
Call cmdClose_Click

Excel.Application.Workbooks("TempSens.xls").Activate
Workbooks("TempSens.xls").Worksheets("Data").Select

Workbooks("TempSens.xls").Sheets("Data").Range("B2:D1441").ClearContents
txtDisp.Text = ""
'Range("G1441").

End Sub
'-----
'Private
Sub cmdOpen_Click()
'Open Port
On Error GoTo CommPortErr

Excel.Application.Workbooks("TempSens.xls").Activate
' So that when you receive the serial data,
' it can find the workbook.
If Not MSComm1.PortOpen Then

```

```

        MSComm1.PortOpen = True
    End If

    cmdOpen.Enabled = False
    cmdClose.Enabled = True

Exit Sub
CommPortErr:
    MsgBox "The serial port could not be opened." _
        & Chr(13) & "Please make sure that other programs" _
        & Chr(13) & "are not using the serial port." _
        , vbExclamation
Exit Sub
End Sub

'-----
'Private
Sub cmdClose_Click()
    'Close Port
    On Error GoTo CommPortErr

    If MSComm1.PortOpen Then
        MSComm1.PortOpen = False
    End If

    cmdOpen.Enabled = True
    cmdClose.Enabled = False

Exit Sub
CommPortErr:
    MsgBox "The serial port could not be closed." _
        , vbExclamation
Exit Sub
End Sub

'-----
'Private
Sub cmdTimerStart_Click()
    Dim interval As Double

    ' get the interval value
    'interval = TimeSerial(0, 0, 5)

    Excel.Application.Workbooks("TempSens.xls").Activate
    Worksheets("TempSens.xls").Worksheets("Data").Select
    interval = CDBl(Worksbooks("TempSens.xls").Sheets("Data").Range("G1441").Value)
        'Period of free time between successive updates.
        'Repetition period may be longer,
        'due to slow internet connection
        'or slow processor speed.

    Worksheets("TempSens.xls").Sheets("Data").Range("F1441") = 0
        'Zero the Curr Temp

    ' start the timer with the specified interval
    Call timer_Start(interval)

    cmdTimerStart.Enabled = False
    cmdTimerStop.Enabled = True

End Sub

'-----
'Private
Sub cmdTimerStop_Click()
    ' stop the timer
    Call Timer_Stop

    cmdTimerStart.Enabled = True
    cmdTimerStop.Enabled = False

End Sub

'-----
'Private
Sub MSComm1_OnComm()

```

```

On Error GoTo OCerr
'On Comm
Dim Data As String
Dim DataLen As Integer
'Dim AscData As Integer
Dim DataNum As Integer

If BusySaving = True Then Exit Sub

If MSComm1.CommEvent = comEvReceive Then
    Data = MSComm1.Input ' Get Data (string)
    'This is the temperature in deg C.
    '(1 decimal pl.)
    DataLen = Len(Data) 'No. of char's in string
    'AscData = Asc(Data)
    'Data is a single character
    'The Ascii value will be from 0 to 255
    '(numeric)

    txtDisp.Text = txtDisp.Text + Data + " " + vbCrLf _
    + "DataLen = " + CStr(DataLen) + " " + vbCrLf _
    + ' CStr will convert the numeric data to String.

    'Temperatures in deg C.(1 decimal pl.)
    'Typical values: 36.9, 37.9, 38.8

    If (DataLen = 5) And IsNumeric(Data) Then
        DataNum = CSng(Data)
        If DataNum > 0 And DataNum < 100 Then
            'Continue
        Else
            Exit Sub
        End If
    Else
        Exit Sub
    End If

    'ActiveWorkbook.
    'Workbooks("TempSens.xls").Worksheets("Data").Range("F1441") _
    = Data
    'Excel.Application
    'Excel.Application.ActivateMicrosoftApp
    '?Application.
    'Excel.Application.Workbooks("TempSens.xls").Activate
    'Workbooks("TempSens.xls").Activate
    '.Worksheets("Data").ActiveSheet
    'Application .Workbooks("index.xls").
    'AppActivate title[, wait]
    'AppActivate "TempSens.xls", False
    'Make sure there is no screen saver.
    'Excel.Application.Workbooks("TempSens.xls").Activate
    'Workbooks("TempSens.xls").Worksheets("Data").Select

    If BusySaving = True Then Exit Sub
    BusyModifying = True
    Workbooks("TempSens.xls").Activate
    Workbooks("TempSens.xls").Worksheets("Data").Select
    Workbooks("TempSens.xls").Worksheets("Data").Range("F1441") = Data
    BusyModifying = False
    'There was always error here because the PC was
    ' busy saving the file, so you cannot write to the file.

    ' check time for 1 min
    'NewTime = Now
    'If NewTime >= CurrentTime + TimeSerial(0, 1, 0) Then
    '    Update_Data
    '    CurrentTime = NewTime
    'End If
    'If FirstData = True Then
    '    NewTime = Now
    '    FirstData = False
    '    Call Update_Data
    'End If

End If
Exit Sub
OCerr:

```

```

        'MsgBox ("Irregularity in MSComm1_OnComm. The program is still functioning well.")
        Open "Report.txt" For Append As #77
        Print #77, CDate(Now),
        Print #77, "Irregularity in MSComm1_OnComm."
        Close #77
        BusyModifying = False
        Exit Sub

End Sub
'-----

```

B.3 Code for Module: TimerModule

```

' Application.OnTimer does not work ?
' It is exactly the same code as planet-source-codes
' Now I will place the codes in a module,
' instead of the Form FMain.
' Yes. Finally, got it to work.
' Application.OnTimer only works in a Module, not a Form.

Option Explicit

Dim abc As Boolean
Dim timer_interval As Double
Dim timer_enabled As Boolean
'Public TimerBusy As Boolean
Public NumTimers As Integer

Sub timer_Start(Optional ByVal interval As Double)
    If interval > 0 Then timer_interval = interval
    timer_enabled = True
    If timer_interval > 0 Then
        NumTimers = NumTimers + 1
        Application.OnTime (Now + timer_interval), "TimerModule.Timer_OnTimer"
    End If
    '.Workbooks ("TempSens.xls")
    'Timer_OnTimer ""
End Sub
'-----
Sub Timer_OnTimer()

    'Excel.Application.Workbooks("TempSens.xls").Activate
    'If TimerBusy = True Then Exit Sub
    ' In case the timer interval is too short,
    ' and one instance of Update_Data is running.
    'TimerBusy = True

    If NumTimers > 1 Then
        Open "Report.txt" For Append As #77
        Print #77, CDate(Now),
        Print #77, "NumTimers > 1. The error is corrected."
        Close #77
        NumTimers = NumTimers - 1
        Exit Sub
    End If

    If timer_enabled = False Then
        NumTimers = 0
        Exit Sub
    End If
    'FMain.txtDisp.Text = FMain.txtDisp.Text + "abc" + vbCrLf

    Call Update_Data

    'TimerBusy = False
    NumTimers = NumTimers - 1
    If timer_enabled Then Call timer_Start

End Sub
'-----
Sub Timer_Stop()
    ' stop the timer
    timer_enabled = False

```

```

End Sub
'-----
'Private
Sub Update_Data()

    On Error GoTo UDerr

    If BusyModifying = True Then Exit Sub
    BusySaving = True

    abc = True

    Workbooks("TempSens.xls").Activate
        ' -- this one can get the xls file working
        ' -- It won't show up on the taskbar.
    Workbooks("TempSens.xls").Worksheets("Data").Select

    ' shift data up
    Workbooks("TempSens.xls").Sheets("Data").Range("B2:D1440").Value2 = _
        Workbooks("TempSens.xls").Sheets("Data").Range("B3:D1441").Value2
    'Must use Value2 (Variant), not Value,
    ' otherwise the date will be nonsense (1900)

    ' Create next data
    Workbooks("TempSens.xls").Sheets("Data").Range("B1441") = CDate(Now)

    'CDate(Sheets("Data").Range("B1440")) + TimeSerial(0, 1, 0)

    ' input temperature data
    Workbooks("TempSens.xls").Sheets("Data").Range("C1441") = _
        Workbooks("TempSens.xls").Sheets("Data").Range("F1441")
    'Location cell of Input Temperature Data

    Workbooks("TempSens.xls").Sheets("Data").Range("D1441").ClearContents
    If Workbooks("TempSens.xls").Sheets("Data").Range("C1441") = 0 Then
        Workbooks("TempSens.xls").Sheets("Data").Range("D1441").Value2 = _
            "Parent's Unit not detected."
    End If

    If Workbooks("TempSens.xls").Sheets("Data").Range("C1441") = 1 Then
        Workbooks("TempSens.xls").Sheets("Data").Range("D1441").Value2 = _
            "Child's Unit not detected."
    End If

    ' zero the old input at F1441
    Workbooks("TempSens.xls").Worksheets("Data").Select
    Workbooks("TempSens.xls").Sheets("Data").Range("F1441") = 0

    'CurrentTime = NewTime
    ' check time for 1 min - test 5 sec
    'Cdbl(TimeSerial(0,0,5))
    '1# / 24 / 60 / 60 * 5
    'Cdbl(Range("G1441").Value)
    'timer_interval = Range("G1441").Value
    'NewTime = CurrentTime + Cdbl(Range("G1441").Value)
    'Application.OnTime (Now + timer_interval), "Update_Data"

    Call Web_Update2
    Call Web_Update3

    Workbooks("TempSens.xls").Save 'Save locally

    'Excel.Application.Workbooks("TempSens.xls").Activate
    'Workbooks("TempSens.xls").Worksheets("Data").Select

    BusySaving = False

    Exit Sub
UDerr:
    'MsgBox ("Irregularity in Update_Data. The program is still functioning well.")
    Open "Report.txt" For Append As #77
    Print #77, CDate(Now),
    Print #77, "Error in Update_Data()."
    Close #77
    BusySaving = False
    Exit Sub

```

```

End Sub
'-----
Sub Web_Update2()

    On Error GoTo WU2err

    Workbooks("TempLog.xls").Worksheets("TempLog").Range("B2:D1441").Value2 _
    = Workbooks("TempSens.xls").Worksheets("Data").Range("B2:D1441").Value2

    'Workbooks("TempLog.xls").Worksheets("TempLog").Select
    'Workbooks("TempLog.xls").Save    'Goes to server

    Workbooks("TempLog.xls").SaveAs Filename:= _
    "ftp://www.innovation-invention.com/d:/innovation-
invention/Telemetry/TempLog.xls" _
    , FileFormat:=xlNormal, Password:="", WriteResPassword:="", _
    ReadOnlyRecommended:=False, CreateBackup:=False

    Exit Sub
WU2err:
    'MsgBox ("Irregularity in Web_Update2. The program is still functioning well.")
    Open "Report.txt" For Append As #77
    Print #77, CDate(Now),
    Print #77, "Web_Update2: Could not upload TempLog.xls."
    Close #77
    Exit Sub

End Sub
'-----
Sub Web_Update3()

    On Error GoTo WU3err

    'Excel.Application.
    Workbooks("TempSens.xls").Activate
        ' -- this one can get the xls file working
        ' -- It won't show up on the taskbar.
        ' -- should activate before selecting it.

    'Workbooks("TempSens.xls").Worksheets.Select
    Workbooks("TempSens.xls").Sheets("TempChart").Select

    '    With Workbooks("TempSens.xls").PublishObjects("index_17584")
    '        .HtmlType = xlHtmlStatic
    '        .Publish (True)    'changed to True
    '                            'True - replaces the file
    '        .AutoRepublish = False
    '        'Don't automatically republish when saving
    '    End With
    '    ' Goes to server

    With Workbooks("TempSens.xls").PublishObjects("index_17584")
        .HtmlType = xlHtmlStatic
        .Filename = _
        "ftp://www.innovation-invention.com/d:/innovation-
invention/Telemetry/TempChart.htm"
        .Publish (True)    'changed to True
                            'True - replaces the file
        .AutoRepublish = False 'Don't automatically republish when saving
    End With
    '    ' Goes to server

    Exit Sub
WU3err:
    Open "Report.txt" For Append As #77
    Print #77, CDate(Now),
    Print #77, "Web_Update3: Could not republish TempChart.htm."
    Close #77
    'Workbooks("TempSens.xls").Activate
    Exit Sub

End Sub

```

Appendix C HTML Code

TempChartControl.htm

```
<HTML>

<HEAD>

<TITLE> Real-time Temperature Chart </TITLE>

<META http-equiv="REFRESH" content="15; URL=TempChartControl.htm" >

<!-- Refresh every 15 seconds -->

</HEAD>

<!-- Have Frames means no Body -->

<FRAMESET rows="0%,*">
    <FRAME src="Top.htm" >
    <FRAME src="http://www.innovation-
invention.com/Telemetry/TempChart.htm">
</FRAMESET>

</HTML>
```

Appendix D Circuit Diagrams

The following figures show the circuit diagrams of the initial hardware setup.

Microsoft Office Visio 2003 was used to draw the circuit diagrams.

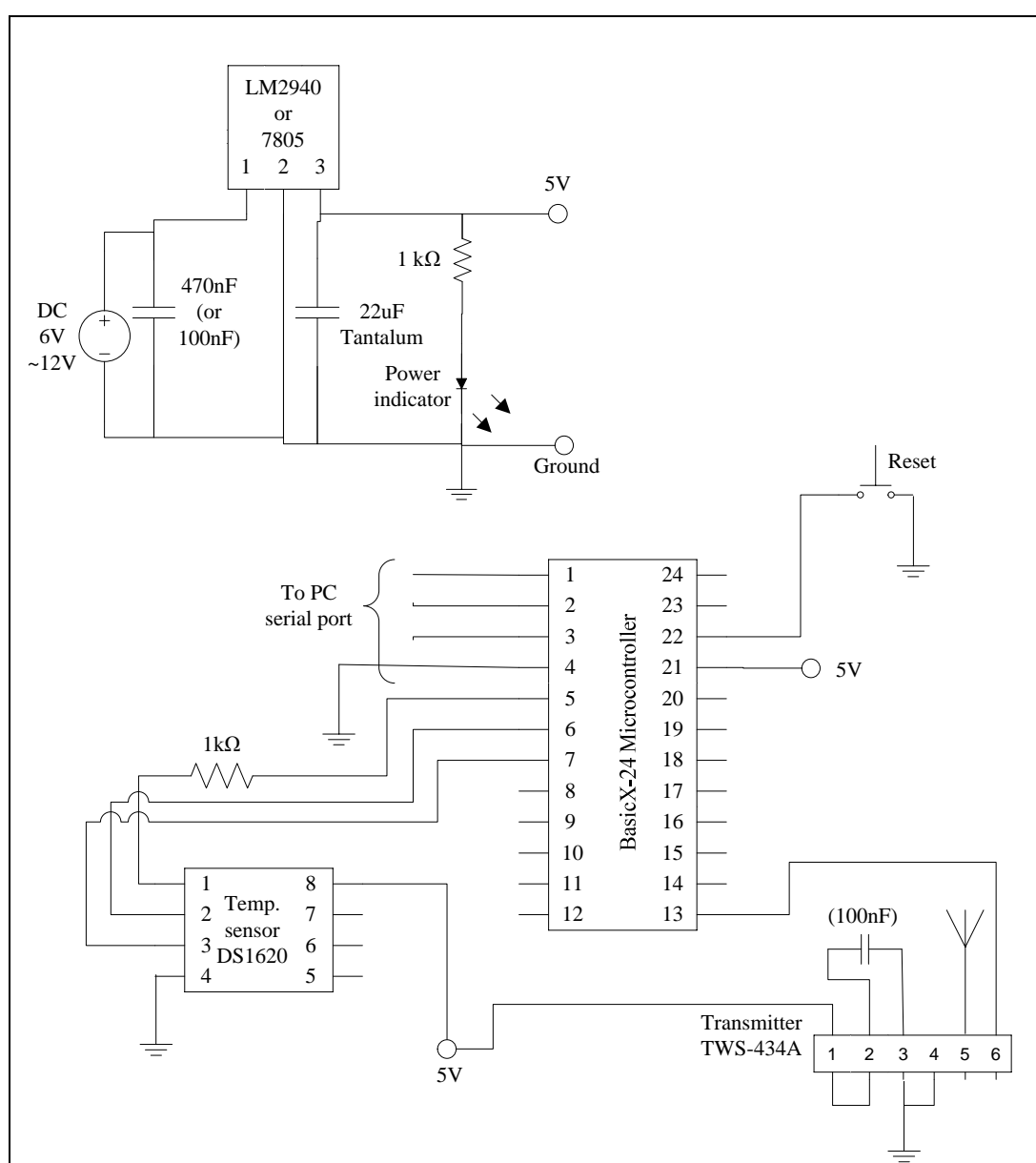


Figure 33: Initial Circuit Diagram for Child's Unit

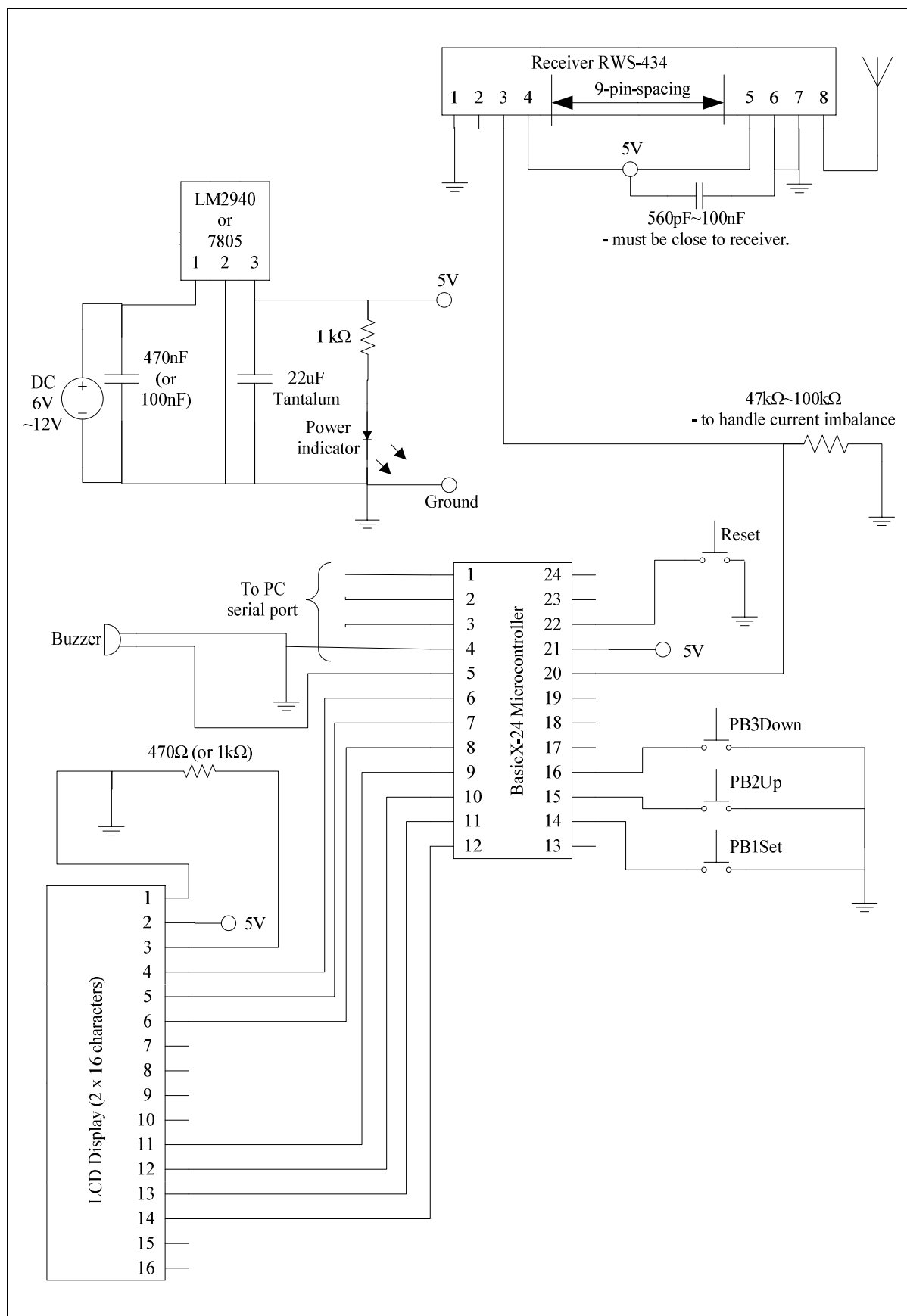


Figure 34: Initial Circuit Diagram for Parent's Unit

The following figures show the circuit diagrams of the final hardware setup.

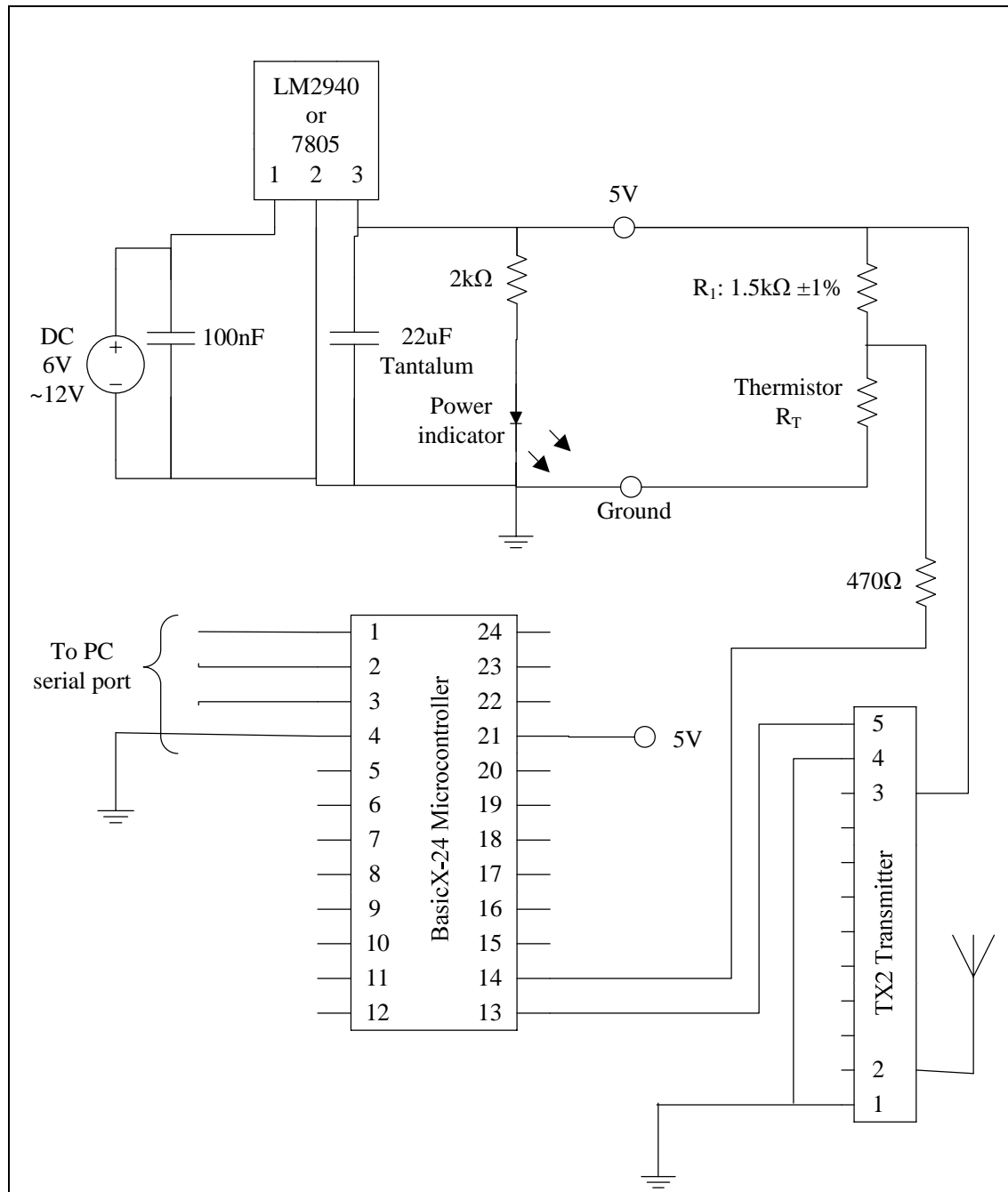


Figure 35: Final Circuit Diagram for Child's Unit

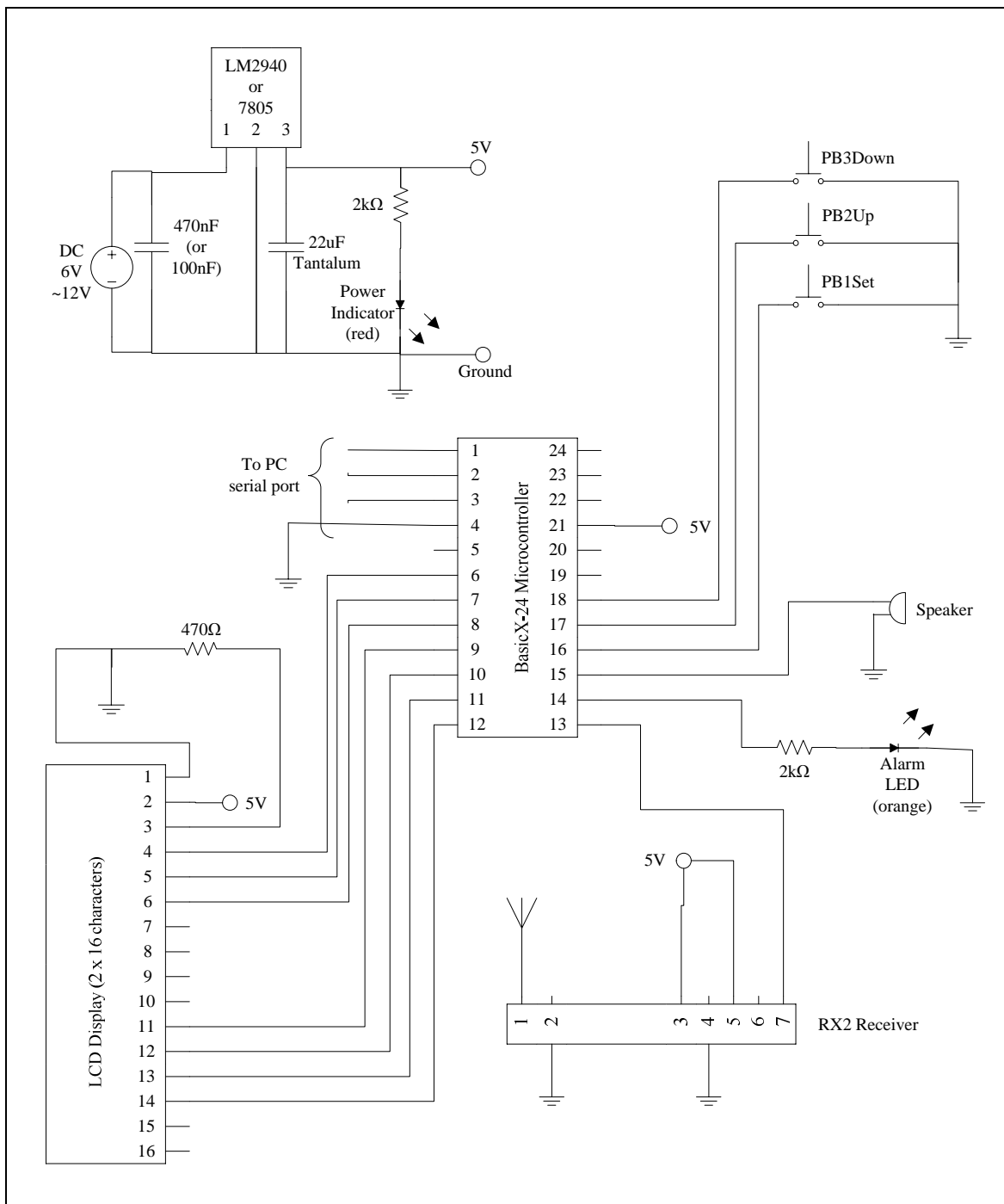


Figure 36: Final Circuit Diagram for Parent's Unit