

**DOKUZ EYLUL UNIVERSITY  
GRADUATE SCHOOL OF NATURAL AND APPLIED  
SCIENCES**

**ARM MICROCONTROLLER BASED WIRELESS  
DATA LOGGER**

by  
**Cemal ÇİÇEKDEŞ**

**February, 2011  
İZMİR**

# **ARM MICROCONTROLLER BASED WIRELESS DATA LOGGER**

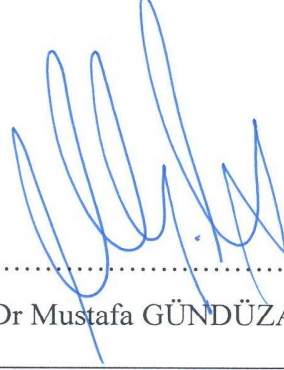
**A Thesis Submitted to the  
Graduate School of Natural and Applied Sciences of Dokuz Eylul University  
In Partial Fulfillment of the Requirements for the Degree of Master of Science  
in Electrical & Electronics Engineering Program**

**by  
Cemal ÇİÇEKDEŞ**

**February, 2011  
İZMİR**

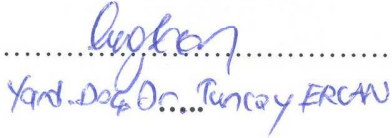
## M.Sc THESIS EXAMINATION RESULT FORM

We have read the thesis entitled “ARM MICROCONTROLLER BASED WIRELESS DATA LOGGER” completed by CEMAL ÇİÇEKDEŞ under supervision of PROF.DR MUSTAFA GÜNDÜZALP and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

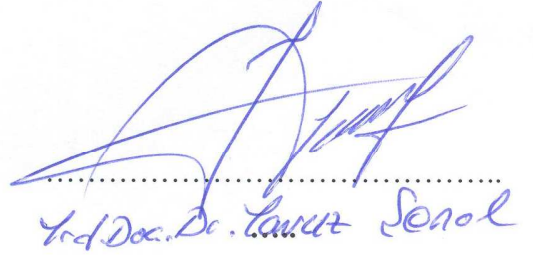


Prof. Dr Mustafa GÜNDÜZALP

Supervisor



(Jury Member)



(Jury Member)



Prof. Dr. Mustafa SABUNCU

Director

Graduate School of Natural and Applied Sciences

## ACKNOWLEDGEMENTS

First of all, I would like to express my gratefulness and my special thanks to my supervisor Prof.Dr. Mustafa Gündüzalp for his guidance, patience and support as well as his understanding.

I want to thank to my sister Bircan for her support and encouragement. She has always been positive and patient to me and she has been the one who always cheered me up.

My brother Malik deserves my sincere thanks for his support and strong brotherhood. The feeling of being loved by someone who will always support me has always given me strength and confidence.

I can not forget to mention about my family's support and love, giving me the opportunity to become the engineer that I have decided to be since I was a child.

Many thanks to all people who have been helpful for my study. With sincere thanks again, I dedicate this thesis to my family.

*Cemal ÇİÇEKDEŞ*

# **ARM MICROCONTROLLER BASED WIRELESS DATA LOGGER**

## **ABSTRACT**

In this study, a battery powered wireless data logger which has multichannel sensor as well as real time data monitoring support is designed using ARM based STR7 (from ST semiconductor company) microcontroller and which has ability to store the data to a SD (Secure Digital) memory card. Most data loggers use EEPROM (Electrically Erasable Programmable Read-Only Memory) to store the data. They need to be taken where the computer is placed in order to get stored data from their EEPROM using special software programs. The designed data logger uses FAT (File Allocation Table) file system on the SD card rather than EEPROM hence no special software is required to read the stored data, thanks to FAT file system.

Our data logger supports five different sensors where ordinary data loggers support one or two. Another important advantage of our data logger project is that it has a PIC16F877 (from Microchip company) microcontroller based battery powered handheld receiver device which can control the data logger and get the wireless data transmitted from the data logger in real time. Most data loggers need a computer for real time data monitoring but this handheld device can monitor the data coming from the data logger.

The data logger stores the data from the sensors continuously but doesn't always transmit them in order to save power. The receiver, by sending special instructions to the logger, can start data transfer, can get current date and time, can modify the date and time, and also change the sampling frequency.

**Keywords:** Data Logger, Wireless, Sensors, Battery Powered, SD Card

# ARM MİKRODENETLEYİCİ TABANLI KABLOSUZ VERİ KAYDEDİCİSİ

## ÖZ

Bu çalışmada, ARM tabanlı STR7 (ST firmasından) mikrodnetleyici kullanılarak, çok kanallı sensör desteđi ve gerçek zamanlı veri gözleme özelliđi ile birlikte verileri SD hafıza kartına depolayan pil beslemeli, kablosuz bir veri kaydedicisi tasarlanmıştır.

Veri kaydedicilerin büyük çođunluđu, verileri kaydetmek için EEPROM kullanır. Kaydedilen verileri almak için bilgisayarın olduđu yere götürölmeleri ve özel programlar aracılıđıyla hafızalarının okunması gerekir. Tasarlanan veri kaydedicisi EEPROM yerine üzerinde FAT dosya sistemi olan SD hafıza kartı kullandıđı için kaydedilen verileri elde etmede özel programlara ihtiyaç duymaz.

Sıradan veri kaydedicileri bir veya iki sensörü desteklerken, veri kaydedicimiz, beş farklı sensörü desteklemektedir. Veri kaydedici projemizin diđer bir avantajı da PIC16F877 (Microchip firmasından) mikrodnetleyici tabanlı, verileri gerçek zamanlı alıp gösterebilen ve veri kaydedicisini kumanda edebilen, pil beslemeli, kablosuz, taşınabilir bir cihaza sahip olmasıdır. Birçok veri kaydedicisi, gerçek zamanlı veri gözlemesi için bir bilgisayara ihtiyaç duyarken bu taşınabilir cihaz verilerin gerçek zamanlı gösterilmesini sağlar.

Veri kaydedicisi, verileri sürekli olarak kaydeder fakat güçten tasarruf etmek için onları devamlı olarak göndermez. Alıcı cihaz, veri kaydedicisine özel komutlar göndererek, veri aktarımı başlatabilir, o anki tarih ve zaman bilgisini alabilir, tarihi ve zamanı deđiştirebilir ve örnekleme frekansını ayarlayabilir.

**Anahtar Kelimeler:** Veri Kaydedicisi, Kablosuz, Sensör, Pil Beslemeli, SD Kart

# CONTENTS

	<b>Page</b>
<b>THESIS EXAMINATION RESULT FORM .....</b>	<b>ii</b>
<b>ACKNOWLEDGEMENTS.....</b>	<b>iii</b>
<b>ABSTRACT .....</b>	<b>iv</b>
<b>ÖZ.....</b>	<b>v</b>
<b>CHAPTER ONE – INTRODUCTION .....</b>	<b>1</b>
1.1 Data Loggers .....	1
1.2 Data Logger Application Areas .....	2
<b>CHAPTER TWO – DATA LOGGER .....</b>	<b>3</b>
2.1 Microcontroller.....	3
2.2 Storage .....	6
2.3 Wireless Communication .....	6
2.4 Sensors .....	7
2.4.1 Temperature Sensor.....	7
2.4.2 Humidity Sensor .....	8
2.4.3 Acceleration Sensor .....	9
2.4.4 Light Intensity Sensor .....	10
2.4.5 Pressure Sensor .....	12
2.5 Battery .....	13
<b>CHAPTER THREE – REMOTE CONTROLLER DEVICE .....</b>	<b>14</b>
3.1 Microcontroller.....	14
3.2 Display.....	15
3.3 Keypad .....	16

3.4 Wireless Communication .....	18
3.5 Battery .....	18
<b>CHAPTER FOUR – HARDWARE DESIGN OF THE PROJECT.....</b>	<b>19</b>
4.1 Data Logger Hardware .....	19
4.1.1 PCB Design .....	22
4.1.2 PCB Creation .....	24
4.2 Remote Controller Device Hardware .....	27
<b>CHAPTER FIVE – SOFTWARE DESIGN OF THE PROJECT.....</b>	<b>31</b>
5.1 Data Logger Software .....	31
5.2 Remote Controller Device Software .....	39
<b>CHAPTER SIX – CONCLUSIONS .....</b>	<b>45</b>
<b>REFERENCES .....</b>	<b>46</b>



# **CHAPTER ONE**

## **INTRODUCTION**

### **1.1 Data Loggers**

A data logger is an electronic measuring device that collects and records the measurement data coming from sensors such as temperature, pressure, relative humidity, light intensity, acceleration sensors in real time.

Typically, data loggers are handheld, battery powered devices that have microcontroller input channels, one or more communication protocols and data storage feature. They have RTC (Real Time Clock) so that collected data can be stored with time stamps. Generally data loggers have a low sampling frequency because they measure gradually changing environment parameters like humidity, temperature, pressure etc. Having a low sampling frequency help them use their storage unit efficiently and save power.

Most data loggers require software on a computer to initiate the logger and view the stored data. Especially the ones that use EEPROMs or internal flash memories have to support at least one communication protocol in order to transfer their stored data to the computer. This communication protocol is generally RS-232 (Recommended Standard 232) as it is very common for computers.

Data loggers are mostly battery powered hence their power consumption should be as optimized as possible for longer usage. In order to use less power, most of them are in sleep mode (low power consuming mode) until it's time to get another sample and store it. If the sampling frequency of a data logger is 60 seconds, then it is better to let it sleep for 60 seconds, wake up by internal timer, collect the sensor data and go to sleep mode for another 60 seconds. Sometimes a higher sampling frequency is desired to be set for a daily record of environmental parameters and in this case the data logger may consume more power.

## **1.2 Data Logger Application Areas**

The data loggers are used for collecting and recording information about any environmental parameters, industrial parameters of process in a production line, experimental field tests and many more. People have curiosity for learning what is happening when no one is around. For example, you may wake up in the morning and find out that sometime during the night the power had gone out. You may not care when the power went out or how long it was out, however, if you have a large cooler full of perishable food, your situation would be quite different. You would want to know how long the power was down and how high the temperature rose and for how long. This is where a data logger is useful.

The application areas of the data loggers are quite wide. Snow avalanche monitoring, dam flood monitoring, meteorology, industrial process monitoring, vehicle testing, wildlife research, road traffic counting are some of the important application areas of data loggers. Obtained data from the data loggers helps people to create statistic graphics for forecasting.

## **CHAPTER TWO**

### **DATA LOGGER**

Data logger project can be considered as two different units or devices: Data logger and the remote controller device. Data logger is the data logging part which consists of different blocks and components like microcontroller part, storage block, wireless communication module, sensor components and battery power unit. The remote controller device is the device which can monitor and control the data logger. Data logger parts will be introduced separately.

#### **2.1 Microcontroller**

An ARM based STR7 microcontroller from STMicroelectronics Inc. is used for the control and process operations. The STR7 family of 32-bit microcontrollers offers the best of both 16 and 32-bit words. It combines the industry-standard ARM7TDMI® 32-bit RISC (Reduced Instruction Set Computing) core, featuring high performance, very low power, and very dense code. The STR7 family comprises a wide range of devices fully loaded in generous packages or optimized in low pin-count packages. STR711 pinout for LQFP (Low-Profile Quad Flat Package) is shown in Figure 2.1 (ST Microelectronics, Ltd, 2008)

The STR7 microcontrollers are aimed to have power and flexibility of a 32-bit microcontroller plus a rich set of on-chip peripherals. This family targets embedded control applications such as industrial control, factory automation, point-of-sale, medical and testing equipment, as well as telecom and consumer applications.

Main features and benefits of this microcontroller are;

- High-performance, industry standard core ARM7 RISC 32-bit CPU for future-proof microcontrollers that easily adapt to customer requirements
- Extensive software and tool support with the complete STR7 library supporting all peripherals including USB dramatically reduces development time and increases ease-of-use

- High-endurance embedded flash with low latency for deterministic behavior in real-time applications
- Industrial temperature range (-40°C to + 85 °C and +105°C) and choice of 3.3V or 5.0V native devices provides flexible application options
- The largest choice of on-chip peripherals including up to 3 CAN, USB, SPI, I2C, 4 UART, 20 timers reduces the system cost
- Flexible power and clock management allows full control over power consumption and performance/power tradeoffs
- 256K Bytes Program Flash
- 64K Bytes RAM (Random Access Memory)
- Real Time Clock (RTC)
- Four 12 bit ADC (Analog to Digital Converter)
- Four UARTs (Universal Asynchronous Receiver/Transmitter)
- Two I2Cs (Inter Integrated Circuit)
- Two SPIs (Serial Peripheral Interface)
- Five 32 bit timers

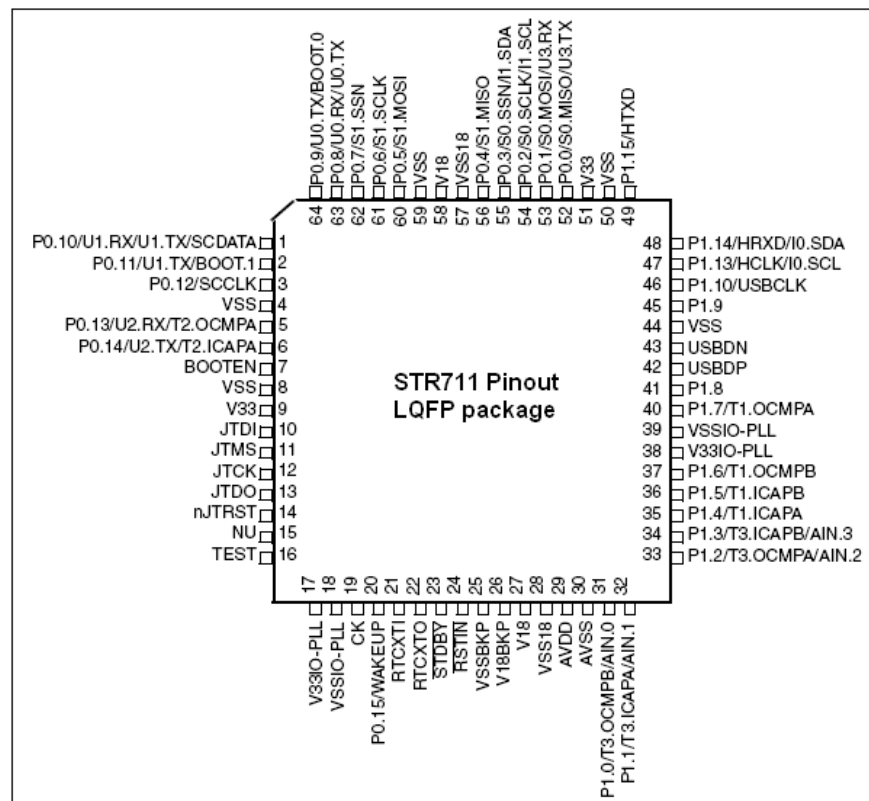


Figure 2.1 STR711 pinout for LQFP (Low-Profile Quad Flat Package)

Program flash memory is large enough to store developed software binary file in the microcontroller and the RAM is also big enough for the memory requirements of the software. RTC is required for the time stamps of the recorded sample so the samples will have the time information when they are recorded to the storage unit. ADCs are required for analog to digital conversion process for analog sensors. Microcontroller can only process the digital data so all analog signals are supposed to be converted into digital data.

UART is used to communicate between the microcontroller and the wireless transceiver because wireless transceiver supports only UART protocol. I2C is used to communicate between the digital sensors which support this protocol. SPI is another communication protocol between digital integrated circuits. This protocol is used for communicating between microcontroller and the SD card. Timers are required for the time based interrupts which are generally used for periodical tasks in the software.

The microcontroller is responsible for all the operations and processes during the data logging. These operations include reading the sensors, performing the analog to digital conversions, storing the collected data on SD card, transmitting the sensor measurements in real time, processing the control commands coming from the remote controller device.

Once powered up, microcontroller starts to run the loaded software and initializes other components like SD card and sensors as well as its own blocks such as clock frequency, GPIO (General Purpose Input/Output) pins, and communication protocols. GPIO pins will be configured according to their usage by the microcontroller. Some of them need to be configured as input to carry data coming from the sensors, and some of them need to be configured as output to carry the necessary commands and signals to the sensors and other parts.

## 2.2 Storage

In this data logger project, SD card is used as data storage unit. This choice provides some advantages like big storage capacity (up to 4 GB) and FAT file system support as well as easy-to-replace feature.

Unlike EEPROM storage units, SD card doesn't require any software to get the stored data as it supports FAT file system. A text based file is opened in order to store sensor information with time stamps and sensor identities. Any computer that has a card reader can read SD card and open the text based file to access the log.

## 2.3 Wireless Communication

The data logger has a wireless transceiver module in order to send real time sensor information to the remote controller device. This module is called Xbee Pro from MaxStream Inc. Main features of the module are;

- Up to 100 m indoor/urban range
- Up to 1500 m outdoor range
- Less than 10  $\mu$ A power-down current
- UART communication protocol
- Programmable output power

The module communicates to the microcontroller via UART protocol at 9600 baud rate. When microcontroller reads sensor values and prepares the data to store on SD card, it also sends the same data to the wireless module via its UART port. The wireless module gets the data and transmits them through the air.

The data logger will always log the sensor data but will not always send it to the remote controller device in order to reduce the power consumption. The wireless module supports pin controlled sleep mode and this pin is controlled by the microcontroller. The microcontroller pulls down the sleep mode control pin of the

module for 2 seconds and this action keeps the module awake for 2 seconds. After two seconds, the microcontroller pulls up the sleep mode control pin for 6 seconds and the module is in sleep mode for 6 seconds. This sleep and wake up operation are repeated until a wake up command is sent from remote controller device.

The remote controller device of the project sends wake up command to the module whenever real time data transfer is required. It sends 20 consecutive wake up commands in 8 seconds so that the command is guaranteed to be received by the module as it is awake only for 2 seconds in every 8 seconds.

Once the wake up command is received and processed by the microcontroller, the module is no longer in sleep mode. It will stay awake as long as the remote controller device keeps it awake by sending wake up command in every 20 seconds. The module will enter sleep and wake up cycle again if the remote controller device doesn't send wake up command for 60 seconds.

## **2.4 Sensors**

The data logger has five different sensors. These are temperature, humidity, light intensity, pressure and acceleration sensors.

### ***2.4.1 Temperature Sensor***

The temperature sensor in the project is DS1620 from Maxim Integrated Products, Inc. This sensor is a digital temperature sensor with 3-wire communication support and requires no external component to be able to operate. It can measure the temperatures between -55 °C and 125 °C in 0.5 °C increments. This temperature measurement range is suitable for environmental temperature changes in daily life.

The 3-wire bus is comprised of three signals. These are the DQ (data) signal, the CLK (clock) signal, and the RST (reset) signal. All data transfers are initiated by driving the RST input high. Communication is terminated by driving the RST input

low. A clock cycle is a sequence of a falling edge followed by a rising edge. The data must be valid during the rising edge of the clock cycle for data inputs.

Data bits are output on the falling edge of the clock and remain valid through the rising edge. The DQ pin goes to a high impedance state while the clock is high during the data reading from the DS1620. Taking 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 sensor has a few simple commands which tell it what to do. A hex value of AA will tell the sensor to send the last temperature conversion result and the next nine clock cycles will output this value. EE hex value will tell the sensor to start the conversion in order to prepare the temperature value in digital format. 22 hex value stops temperature conversion and no further data is required. This command may be used to halt a DS1620 in continuous conversion mode. After issuing this command the current temperature measurement will be completed and then the DS1620 will remain idle until start command is issued to resume the operation.

The microcontroller initializes the temperature sensor after power up and puts it into continuous conversion mode so that the sensor keeps measuring the temperature at all times. The temperature value is read from the sensor in every sampling frequency by the microcontroller and stored to the SD card with its time stamp.

#### ***2.4.2 Humidity Sensor***

The humidity sensor is HH10D from Hope Microelectronics Co., Ltd. This relative humidity sensor module is comprised with a capacitive type humidity sensor, a CMOS (Complementary Metal Oxide Semiconductor) capacitor to frequency converter and an EEPROM used for holding the calibration factors. Due to the characteristics of capacitor type humidity sensor, the system can respond to humidity change very fast. The sensor is calibrated twice at two different accurate humidity chambers and two unique sensor specific coefficients are stored onto the



EEPROM on the module. These two unique sensor coefficients are used for humidity calculations.

Sensor specific unique coefficients are read from the EEPROM on the module via I2C communication protocol by the microcontroller. These coefficients are required to be read once as they are specific for each module and don't change during the life time of the sensor module.

Unlike most relative humidity sensors, this sensor outputs the humidity measurement as frequency. In order to measure the frequency output of the sensor module, a timer and a counter are used in the microcontroller. The counter is clocked by frequency output of the sensor module and the frequency is calculated as a counter value in every second by the help of the timer. The timer creates an interrupt in every seconds and the counter value is reset for the next measurement.

The two coefficients and the frequency value are used in a formula to calculate the relative humidity as follows;

$$\text{Relative Humidity (\%)} = (\text{Offset} - \text{Frequency Value}) \times \text{Sensitivity} / 4096$$

Sensitivity: Stored in the EEPROM address 10h of the sensor module

Offset coefficient: Stored in the EEPROM address 12h of the sensor module

### ***2.4.3 Acceleration Sensor***

The acceleration sensor is MMA1260KEG from Freescale Semiconductor, Inc. It is capable of measuring  $\pm 1.5g$  with an analogue linear output signal. The sensor sensitivity is typically 1200 mV/g which is high enough to measure small acceleration values.

In the data logger project, the acceleration sensor is used for slope measurement in degree format. The sensor has an offset voltage of 2.5V at zero g and zero g can be

obtained by holding the sensor perpendicular to the ground. The slope in degree format can be calculated by the formula as follows;

$$\text{Slope} = \frac{\text{CurrentOutputVoltage} - \text{OutputVoltageAtMinusG}}{\text{OutputVoltageAtPlusG} - \text{OutputVoltageAtMinusG}} \times 180^\circ$$

Output voltages obtained under earth gravity at different positions are shown in Figure 2.2. ( Freescale Semiconductor, Inc, 2009)

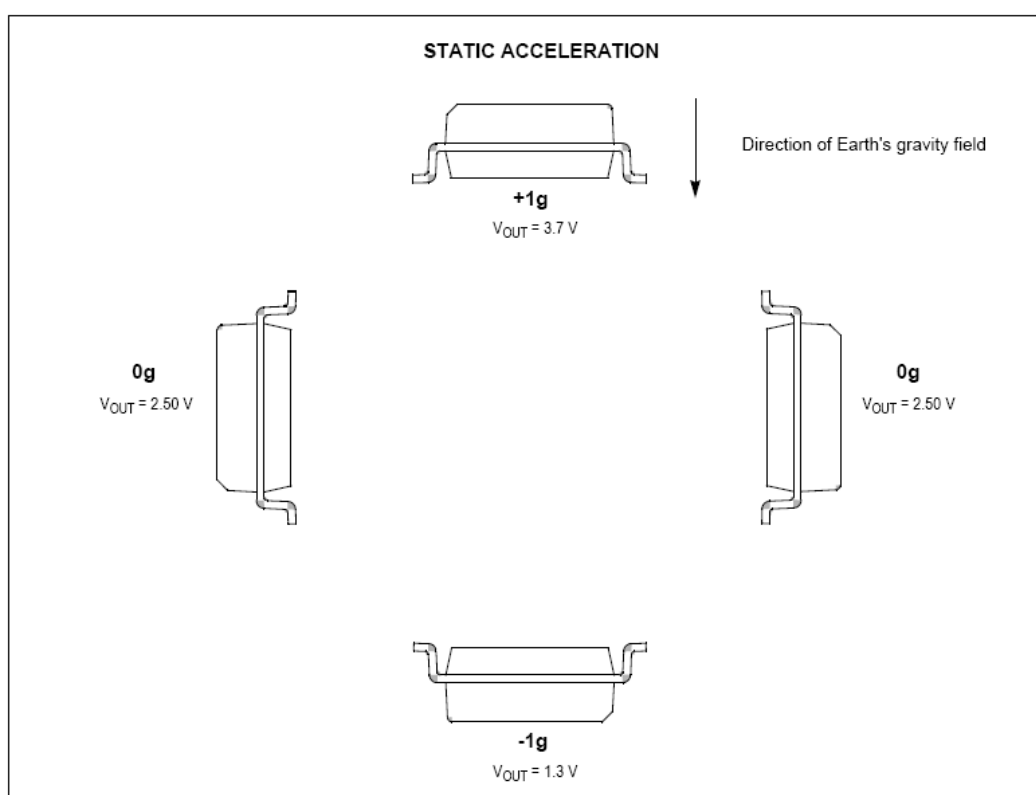


Figure 2.2 Output voltage of the sensor at different positions in static acceleration

The microcontroller converts the analogue output of the sensor into digital format by using its internal ADC so that it can calculate the slope.

#### 2.4.4 Light Intensity Sensor

The light intensity sensor is LLS05-A from Senba Optical Electronic Co., Ltd. This sensor has linear output conforming to illuminance and built-in optical filter for

spectral response similar to that of the human eye. The output of the light intensity sensor can be obtained by multiplying its current with the value of serial resistor connected to it as shown in Figure 2.3 (Senba Optical Electronic Co., Ltd, 2005)

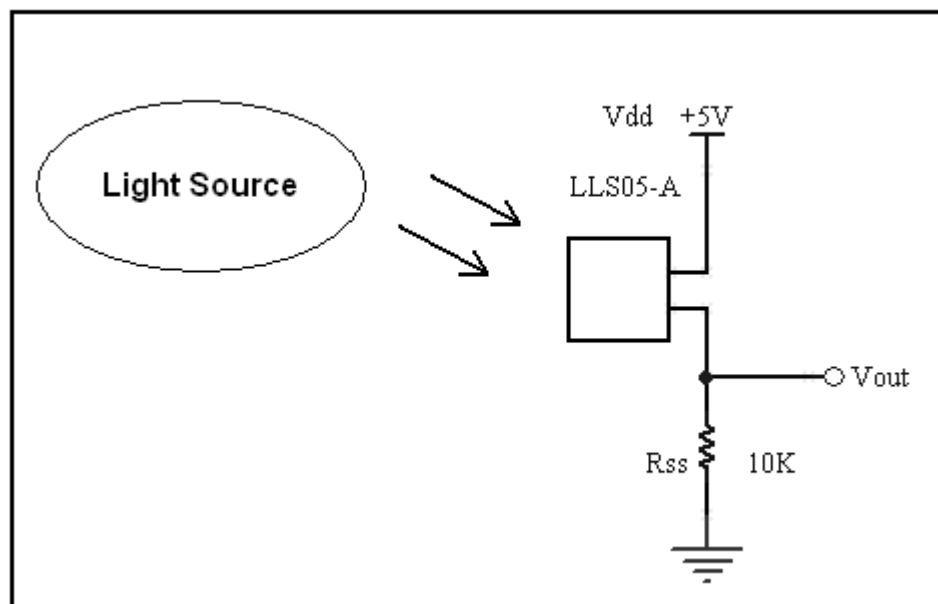


Figure 2.3 Photocurrent measurement circuit. Rss is recommended to be high stable resistor for better performance.

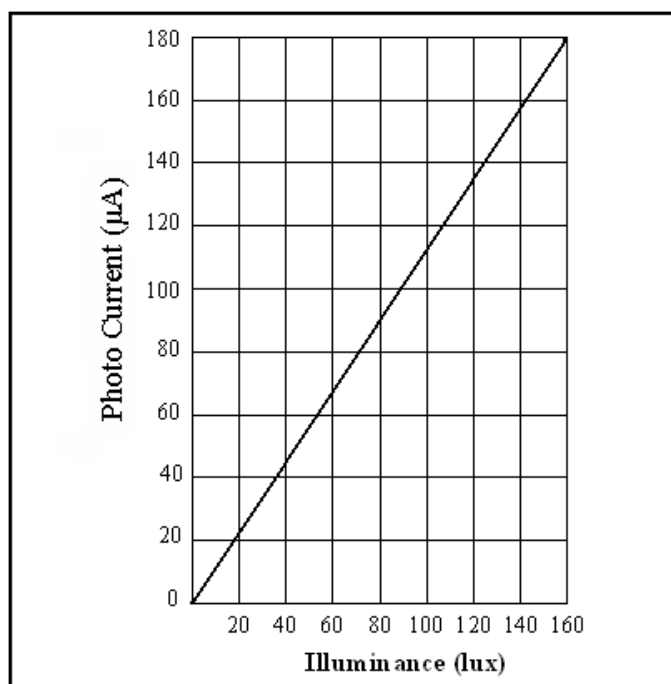


Figure 2.4 Photo current vs. Illuminance graphic of the sensor

As the more light hits the sensor surface, the more current passes through it. Microcontroller can not read the current of the sensor directly but the serial resistor will help it to read the output as voltage.

$$\text{Illuminance} = \frac{V_{out}}{R_{ss}} \times \frac{8}{9}$$

The current passing through the sensor is directly proportional to the illuminance as it can be seen from Figure 2.4 (Senba Optical Electronic Co., Ltd, 2005). The slope of the graphic is used in the formula to calculate the illuminance from the sensor current.

#### 2.4.5 Pressure Sensor

The pressure sensor in the project is MPVZ5010GW7U from Freescale Semiconductor, Inc. It can measure the pressures between 0-10 kPa (kilo pascal) and the voltage output range is 0.2V to 4.7V. Figure 2.5 shows the output voltage versus differential pressure graphic. (Freescale Semiconductor, Inc, 2005)

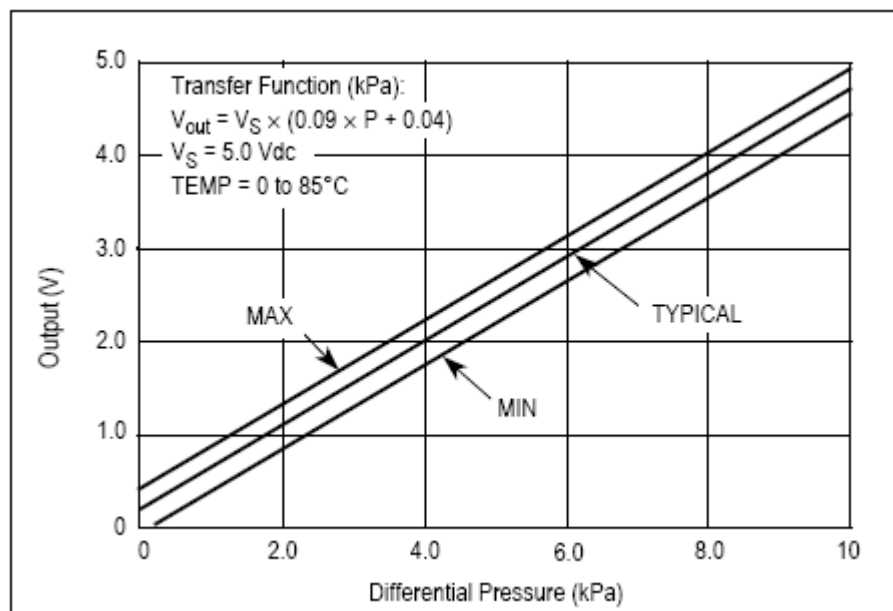


Figure 2.5 Output voltage vs. differential pressure graphic of the pressure sensor

$$\text{Pressure (kPa)} = [(Output \text{ Voltage}/Supply \text{ Voltage}) - 0.04] / 0.09$$

From the Figure 2.5, it can be seen that output voltage is directly proportional to the applied pressure in 0-10 kPa range. The microcontroller reads the output voltage in digital format by using its internal ADC and calculates the pressure value according to given formula above.

## **2.5 Battery**

A 9V alkaline battery is used as power source for the main part of the data logger. The battery powers up the microcontroller board and sensor board together. The microcontroller board requires 3.3V for its microcontroller and SD card connector. On the other hand, sensor board needs both 5V and 3.3V because all the sensors require 5V except the humidity sensor and the wireless module which require 3.3V.

The microcontroller board has its own regulator which can produce 3.3V from 9V input and powers up all the components on the board. The sensor board has LM7805 voltage regulator which can produce 5V from 9V input. For the wireless module and the humidity sensor, 3.3V is supplied from microcontroller board via two extra wires.

## CHAPTER THREE

### REMOTE CONTROLLER DEVICE

Remote controller device is a monitoring and control device of the data logger. This device receives and monitors the transmitted data from the data logger. Also it can control the data logger by sending special commands like setting the sampling frequency, getting and setting the time information. The remote controller device has a microcontroller board with keypad, display, wireless module and battery.

#### 3.1 Microcontroller

The microcontroller of the remote controller device is PIC16F877 from Microchip Technology Inc. This microcontroller has UART communication support to communicate with wireless module, 8K flash program memory, and more than 30 configurable IO (input-output) pins. (Microchip Technology Inc, 2001)

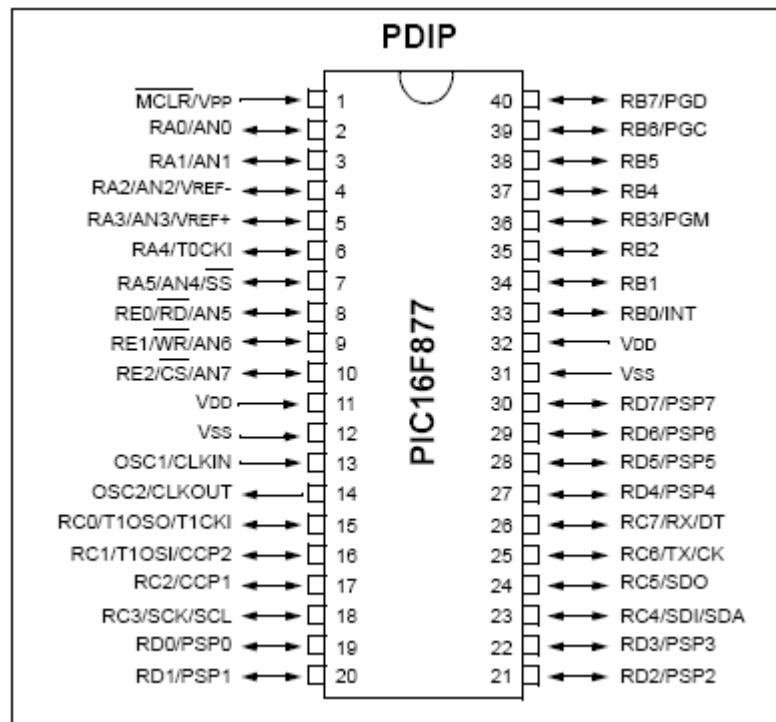


Figure 3.1 PIC16F877 Pin Diagram for PDIP (Plastic Dual Inline Package)

The microcontroller is responsible for the control of everything on the remote controller such as keypad, wireless module, and display. It controls the wireless module to get the transmitted sensor information from data logger and drives the display to show the sensor information on the display. It also reads the keypad while the user presses and decides what to do according to pressed key.

### 3.2 Display

The remote controller device has a Nokia 3310 mobile phone LCD (Liquid Crystal Display) from Nokia Corp. This display has its own display controller which is PCD8544 display controller/driver from Philips Semiconductors (acquired by NXP Semiconductors).

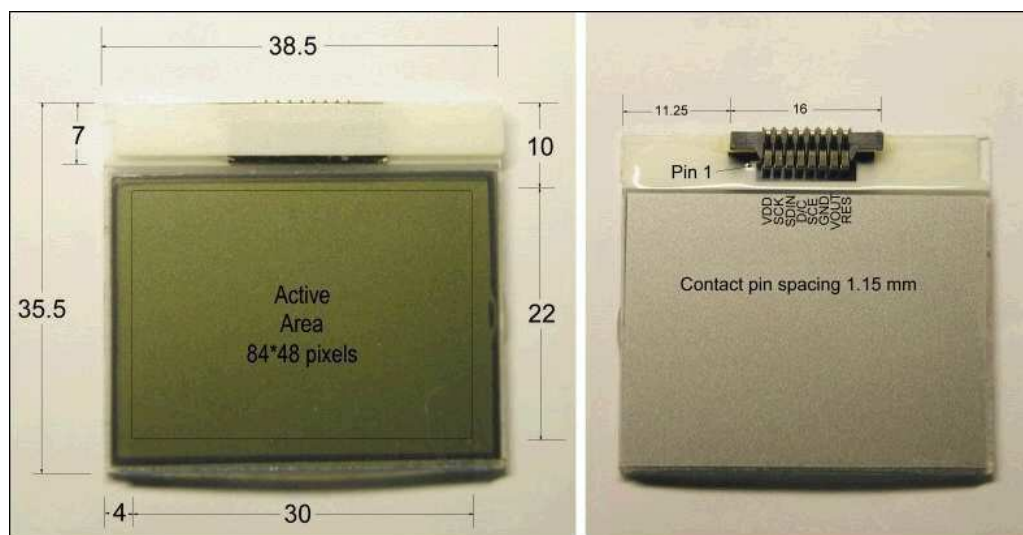


Figure 3.2 Remote controller device display, 84x48 pixels graphic LCD taken from an old Nokia 3310 mobile phone.

This display is a monochrome graphic LCD with 84x48 pixels. It is controlled via I2C communication protocol and requires 3.3V supply voltage to operate. DDRAM (Display Data RAM) of the display is divided into six banks of 84 bytes (6x8x84 bits). During RAM access, data is transferred to the RAM through the serial interface. There is a direct correspondence between the X-address and the column output number. The columns are addressed by the address pointer and the address ranges are 0 to 83 for X and 0 to 5 for Y axis.

In the horizontal addressing mode, the X address is incremented after each byte. After the last X address ( $X = 83$ ), X wraps around to 0 and Y address is incremented to address the next row. After the very last address ( $X = 83$  and  $Y = 5$ ), the address pointers wrap around to address ( $X = 0$  and  $Y = 0$ ).

### 3.3 Keypad

The keypad on the remote controller device is a 4x3 keypad with 12 keys. This keypad provides a user interface to the remote controller device and lets user enter the commands to be sent to the data logger.

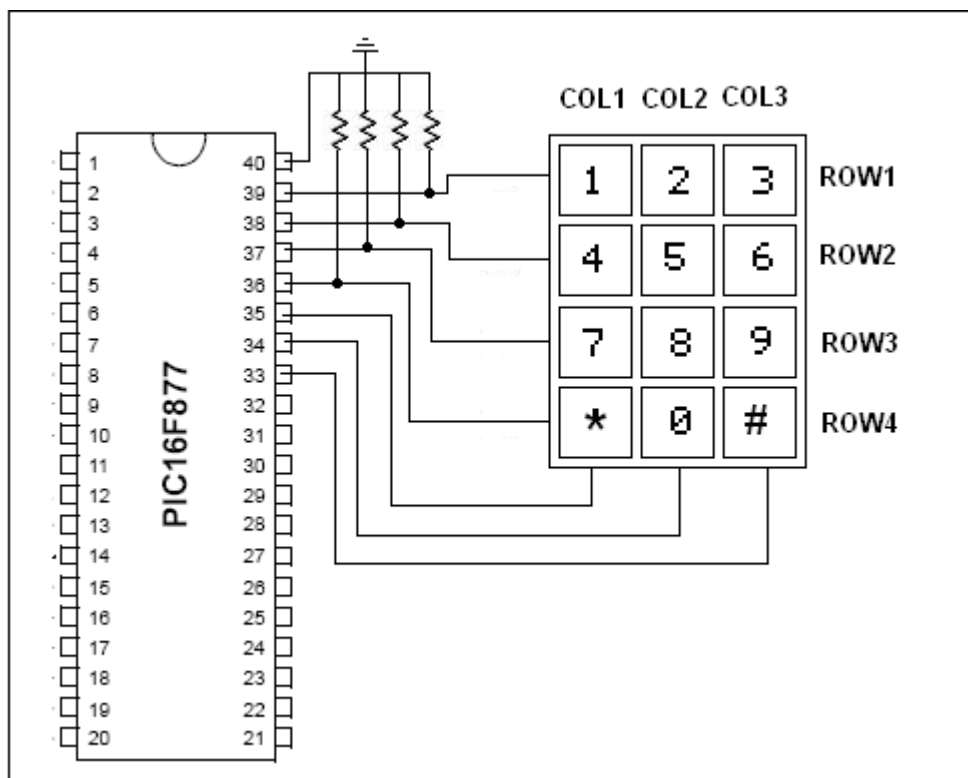


Figure 3.3 Keypad diagram of the remote controller device. Columns and rows are connected to 7 different microcontroller IOs.

Figure 3.3 shows how the keypad is connected; each square can be considered as a switch which connects the horizontal wires (rows) with the vertical wires (columns). If the button '1' is pressed, it will connect COL1 with ROW4, or pressing button '9' will connect COL3 with ROW3.



As the switches are all interconnected, we need to find a way to differentiate between the different keys. The rows are all connected to the ground with a resistor in series. In order to read which key is pressed, one of the columns is set to high and other two columns are set to low and all the rows are set to low by the resistors connected to the ground.

If one of the keys on that column is pressed, this key will connect the related row to this column and we will be able to know that any key is pressed by doing an AND operation between the number we read and 0x11111000b. The AND operation result will tell us if any key is pressed or not. If the result is 0, then none of the keys is pressed. We will set the next column to high while setting the other two to low and try to read again. This reading process will be repeated until we find the pressed key or maximum three times as we have only three columns.

Table 3.1 shows the states of the IO pins and which key is pressed according to the read value from the port pins state.

Table 3.1 Pressed key according to port pins state

State of the port pins	Pressed key
0x01000100	1
0x01000010	2
0x01000001	3
0x00100100	4
0x00100010	5
0x00100001	6
0x00010100	7
0x00010010	8
0x00010001	9
0x00001100	*
0x00001010	0
0x00001001	#

### **3.4 Wireless Communication**

The remote controller device has the same wireless transceiver module as the data logger. The microcontroller communicates to the wireless module via UART protocol at 9600 baud rate. The wireless transceiver receives the sensor data and command responses from data logger and transmits the commands from remote controller device to the data logger.

The remote controller device can be activated any time but in order to receive the sensor data, data logger must be awoken up. The remote controller sends 20 consecutive wake up command until one of the wake up command is responded by the data logger. These consecutive commands are required because data logger can only respond when it is awake during 2 seconds period otherwise it is sleeping for 6 seconds to save power.

### **3.5 Battery**

The remote controller device is powered up with three AA size batteries connected in series. The wireless module requires 3.3V and the microcontroller requires 5V in order to operate but these voltages are optimal values and supply voltages can be little bit away from these values. As three AA size batteries produce 4.5V when in series, this is still enough for the microcontroller to operate well.

The supply voltage of the wireless module is provided using two diodes in series to create 1.4V voltage drop from 4.5V and it will be 3.1V for the module. This 3.1V value is not a problem for the module as it can operate between 2.8V and 3.4V.

## **CHAPTER FOUR**

### **HARDWARE DESIGN OF THE PROJECT**

The hardware design of the project will be introduced in two separate parts as there are two devices; the data logger and the remote controller device.

#### **4.1 Data Logger Hardware**

The data logger has two different boards connected to each other via several wires. The microcontroller and the SD card connector are on the demo board called STR7-P711 from Olimex Ltd. Company. This board has all the necessary external components like resistors, capacitors, transistors etc. that microcontroller and SD card need. Main features that this demo board has are as follows;

- STR711 microcontroller
- Standard JTAG connector for debugging purpose
- USB connector
- Two channel RS232 interface and drivers
- SD card connector
- Two general purpose buttons
- Trimpot connected to ADC
- Two status LEDs and power supply LED (Light Emitting Diode)
- Buzzer
- 2x SPI connectors
- I2C connector
- Onboard voltage regulator 3.3V with up to 800mA current
- Single power supply: 6V AC or DC required, USB port can power the board
- Power supply filtering capacitor
- reset circuit with reset button
- 4 MHz crystal oscillator
- 32768 Hz crystal for RTC

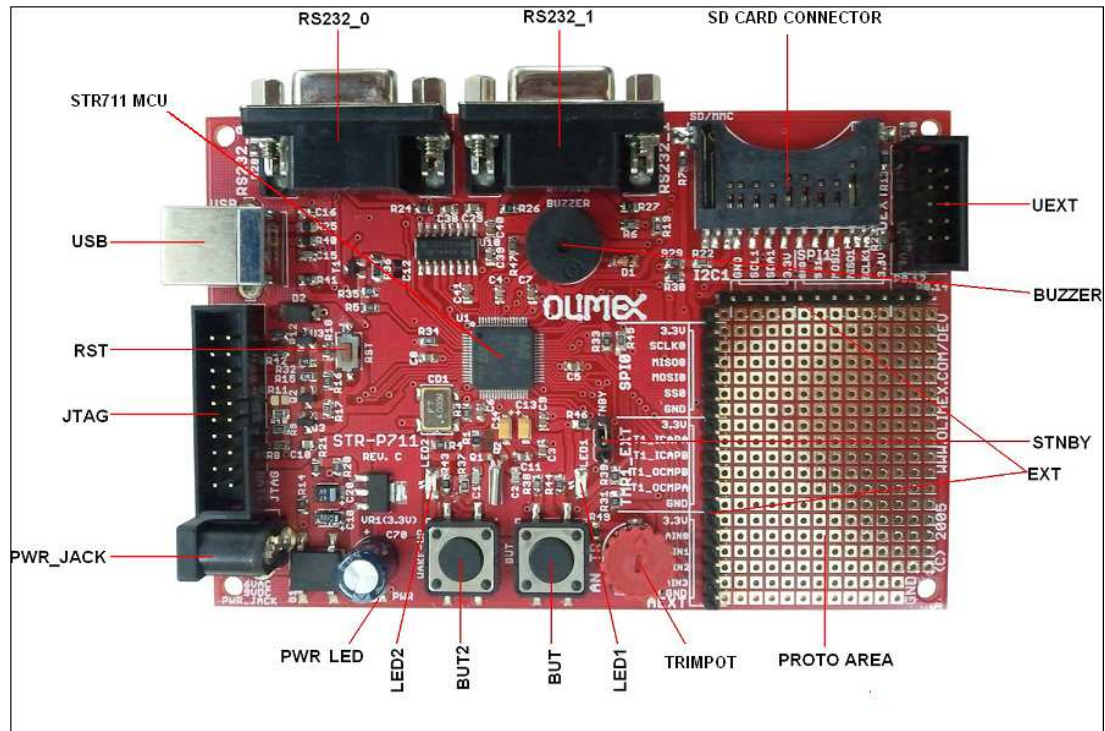


Figure 4.1 STR-P711 demo board. All required components are included on the board as well as SD card connector.

The demo board was very useful during software development stage because it has many GPIOs that can be set for testing, direct connection to the SD card connector, a trimpot connected to one of the ADC inputs to test ADC functions, two buttons for general purpose, a buzzer and two LEDs as indicators. Especially JTAG (Join Test Action Group) connector is very useful for debugging the developed software and for programming the flash memory of the microcontroller.

The second board is the sensor board that has all the sensors and the wireless module. The sensor board is designed for supporting all necessary data communications and providing supply voltages. Every sensor has its own design and required external components on the board. These components are resistors which are used for voltage dividing and current limiting, capacitors which are used for voltage regulating. The sensor board has a 5V voltage regulator on it to provide power for the sensors which require 5V. The power supply for the sensor module which requires 3.3V is provided from the demo board.

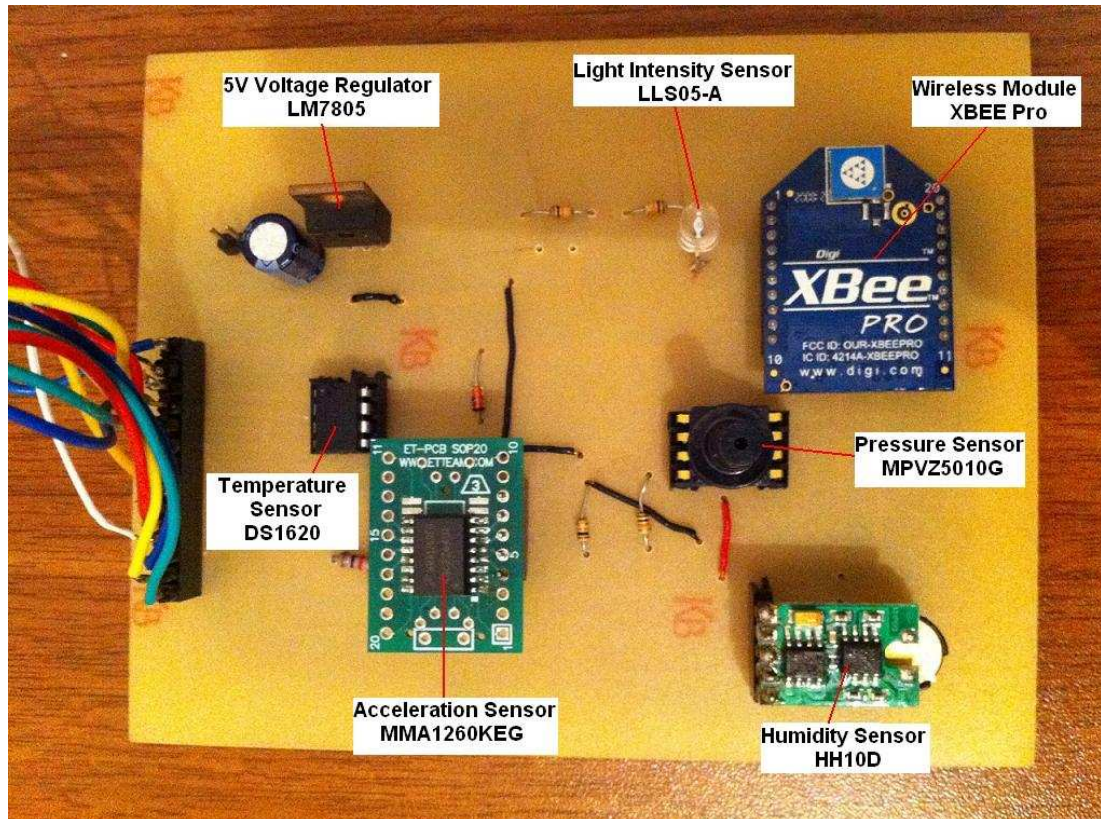


Figure 4.2 Sensor board with acceleration, humidity, pressure, temperature, and light intensity sensor including wireless module and the voltage regulator.

The acceleration, light intensity and the pressure sensors have an output voltage range which is higher than the microcontroller can accept to its ADC input. ADC of the microcontroller can convert input signals in the range of 0V to 2.5V but these three sensors have output between 0V to 5V. In order to overcome this problem, a very simple but effective way is to use voltage divider. Figure 4.3 shows the simple voltage divider used for the sensors.

Each sensor on the board has a socket except the light intensity sensor. The sockets are used for replacement purposes in case of failure of the sensor. If any sensor is damaged or malfunctions, it will be easily replaced without the need for soldering. Some sensitive sensors might get overheated and damaged during the soldering procedure but thank to the sockets, they don't have to be soldered on the board.

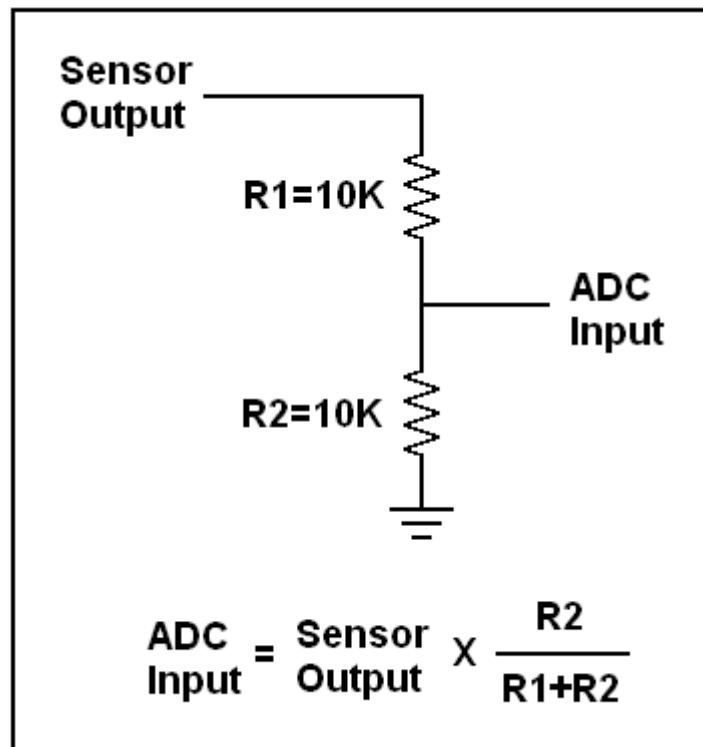


Figure 4.3 A simple voltage divider circuit used for sensor output voltage adaptation to ADC of the microcontroller.

The wires that connect two boards are placed in a pin female header connector to tidy up the connection and this is also useful when a pin connection needs to be changed due to design changes.

#### ***4.1.1 PCB Design***

The sensor board is a single layer PCB (Printed Circuit Board) designed in Proteus Design Suite program (from Labcenter Electronics Ltd.). The program helps to place the components in such a way that the components have enough room and their connections don't block each other's path.

In a PCB design, one of the most important points is placing the components very carefully so that they don't occupy much room while keeping the connecting lines far enough away from each other. First of all, we need to decide how many layers our PCB will have. The sensor board designed to be single layer in order to be printed

and created at home without a PCB machine. After deciding the number of layers, the footprint of the each component can be collected from the component library if they exist. The pressure sensor and the wireless module don't have a predefined footprint in the program library so their datasheet was used to get their footprint dimensions and create a new package in the library.

Placing the components is requires extra attention to be able to create a successful PCB. A component's footprint might be perfectly placed on the design layout but it doesn't mean it will be perfectly soldered and placed after the board is printed out. The same component's physical size also should be taken into account because some components occupy larger room than their footprint. For example, wireless module and humidity sensor occupy larger room than their footprints and if we have placed them close to each other according to their footprints, they would intersect on the component side of the board.

The traces on the PCB are another important point that should be considered very well. The distance between two parallel traces should be long enough so that the PCB can be printed and created at home without using a machine. Also through-holes are important and will be very useful if two traces are intersected when they shouldn't. In some cases, it is impossible to place all the components and make necessary connections without intersecting the traces. In such a case, through-holes are used for creating a bridge connection on the component side. This bridge connection will not exist as a trace on the PCB but two through-holes will be placed so that the designer will be able to connect these holes via a wire on the component side.

The board will need drill holes for the components to be placed and soldered. The drill hole for a component should be large enough to let the pins fit but narrow enough to solder the components well. Every component has its own pin diameter that will determine the drill hole and this information should be obtained from its datasheet for the accuracy. Also the copper round that will cover the drill hole should

be large enough to provide enough soldering area for the pins. This will help a component to be soldered and placed well for electrical connection quality.

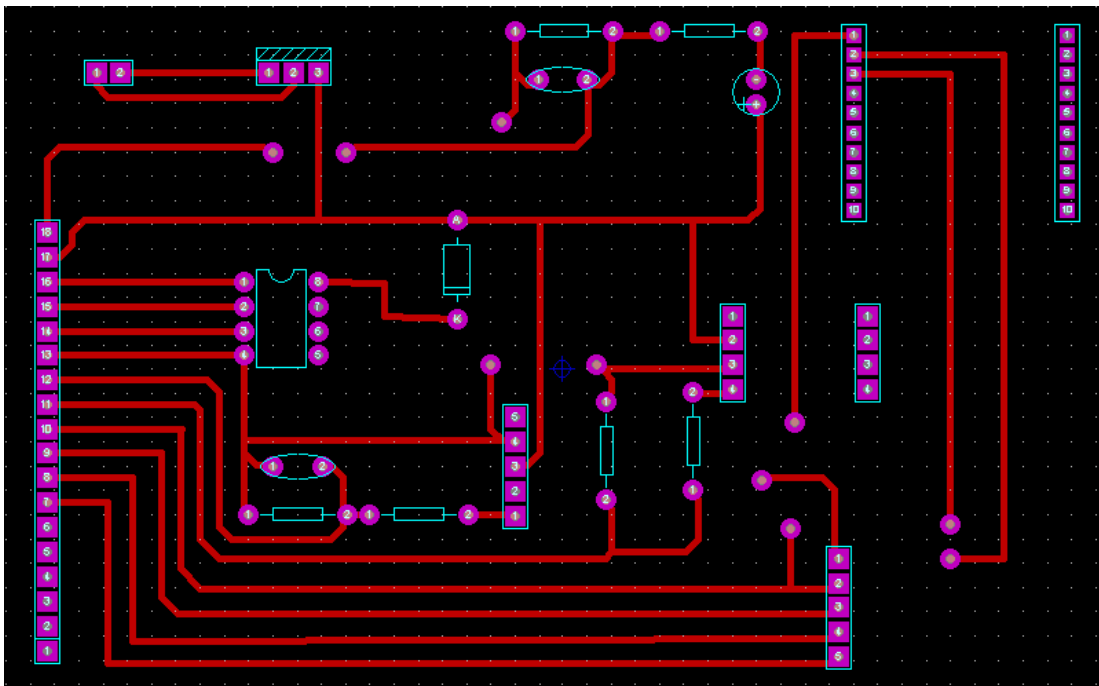


Figure 4.4 Sensor board PCB showing the bottom copper side

#### 4.1.2 PCB Creation

The sensor board and the remote controller board were created at home by using toner transfer method. This method is very useful to create cheap but effective PCBs at home. The final PCB design should consist of black-and-white bitmap of the copper bottom side of the PCB. Figure 4.4 shows the final PCB design and if it is printed in black-and-white, it will be ready to be used.

The method is based on the fact that a copper surface which is covered by toner isn't affected from the etching operation. If the designed PCB is transferred to the copper covered board and then the board is etched, the toner will protect the drawn parts and all the remaining parts will be etched.

The designed PCB should be printed on a glossy paper or on acetate and the printer should be a laser printer because only the toner can protect the copper from etching.



The next procedure is ironing the acetate carefully for toner transfer. The important point of the ironing is that the acetate should be watched carefully in order not to cause it to melt. The color of the toner will change after some time and it will be possible to distinguish which areas still needs to be ironed. The quality of the PCB is directly proportional to the toner density of the printer and the cleanness, smoothness of the copper surface of the board.

Completing the last corrections, it is time to prepare the etching mixture. This mixture consists of hydrochloride and the hydrogen peroxide. The ratio of the each component in the mixture is not strictly determined but the best way is to add some hydrochloride in a plastic container until the water level can cover the whole board and start mixing the hydrogen peroxide gradually. The board will start being etched as the hydrogen peroxide is being added and the mixture will be completed when the color of the mixture is greenish.

The PCB board is taken out of the mixture by pliers and is washed under water to clean the dangerous mixture from the board and stop etching process. After the board is washed and dried, the toner should be cleaned using some acetone. If the toner is not cleaned, it will prevent solder to stick to the copper and the electrical connection of the components will not be in a good quality. The board may be sandpapered to improve the quality of the soldering process.

The board is ready for drilling and a small driller is required. After every through-hole is drilled, the components are placed on the board and soldered. Figure 4.5 shows the complete data logger, two boards are connected.

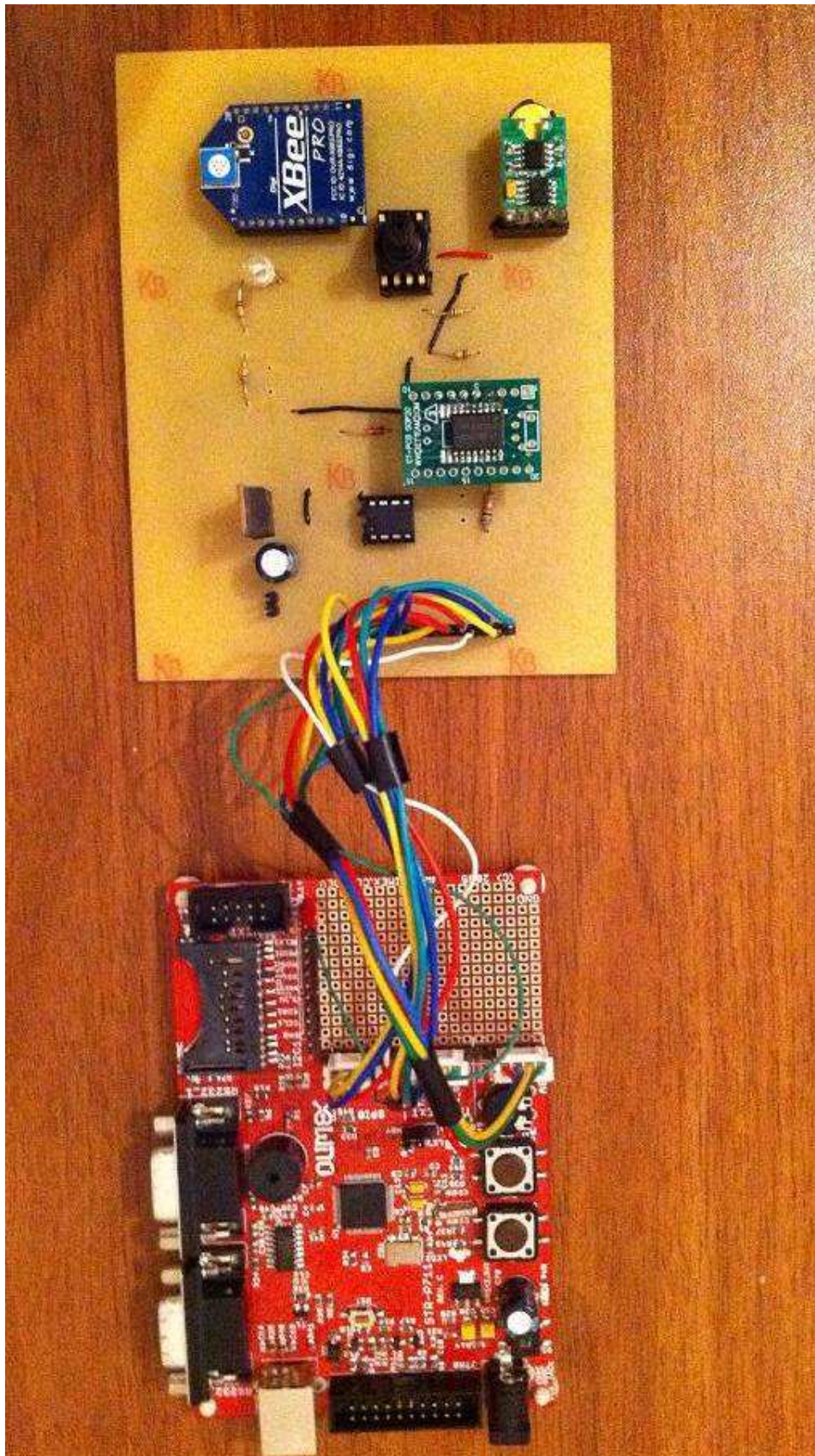


Figure 4.5 Complete data logger part. Two boards are connected.

## 4.2 Remote Controller Device Hardware

The remote controller device has a wireless module, a keypad, a LCD and a microcontroller as well as batteries. The microcontroller requires a voltage level between 4.5V-5.5V and the wireless module requires between 2.8V-3.4V. These different voltage levels are provided by three AA size batteries connected in series. The three batteries connected in series can produce 4.5V ideally but for the wireless module, we need around 3V. This problem was overcome by using two diodes in series to produce 1.4V voltage drop.

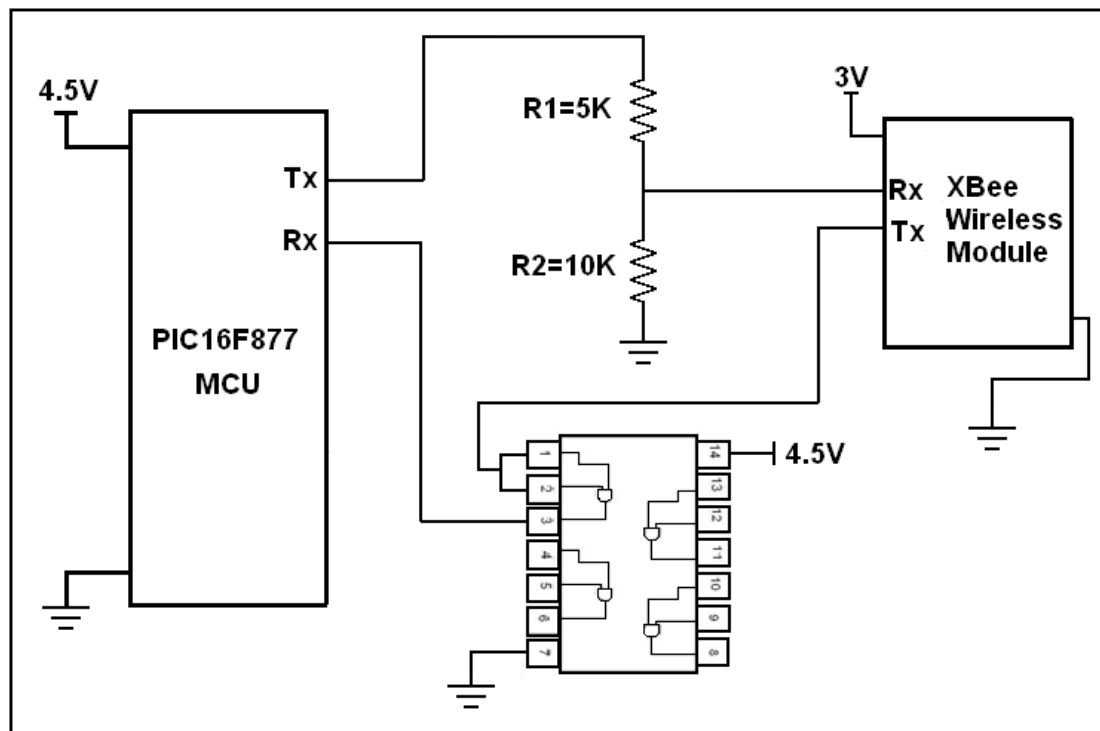


Figure 4.6 UART connection between microcontroller and the wireless module. 3V-5V compatibility problem is solved with the shown configuration.

The microcontroller and the wireless module communicate via UART protocol but the problem was again the voltage level. The microcontroller is powered with 4.5V and its data pins also will be 4.5V when set to high. This will damage the wireless module as it can not tolerate that much voltage difference. In order to solve this problem, a simple voltage divider was used. The voltage divider will divide the

voltage with a ratio of 3/2 at the transmitter pin of the microcontroller and will produce 3V from 4.5V, connecting it to the receiver pin of the wireless module.

Another problem with the voltage level occurs in the transmitter part of the wireless module. Wireless module can not supply 4.5V to the microcontroller's receiver pin but the problem was solved using a Quad 2-Input AND gate IC, 7408. The AND gate IC will assume 3V as a high input and produce 4.5V at its output. Two inputs of the gate are short circuited to the transmitter pin of the wireless module so AND gate is used as a buffer not as a logic gate. The connection is shown in Figure 4.6

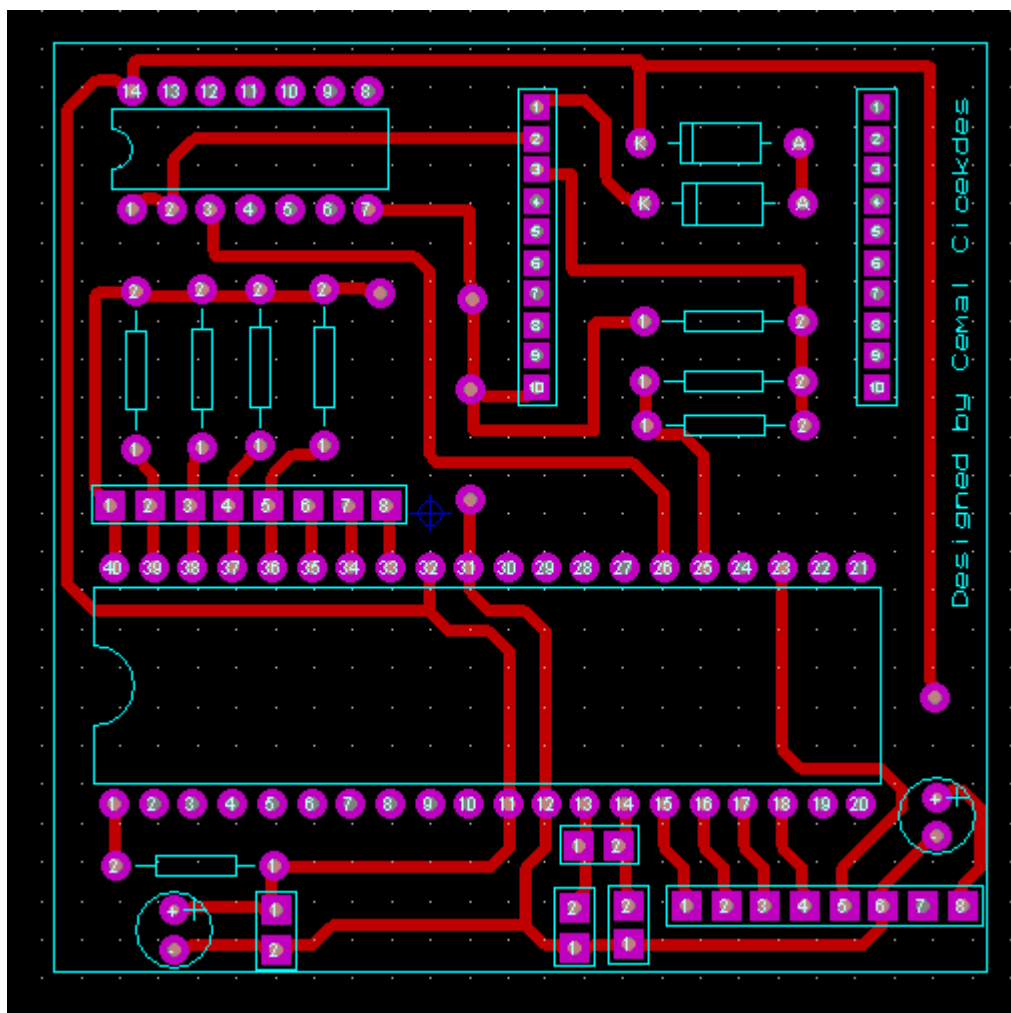


Figure 4.7 PCB design of the remote controller device.



The remote controller device was first constructed on a breadboard and all the hardware design checks were completed before it was drawn on the computer. The PCB design of the remote controller device is shown in Figure 4.7. After it is designed on the computer, it is printed on the acetate and transferred to the copper covered board by ironing. The etched PCB is shown in Figure 4.8.

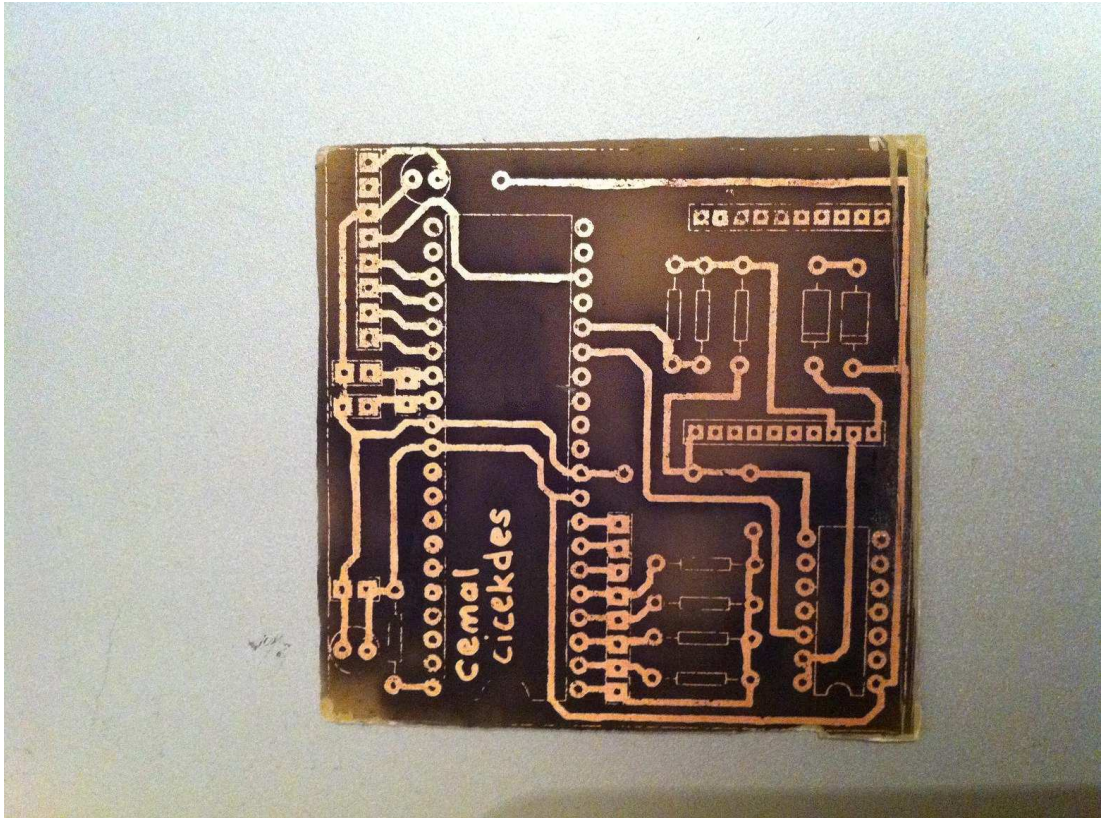


Figure 4.8 PCB of the remote controller device. It is created using toner transfer method and etching.

The remote controller device was placed in a project box that's why the PCB was designed as small as possible in order to fit in the box. The box is useful because it can protect the PCB and its connections, carry the batteries and hold the LCD. The box provides a compact appearance. Figure 4.9 and Figure 4.10 show the open box and closed box appearance of the remote controller device.



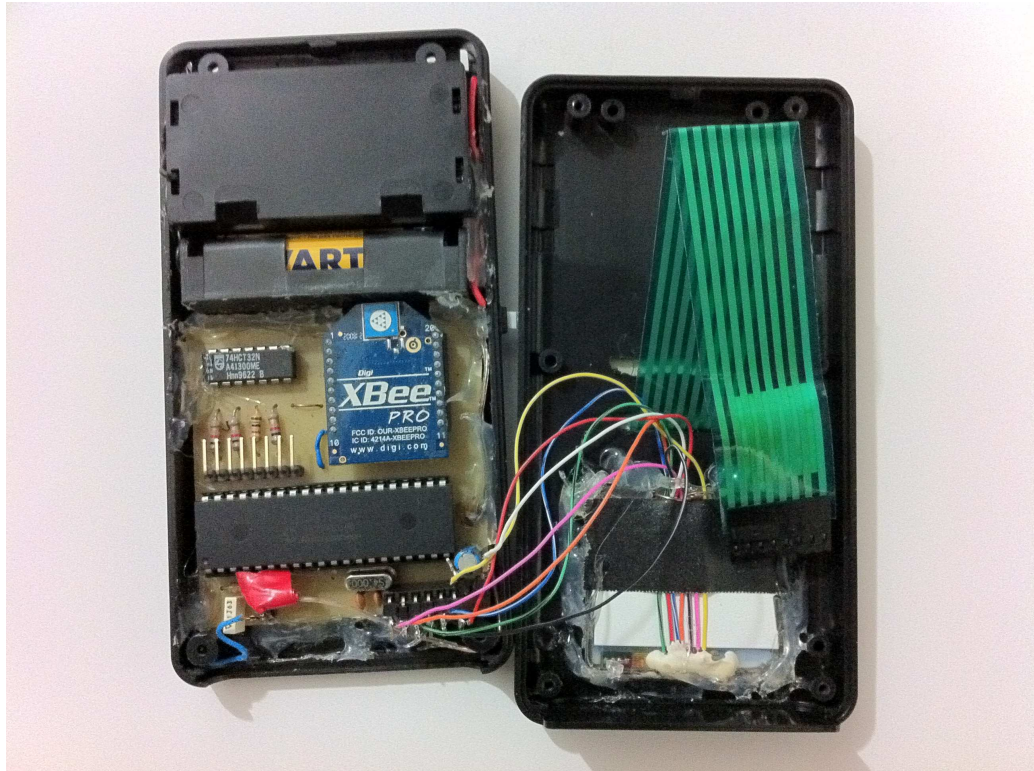


Figure 4.9 Open box view of the remote controller device



Figure 4.10 Closed box view of the remote controller device

## CHAPTER FIVE

### SOFTWARE DESIGN OF THE PROJECT

The software design of the project will be introduced in two separate parts. The data logger software and the remote controller software are both written in C programming language using different compilers.

#### 5.1 Data Logger Software

The data logger is the main part of the project and its software is designed to make it work without any need to remote controller device. The data logger can read the sensor information and store them even if the remote controller is not around. The software is single threaded using interrupts in order to handle real time process requirements.

The software starts with the initialization of the microcontroller and its settings such as clock speed and configuration, SPI speed and configuration, GPIO configuration, ADC initialization and configuration, UART configuration and timer configuration. After the initialization of the microcontroller and its settings, FAT file system is initialized and a log file is opened. The main body of the software will be introduced parts by parts. The initialization sequence and the code are as follows;

```
Initialize(); //initialize all the MCU units and set settings
RTC_Initialize(); // initialize the real time clock settings
fat_initialize(); // Initialize the FAT library.
Init_DS1620(); // initialize the DS1620 temperature sensor
Start_DS1620_Convertion(); // start measuring the temperature
delay_ms(1000); // wait about 1000 ms
for (i = 0; i < 10; i++) // Try to open a log file 10 times until it is successful
{
    sprintf(title,"LOGX%d.txt",i);
    handle = fat_openWrite(title);
```

```

    delay_ms(100);
    if (handle >= 0) break;
}
if (handle < 0) // If we can't open a file, inform and continue without logging
{
    unsigned char j = 0;
    for(j=0;j<=10;j++) // Turn on and off the LED 10 times
    {
        GPIO_BitWrite(GPIO1, 0x0009, 0); // Turn on the LED on the board
        delay_ms(1000);
        GPIO_BitWrite(GPIO1, 0x0009, 1); // Turn off the LED on the board
        delay_ms(1000);
    }
}
if (handle >= 0) // if we opened a file successfully
{
    strcpy(stringBuf,"Date&Time\t\t Temperature\t\t Degree\t\tHumidity
    Light Intensity\tPressure\r\n-----      -----"
    "\t-----\t-----      -----\t\t-----\r\n"); // Prepare the log file header
    stringSize = strlen(stringBuf);
    fat_write(handle,(unsigned char *)stringBuf, stringSize); //Write the header
}

```

This code part is run only once after boot up and then the code continues to an endless while loop. Every operation such as reading the sensor values, responding the remote controller device commands and writing to log file is performed in this loop. Also while the processes are done in this loop, any occurrence of an interrupt will make the program counter leave the loop and do whatever interrupt service routine does. This is required to be able to handle the time based operations and wireless communication because they can occur any time at any point of the endless while loop.



The sensor reading and the data preparation are done by the following code;

```

while(1)
{
    temperature = (int)Read_DS1620_Temp();//Read the temperature from DS1620
    humidity = Read_HH10D_Humidity();//Read the humidity from HH10D
    itoa(temperature,10,TempBuf); //Convert integer temperature value to string
    itoa(humidity,10,HumidityBuf); //Convert integer humidity value to string
    delay_ms(200); //Wait 200ms to decrease the sampling frequency
    Voltage = Read_ADC_Voltage(ADC12_CHANNEL2); // Measure the light
    intensity sensor voltage
    light_intensity = (int)(((Voltage*100*8)/9) + 0.5);// Calculate the light intensity
    sprintf(string_light, "%d", light_intensity); //Copy the value into a buffer
    string_light[3]='\0'; //End the buffer with a NULL character
    if(light_intensity < 100 && light_intensity > 9) //If the value is two digits
    { //Make a format like 024 if the temperature is 24°C
        string_light[2] = string_light[1];
        string_light[1] = string_light[0];
        string_light[0] = '0';
    }
    else if(light_intensity < 10) //If the value is one digits
    { Make a format like 004 if the temperature is 4°C
        string_light[2] = string_light[0];
        string_light[0] = '0';
        string_light[1] = '0';
    }
    Voltage = Read_ADC_Voltage(ADC12_CHANNEL1); Measure the
    acceleration sensor voltage
    Voltage = (int)(Voltage*100); //Cast to an integer value
    Voltage = (float)(Voltage/100);
    temp_voltage = 2*Voltage - 2*acc_at_0_degree;
    degree = (temp_voltage*180.0/2.42) + 0.5; //Calculate the degree

```

```

printf(string_degree, "%d", degree);
string_degree[3]='\0';

if(degree < 100 && degree > 9)
{
    string_degree[2] = string_degree[1];
    string_degree[1] = string_degree[0];
    string_degree[0] = '0';
}
else if(degree < 10)
{
    string_degree[2] = string_degree[0];
    string_degree[0] = '0';
    string_degree[1] = '0';
}
Voltage = Read_ADC_Voltage(ADC12_CHANNEL3); Measure the pressure
sensor voltage
pressure = (2*(Voltage/5.0) - 0.02)/0.09; // pressure = (Vout/Vs - 0.04) / 0.09
if(pressure < 0) //if the pressure is a minus value
    pressure = 123; //Set it to 123 to indicate there was a measuring error
printf(string_pressure, "%1.1f", pressure);
string_pressure[3]='\0';
printf(stringBuf, "#$%s$%s$%s$%s$%s$%s$@@%%",TempBuf,HumidityBuf,
string_degree,string_pressure,string_light); //prepare data format

```

The commands that come from the remote controller device are handled after the sensor reading and data preparation process. If any command is received during sensor reading or data preparation, the command is stored in a buffer in the interrupt service routine and that command is processed as soon as data preparation process is finished. It is very important to get out of interrupt service routines as quick as possible because another interrupt may occur any time and that one will be missed if we spend a lot of time in the routine.

Date&Time	Temperature	Degree	Humidity	Light Intensity	Pressure
17.26.59 26.01.2011	17°C	180°	69	045	0.4
17.27.01 26.01.2011	17°C	179°	69	045	0.4
17.27.03 26.01.2011	17°C	179°	69	048	0.4
17.27.05 26.01.2011	17°C	180°	69	051	0.4
17.27.07 26.01.2011	17°C	179°	69	048	0.4
17.27.09 26.01.2011	17°C	180°	68	052	0.4
17.27.11 26.01.2011	17°C	179°	68	052	0.4
17.27.13 26.01.2011	17°C	180°	68	050	0.4
17.27.15 26.01.2011	17°C	180°	68	051	0.4
17.27.17 26.01.2011	17°C	180°	68	049	0.4
17.27.19 26.01.2011	17°C	180°	68	048	0.4
17.27.21 26.01.2011	17°C	180°	68	050	0.4
17.27.23 26.01.2011	17°C	180°	68	049	0.4
17.27.25 26.01.2011	17°C	180°	68	048	0.4
17.27.27 26.01.2011	17°C	179°	68	051	0.4
17.27.29 26.01.2011	17°C	180°	67	047	0.4
17.27.31 26.01.2011	17°C	179°	67	048	0.4
17.27.33 26.01.2011	17°C	180°	67	048	0.4
17.27.35 26.01.2011	17°C	179°	67	049	0.4
17.27.37 26.01.2011	17°C	179°	67	051	0.4
17.27.39 26.01.2011	17°C	180°	67	051	0.4
17.27.41 26.01.2011	17°C	179°	67	050	0.4
17.27.43 26.01.2011	17°C	180°	67	050	0.4
17.27.45 26.01.2011	17°C	179°	67	052	0.4
17.27.47 26.01.2011	17°C	179°	67	049	0.4
17.27.49 26.01.2011	17°C	180°	67	050	0.4
17.27.51 26.01.2011	17°C	179°	67	048	0.4
17.27.53 26.01.2011	17°C	179°	67	046	0.4
17.27.55 26.01.2011	17°C	180°	67	046	0.4
17.27.57 26.01.2011	17°C	179°	67	043	0.4
17.27.59 26.01.2011	17°C	179°	67	047	0.4
17.28.01 26.01.2011	17°C	180°	67	047	0.5
17.28.03 26.01.2011	17°C	179°	67	049	0.4
17.28.05 26.01.2011	17°C	180°	67	049	0.4
17.28.07 26.01.2011	17°C	179°	66	050	0.4
17.28.09 26.01.2011	17°C	179°	66	048	0.4
17.28.11 26.01.2011	17°C	180°	66	048	0.4
17.28.13 26.01.2011	17°C	179°	66	049	0.4
17.28.15 26.01.2011	17°C	179°	66	051	0.4
17.28.17 26.01.2011	17°C	179°	66	051	0.4
17.28.19 26.01.2011	17°C	179°	66	046	0.4
17.28.21 26.01.2011	17°C	179°	66	035	0.4
17.28.23 26.01.2011	17°C	179°	66	040	0.4
17.28.25 26.01.2011	17°C	180°	67	047	0.4
17.28.27 26.01.2011	17°C	179°	67	047	0.4
17.28.29 26.01.2011	17°C	179°	66	033	0.4
17.28.31 26.01.2011	17°C	173°	66	028	0.4
17.28.33 26.01.2011	17°C	170°	67	029	0.4
17.28.35 26.01.2011	17°C	179°	67	016	0.4
17.28.37 26.01.2011	17°C	179°	67	017	0.4
17.28.39 26.01.2011	17°C	180°	67	018	0.4
17.28.41 26.01.2011	17°C	180°	67	017	0.4
17.28.43 26.01.2011	17°C	179°	67	016	0.4
17.28.45 26.01.2011	17°C	179°	67	020	0.4
17.28.47 26.01.2011	17°C	179°	67	023	0.4
17.28.49 26.01.2011	17°C	179°	67	050	0.4
17.28.51 26.01.2011	17°C	180°	67	046	0.4
17.28.53 26.01.2011	17°C	179°	67	044	0.4
17.28.55 26.01.2011	17°C	173°	67	025	0.4
17.28.57 26.01.2011	17°C	179°	67	027	0.4
17.28.59 26.01.2011	17°C	174°	67	026	0.4
17.29.01 26.01.2011	17°C	176°	67	031	0.4
17.29.03 26.01.2011	17°C	176°	67	035	0.4
17.29.05 26.01.2011	17°C	174°	67	036	0.4
17.29.07 26.01.2011	17°C	174°	67	033	0.4
17.29.09 26.01.2011	17°C	174°	67	034	0.4
17.29.11 26.01.2011	17°C	174°	67	034	0.4
17.29.13 26.01.2011	17°C	174°	67	032	0.4
17.29.15 26.01.2011	17°C	174°	66	031	0.4
17.29.17 26.01.2011	17°C	174°	66	031	0.4
17.29.19 26.01.2011	17°C	174°	66	031	0.4
17.29.21 26.01.2011	17°C	174°	66	031	0.4

Figure 5.1 An example log file that was stored with the designed data logger. The log file shows the time stamps and the values at that time with the sensor identity.

The following code handles the commands coming from the remote controller device.

```

if(is_awesome) //If the wireless module is awake, send sensor data
{
  for(i=0;i<23;i++)//data format is #1212123123123123$@$%
  {
    UART_ByteSend(UART3, (u8 *)&stringBuf[i]);
    /* wait until the data transmission is finished */
    while(!((UART_FlagStatus(UART3)) & UART_TxEmpty));
  }
}
if(command_ready == 1) //If any command is received
{
  if(UART_DataBuf[0] == '#' && UART_DataBuf[15] == '%' &&
  UART_DataBuf[5] == '2' && UART_DataBuf[6] == '0')
  /*Date setting command, format is #16022011133705%*/
  ptime = malloc(sizeof(*ptime)); //allocate some memory
  //Convert char values into integer to calculate the new time setting
  ptime->tm_sec = ((UART_DataBuf[13]-'0')*10 + UART_DataBuf[14]-'0');
  ptime->tm_min = ((UART_DataBuf[11]-'0')*10 + UART_DataBuf[12]-'0');
  ptime->tm_hour = ((UART_DataBuf[9] -'0')*10 + UART_DataBuf[10]-'0');
  ptime->tm_mday = ((UART_DataBuf[1] - '0')*10 + UART_DataBuf[2]-'0');
  ptime->tm_mon = ((UART_DataBuf[3] - '0')*10 + UART_DataBuf[4]-'0');
  ptime->tm_mon--;
  ptime->tm_year = ((UART_DataBuf[5] -'0')*1000 + (UART_DataBuf[6] -
  '0')*100 + (UART_DataBuf[7] -'0')*10 + UART_DataBuf[8]-'0');
  ptime->tm_year -= 1900;
  RTC_Set_Time(ptime); //Set the new time
  free(ptime); //free the allocated memory
}

```



```

    {
        UART_ByteSend(UART3,(u8 *)&stringBuf[i]);
        i++;
    }
    GPIO_BitWrite(GPIO0, 0x000E, 0); //set wireless module wake up pin low
    is_awake = 1; //Wireless module is awake now
}
}

```

Every command and the response to a command have its own format in order to be distinguished from each other. Every command starts with “#” character, ends with “%” character and must be totally 16 characters. The data format starts with “#” character, ends with “@@%” character set and must be totally 23 characters. The command formats received and send from the data logger as well as the data format are as follows;

Data format sent to the remote controller device: **#\$12\$12\$123\$123\$123\$@@%**  
 The numbers between two consecutive “\$” character are temperature, humidity, degree, pressure and light intensity values respectively.

Date setting command format: **#16022011133705%**  
 Date setting command format starts with the day followed by month, year, hour, minute and second. Example format shows the date 16.02.2011 13:37:05

Sampling frequency setting format: **#12@@@@@@@@@@@@%**  
 Sampling frequency has to be only two digits followed by “@” character.

Getting time command: **#1/2/2/2/2/2/2/2/2/2/2/2/2/2/2/2%**  
 Getting time command format consists of full “1/2” characters.

Wake up command: **#~~~~~%**  
 Wake up command format consists of full “~” characters.

Response for wake up command: #\$%\$

The response to a successful wake up command is formatted with full "\$" characters.

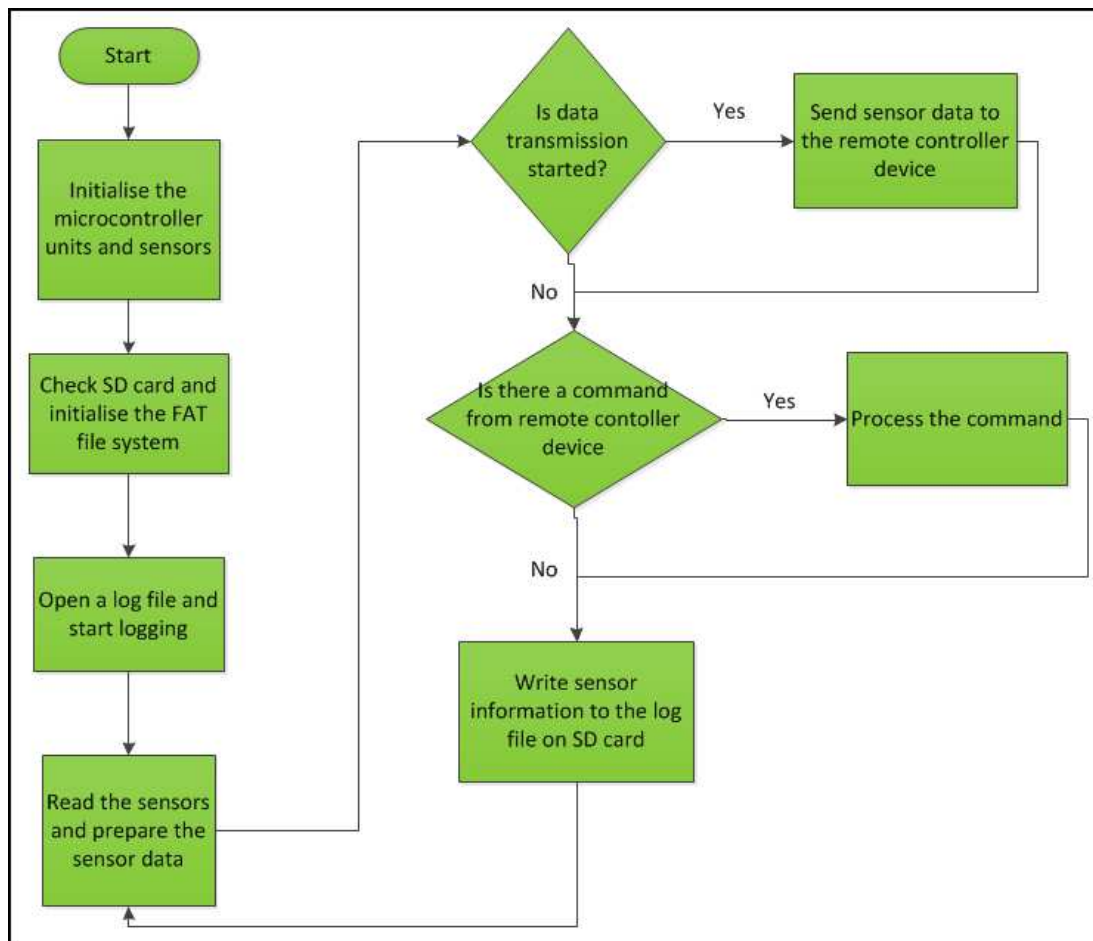


Figure 5.2 Software flowchart of the data logger. Flowchart shows only the main blocks of the software.

## 5.2 Remote Controller Device Software

Remote controller device software is designed to support user interface even if it is a single threaded software. It initializes the microcontroller by setting its required settings for the designed hardware, initializes the LCD and keypad, then enters in an endless while loop. In the while loop, the software reads the keypad to check if the user pressed any key and also displays the sensor data coming from the data logger. The UART communication is handled by the interrupts because it can not be handled by polling the ports.

The main body of the remote controller device software is as follows;

```

void main (void)
{
  unsigned short pressed_key = 0; // define the local variables
  unsigned short keep_awake_counter = 0;
  GIE=1; // set the global interrupt enable bit
  PEIE=1; // set the peripheral interrupt enable bit
  RCIE=1; // set the UART receive interrupt enable bit
  PORTC = 0; // set all port C pins to low
  TRISC = 0; // define all port C pins as output
  Init_Keypad(); // initialize the keypad
  Usart_Init(); // initialize the uart
  InitLCD(); // initialize the LCD
  delay_ms(250); // wait for the LCD to finish the initialization
  strConstCpy(string_buf,lookup[10]); // copy "Designed by" from lookup table
  PutString(string_buf); // write it to the LCD
  CursorXY (0,1); // go the second line of LCD
  strConstCpy(string_buf,lookup[9]); //copy "Cemal Cicekdes" from lookup table
  PutString(string_buf); // write it to the LCD
  delay_ms(2000); // keep whatever is written on the LCD for 2 seconds
  ClearRAM(); // clear all the screen of LCD
  delay_ms(200); // wait to finish the clearing
  PicturePlot(); // plot the DEU EEE picture on the screen
  delay_ms(3000); // keep the picture on the screen for 3 seconds
  ClearRAM(); // clear all the screen of LCD
  while(1) // endless loop
  {
    pressed_key = Read_Keypad(); // read the keypad
    if(pressed_key == 12) //if "*" key is pressed
    {
      OpenMenu(); //open the user interface menu
    }
  }
}

```



```

}
if((data_ready == 1)) //if received data is valid and ready to be parsed
{
  if(GBuf[1] == '$' && GBuf[4] == '$' && GBuf[7] == '$' && GBuf[11]
  == '$' && GBuf[15] == '$' && GBuf[19] == '$' && GBuf[20] == '@'
  && GBuf[21] == '@')
  { //check if this data is for sensor values to be parsed
    CursorXY (0,0); // go to the first line of the LCD screen
    strConstCpy(string_buf,lookup[7]); //copy "Temper.=" from lookup table
    PutString(string_buf); //write it to the LCD screen
    GBuf[4] = 0; //define where it should stop writing (at 5th character)
    PutString(GBuf+2); //start from the 3rd character in the buffer
    strConstCpy(string_buf,lookup[13]); //copy "°C" from lookup table
    PutString(string_buf); // write it to the LCD screen
    CursorXY (0,1); // go to the second line of the LCD screen
    strConstCpy(string_buf,lookup[18]); //copy "Humidity=%" from lookup table
    PutString(string_buf); //write it to the LCD screen
    GBuf[7] = 0; //define where it should stop writing (at 8th character)
    PutString(GBuf+5); //start from the 6th character in the buffer
    CursorXY (0,2); // go to the third line of the LCD screen
    strConstCpy(string_buf,lookup[19]); //copy "Degree=" from lookup table
    PutString(string_buf); //write it to the LCD screen
    GBuf[11] = 0; //define where it should stop writing (at 12th character)
    PutString(GBuf+8); //start from the 9th character in the buffer
    CursorXY (0,3); // go to the forth line of the LCD screen
    strConstCpy(string_buf,lookup[20]); //copy "Pressure=" from lookup table
    PutString(string_buf); //write it to the LCD screen
    GBuf[15] = 0; //define where it should stop writing (at 16th character)
    PutString(GBuf+12); //start from the 13th character in the buffer
    CursorXY (0,4); // go to the fifth line of the LCD screen
    strConstCpy(string_buf,lookup[21]); //copy "Light(lux)=" from lookup table
    PutString(string_buf); //write it to the LCD screen
  }
}

```

```

    GBuf[19] = 0; //define where it should stop writing (at 20th character)
    PutString(GBuf+16); //start from the 17th character in the buffer
}
data_ready = 0; //set data_ready flag to 0 as we processed the data
}
delay_ms(100);
if(wake_up_requested) //if user sent a wake up request to the data logger
{
    keep_awake_counter++; // increase the counter for keeping the module awake
    if(keep_awake_counter >= 250)
    { //if (250*100ms) 25 second has passed, send a command to keep it awake
        for(pressed_key = 0;pressed_key <= 14;pressed_key++)
            MenuBuf[pressed_key] = '~';
        MenuBuf[14] = '\0';
        Usart_Write('#'); //command start character
        UART_PutString(MenuBuf);
        Usart_Write('%'); //command end character
        keep_awake_counter = 0; // clear the counter
    }
}
}
}
}

```

The user interface menu is opened when “#” key is pressed and user can choose one of four options in the menu. The available options are getting the time, setting the time, setting the sampling frequency and sending wake up command. Getting time and sending wake up command require no data entry and the command is sent as soon as the user presses the corresponding number on the keypad.

Selecting the time and date menu will ask the user to enter a valid format of setting. The user interface menu will show the format as “ddmmyyyhhmmss” telling the user to follow the format rule otherwise user can not set a time or date. For

example, if the user tries or accidentally enters a number greater than 59 for seconds setting, it won't be shown or set on the screen. The similar rule is applicable for hour, month and day settings. After the time and date setting is entered in the correct format, it is required to press "\*" key on the keypad in order for the data to be sent. The same procedure is valid for the sampling frequency setting. The user can only enter a sampling frequency value smaller than 100 in two digits.

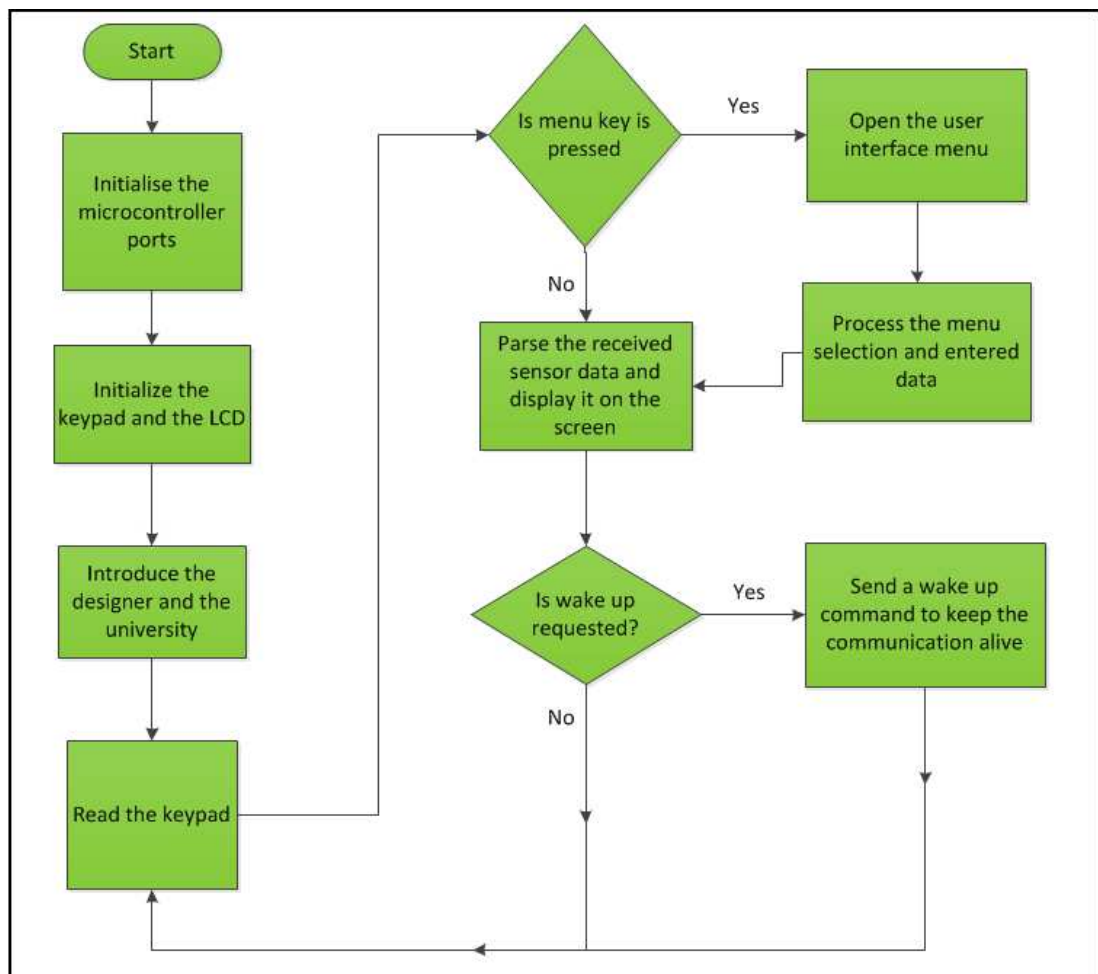


Figure 5.3 Software flowchart of the remote controller device. Flowchart shows only the main blocks of the software.

The user interface menu and the sensor data information screen are shown in Figure 5.4. Time setting and sampling frequency example are shown in Figure 5.5.



Figure 5.4 The user interface menu and the sensor data information screen.



Figure 5.5 Example entries for time and sampling frequency items.

## **CHAPTER SIX**

### **CONCLUSIONS**

The purpose of our study was to design an ARM microcontroller based wireless data logger as well as adding extra advantageous features over the ordinary data loggers. After making some research about data loggers, we noticed that most data loggers have limited, not expandable memory, require a connection to a computer for retrieving and monitoring the stored data and support two or less sensors. The designed data logger overcomes all these limitations and shortcomings.

Our data logger was designed in such a way that it can work as a stand alone without need from any device. The remote controller device was designed as a handheld device for monitoring and controlling purpose. This device removes the need for a computer. The support of five different sensors added another advantage to our data logger. SD storage unit was chosen to make the system storage expandable and FAT file system support made it possible to read the data without any extra software on a computer. The wireless support was one of the other advantages of our data logger. It added a real time wireless data monitoring feature to our data logger and therefore made it possible to control it remotely.

In the hardware design stage, we encountered a problem for handling devices requiring different supply voltages and we overcame this problem by adding voltage dividers, using diodes to drop the voltage and using a logic gate as a voltage buffer. PCBs for both data logger and remote controller device were designed using Proteus Design Suite program on a computer and they were created at home using toner transfer method. Making PCBs at home helped to understand the PCB design better and put extra experience.

Software design was the most critical and important part of the project, to have a robust data logger system. The software reads the sensors, stores their values, controls all units of the system and provides user interface. Both softwares were developed as a single threaded program using C programming language.

## REFERENCES

- Ameur, S., Laghrouche, M., Adane, A. (2001). Monitoring a greenhouse using a microcontroller-based meteorological data-acquisition system. *Renewable Energy*, 24, 19-30. Retrieved April 2010 from ScienceDirect database.
- Benghanem, M. (2009). Measurement of meteorological data based on wireless data acquisition system monitoring. *Applied Energy*, 86, 2651-2660. Retrieved May 2010 from ScienceDirect database.
- Creating PCBs With The Toner Transfer Method*, (n.d). Retrieved November 2010, from <http://www.dr-lex.be/hardware/tonertransfer.html>
- Engelberg, S., Kaminsky, T., Horesh, M. (2007). A USB-Enabled, FLASH-Disk-Based Data Logger. *Instrumentation & Measurement Magazine, IEEE*, 10, 63-66. Retrieved July 2008 from IEEE Xplore Digital Library.
- Fairchild Semiconductor, Co., (April, 2010). *LM7805 Voltage Regulator Datasheet*. Retrieved June 2010, from <http://www.fairchildsemi.com/ds/LM/LM7805.pdf>
- Freescale Semiconductor, Inc, (November, 2009). *MMA1260KEG Low G Micromachined Accelerometer Technical Data*. Retrieved December 2008, from [http://cache.freescale.com/files/sensors/doc/data\\_sheet/MMA1260KEG.pdf](http://cache.freescale.com/files/sensors/doc/data_sheet/MMA1260KEG.pdf)
- Freescale Semiconductor, Inc, (September, 2005). *MPVZ5010G Integrated Silicon Pressure Sensor Technical Data*. Retrieved March 2009, from [http://cache.freescale.com/files/sensors/doc/data\\_sheet/MPVZ5010G.pdf](http://cache.freescale.com/files/sensors/doc/data_sheet/MPVZ5010G.pdf)
- Hope Microelectronics Co.,Ltd, (December 14, 2006). *HH10D Humidity Sensor Datasheet*. Retrieved January 2009, from <http://www.futurlec.com/HH10D.shtml>

Ibrahim, D. (2010). Microcontroller Systems. In *SD Card Projects Using the PIC Microcontroller* (1-40). Oxford: Newnes.

Maxim Integrated Products, Inc, (n.d). *Digital Thermometer and Thermostat Datasheet*. Retrieved March 2008, from <http://pdfserv.maxim-ic.com/en/ds/DS1620.pdf>

Maxstream, Inc, (May 2007). *Xbee Pro OEM RF Modules User Manual*. Retrieved August 2008, from <http://www.sparkfun.com/datasheets/Wireless/Zigbee/XBee-Datasheet.pdf>

Microchip Technology Inc, (2001). *PIC16F87x Datasheet*. Retrieved April 2008, from <http://ww1.microchip.com/downloads/en/devicedoc/30292c.pdf>

Olimex Ltd, (June, 2010). *STR-P711 Development Board Users Manual*. Retrieved July 2010, from <http://www.olimex.com/dev/pdf/ARM/ST/STR-P711.pdf>

Philips Semiconductors, (April, 1999). *PCD8544 48x84 Pixels Matrix LCD Controller/Driver Datasheet*. Retrieved June 2008, from <http://www.datasheetcatalog.org/datasheet/philips/PCD8544U.pdf>

Senba Optical Electronic Co., Ltd, (July 2005). *LLS05-A Linear Light Sensor Technical Data*. Retrieved May 2009, from [http://www.futurlec.com/Light\\_Sensor.shtml](http://www.futurlec.com/Light_Sensor.shtml)

ST Microelectronics, Ltd, (July, 2007). *STR71xF Microcontroller Family Reference Manual*. Retrieved April 2008, from [http://www.st.com/internet/com/TECHNICAL\\_RESOURCES/TECHNICAL\\_LITERATURE/REFERENCE\\_MANUAL/CD00164787.pdf](http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/REFERENCE_MANUAL/CD00164787.pdf)