**AN-NAJAH NATIONAL UNIVERSITY**

FACULTY OF ENGINEERING

ELECTRICAL ENGINEERING DEPARTMENT

**ROOM OCCUPANCY**

PREPARED BY:

Abeer zidan 10508966

Fatima Al-sayyied 10508963

SUPERVISED BY:

Dr. Jamal kharosheh

A graduation project submitted to the electrical engineering department in partial fulfillment of the requirements for the degree of B.Sc. in Electrical Engineering.

**ABSTRACT**

**This device uses two ultrasonic sensors to determine if a person entered or exited a room, thereby determining the occupancy value over time. The microcontroller tabulates the data and continuously has the current occupancy value on hand. The transmitter takes this information and wirelessly sends it to a receiver, which can then be viewed on a display through a RS-232/MAX-232 conversion protocol. Each component discussed thoroughly, as well as the corresponding tests and measurements necessary for a functional device. Our design decisionsways to improve the product, as well as costs, are also well documented.**

**TABLE OF CONTENTS**

**1. INTRODUCTION ................................................................................................................................**

**1.1 Purpose ...............................................................................................................................**

**1.2 Specifications ................................................................................................................................**

**1.3 Subprojects**

**................................................................................................................................**

**2. DESIGN PROCEDURE ..............................................................................................................................**

**2.1 Ultrasonic Sensors ................................................................................................................................**

**2.2 Microcontroller ................................................................................................................................**

**2.3 Wireless Transmitter/Receiver ................................................................................................................................**

**2.4 MAX-232 ................................................................................................................................**

**2.5 RS-232 ....................................................................................................................**

**3.design details**

**................................................................................................................................**

**3.1 Ultrasonic sensors ................................................................................................................................**

**3.2 Microcontroller ................................................................................................................................**

**3.3 Wireless Transmitter ...............................................................................................................................**

**3.5 MAX-232 ...............................................................................................................................**

**3.6 RS-232 ...............................................................................................................................**

**4. CONCLUSIONS ...............................................................................................**

**5.REFERENCES.................................................................................................**

**6. Appendix……………………………………………………………………**

**1. INTRODUCTION**

**1.1 Purpose**

**The purpose of our project is to keep track of the number of people in a targeted room. This value will be displayed on a wireless device depending on which frequency it is operating at. Our design would be useful in many applications. For example, one can keep track of how many people are on a bus when on a field trip rather than taking a head count. Also, firefighters can have a display and quickly find someone in a trapped room that is on fire. Another use would be for the mailman to determine if anyone is home without wasting time and knocking on the door.**

**1.2 Specifications**

**Even though our project was functional in the end, there were still some specifications that had to be met. We could not walk too fast through the sensor system, or else the sensors would not correctly increment or decrement the count. We had to adjust our speed to account for this. We calculated our total time to walk through the sensor system to be about 3s. The distance between each of the sensors was also important. The farther they were apart, the better our results. There was a minimum distance between the sensors for them to function, which was 1ft and 2in long. Because the sensors transmitted at a 22° angle in all directions, the waves would have interfered, thereby rendering sensor readings useless if they were placed any closer. A final specification for the sensors is that the range of them is between 3 in and 4m. They would not detect someone if the person walked less than 3 inches from the sensors and since the distance from each side of the door is less than 4m, the upper part of the range is not a problem. As for the wireless system, the maximum range it can function is about 1000ft. Any further would have led to the count value not being received at the end of the display.**

**The focus of our project was to create a functional system where the occupancy value of a room could be passed along, from the room under test to the master control room. We used two ultrasonic sensors to initially determine whether or not an enter or exit pattern occurred. The microcontroller processed this data to determine the occupancy value. It was then sent to the wireless system, composed of a transmitter and receiver, and through a series of conversions with the MAX-232 and RS-232 protocol, was displayed on a computer with the use of LabView.**

**1.3 Subprojects**

**Door Input: This acted as the input to the sensors. This initial input, typically people entering or exiting, accounted for variations of entering or exiting a room. The sensor placement plays a role in how this should be interpreted.**

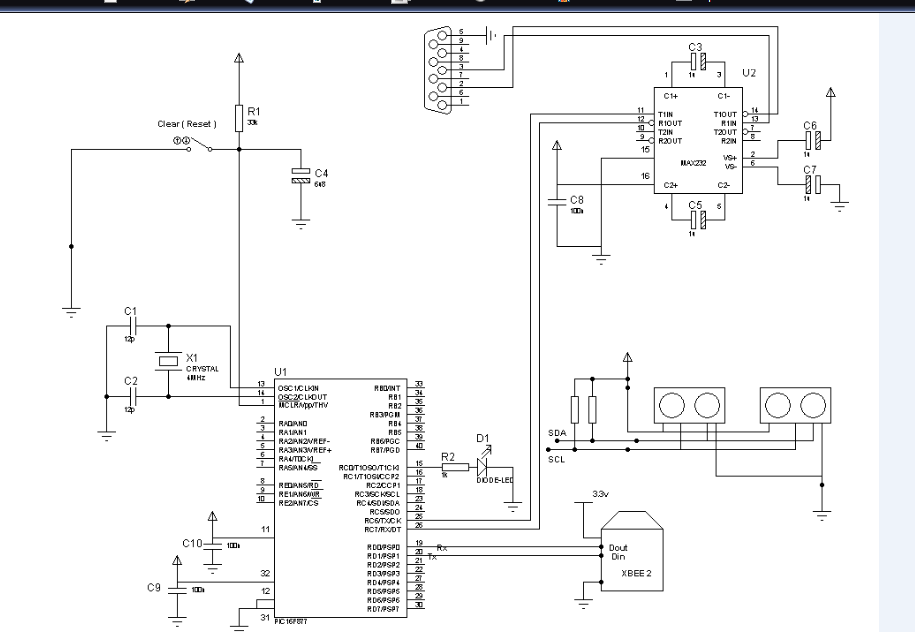
**Sensor System: This consisted of two adjacent ultrasonic sensors. The sensors sent out waves and if a signal arrived back to its respective sensor faster than normal, then the appropriate sensor detected a person. We used two sensors in order to detect the direction the person was going. If the sensor closer to the door had gone off first, then someone entered. If the sensor that was further in the room went off first, then someone exited.**

**PICs: The system had two PICs. One simply acted as an oscillating input for the sensors. It mimicked a 1Hz, 13% duty cycle signal that signaled the sensors to send out ultrasonic waves. The second PIC had the logic setup which determined the current count of people in the room. The sensor system was its input and the PIC used a state machine setup to determine the room occupancy value. Once this value had been calculated, it passed it into a wireless transmitter. 2 Transmitter/Receiver System: This portion of our design wirelessly transferred the occupancy value from the first PIC to the MAX-232.**

**MAX-232/RS-232: The MAX-232 converted the signal that came from the receiver into a signal that the RS-232 could read. The RS-232 then fed this signal into the display.**

**Display: We stored the room’s occupancy value in a monitoring station. To connect this system to the RS-232, we displayed this data visually through the front panel of LabVIEW. The block diagram was utilized to properly pass the information onto this display.**

**Figure 1 is the schematic of the entire project.**

****

**Figure1**

**2. DESIGN PROCEDURE**

**2.1 Ultrasonic Sensors**

**Two sensors were placed on the same wall adjacent to the entryway; however, the sensors were spaced slightly apart from each other. The sensors served as inputs in determining the room occupancy estimation logic. The sensors outputted a digital value of either high or low (i.e. source voltage or ground), but they were analog due to their changing pulse width. The closer an object was, the less time a sensor would stay high. Knowing this, we found the sensor’s pulse width when it detected the wall across from it, and compared all pulse widths relative to this. If the sensor’s pulse width was smaller, then the sensor had detected someone. The echo outputs capturing these pulse widths fed directly into the microcontroller, which used this data to determine whether someone had entered or exited a room.**

**2.2 Microcontroller**

**The sensors served as inputs into our engineering system. This system started with the PIC 16F877A microcontroller, which took their values and through assembly programming, created and adjusted a count of the number of people in a room. This process was done by making a state machine in order to know previous input values and to handle the frequency issues involved in checking the sensors’ outputs. Each bit of the count was then passed serially into the transmitter. Various tools were needed in order for this to work. First of all, knowing assembly was essential and was achieved through tutorials and datasheets [1][2]. After programming the PIC, it was tested with LEDs, and the oscilloscope when trying to work in congruence with parts external to it. It was powered by a 5V supply and had a 4MHz oscillator as a clock. There were four inputs to help determine and adjust the count. There were the two sensors, but there was also a reset switch and a mode select input. The default mode was the room occupancy indicator, which tracked the current amount of people in a room. The secondary mode calculated the total amount of people who had entered a room over time. The reset sent the count value back to zero. Once all functionality of the microcontroller worked and the proper count was displayed**

**on the LEDs, it was passed into the wireless transmitter.**

**2.3 Wireless Transmitter/Receiver**

**To accommodate for the link between the calculated occupancy value of the microcontroller to ultimately the display on a computer, we decided to add versatility to our project by setting up a wireless system. We used Linx’s TXM-900-HP3-PPS transmitter and RXM-900-HP3-PPS receiver, both responsible for the transmission of data in the 902-927MHz range. Although a wired transmission could**

**have been used, a wireless component added much practicality to our project, since the distance between the monitoring station and the room under test would not be limited by the length of the wire. Due to its availability in the lab, data transmission within a range of 1000ft (which was more than enough for the scope of our project), the power down capability, and being able to read the received signal’s strength, it proved to be an ideal choice. However, because this type is not available , we searched the different types of wireless Tx \Rx and we decided to use Xbee Tx\Rx (ZigBee series 2) This is the very popular 2.4GHz XBee module from Digi (formally Maxstream). These modules take the** [**802.15.4**](http://en.wikipedia.org/wiki/IEEE_802.15.4) **stack (the basis for Zigbee) and wrap it into a simple to use serial command set. These modules allow a very reliable and simple communication between microcontrollers, computers, systems, really anything with a serial port! Point to point and multi-point networks are supported.**

**2.4 MAX-232**

**We needed some conversion to interface our TTL logic with the RS-232 and computer. Without this chip, our project would not have functioned and the count value would not have been passed to the 4 computer. This chip alters the voltage levels since the PC outputs 12V, while 5V is desired for TTL logic. This component is the bridge to linking the PIC, RS-232, wireless system, and the computer.**

**2.5 RS-232**

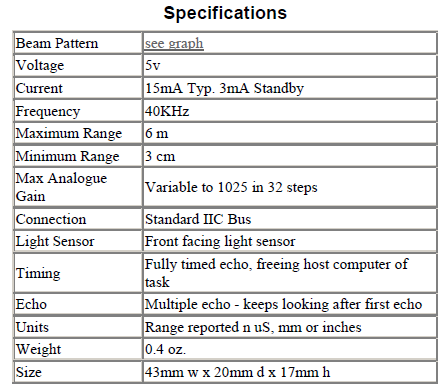
**We used the 9 pin male RS-232 to transfer the data onto the computer through a serial port labeled COM1. To simplify our design, only 3 pins were used from the 9 pins because the other pins were unnecessary and not required. In order to debug our circuit, we used a program called HyperTerminal where we could display our value.**

**3. DESIGN DETAILS**

**3.1 Ultrasonic Sensors**

****

**We chose to use Devantech’s SRF08 Ultrasonic Sensor as a device to determine whether someone has entered or exited a room. An ultrasonic sensor uses wave propagation to determine how far an object is from itself. This is calculated by the time it takes for a wave to bounce back to the sensor it was originally transmitted from. There were other types of sensors we could have used in this design, such as infrared and lasers, but the most accessible and efficient for our system was the aforementioned sensor. An alternate design and possible improvement to our project would utilize an infrared sensor complementary to our sensor system, as this determines whether a person or inanimate object crossed a door’s threshold. This Devantech high performance ultrasonic range finder is compact and measures an amazingly wide range from 3cm to 6m. The SRF08 interfaces to your microcontroller via the industry standard I2C bus. This ranger is perfect for your robot, or any other projects requiring accurate ranging information. There is even a built-in light sensor on the front of the module.**

****

****

**Dimensions**

****

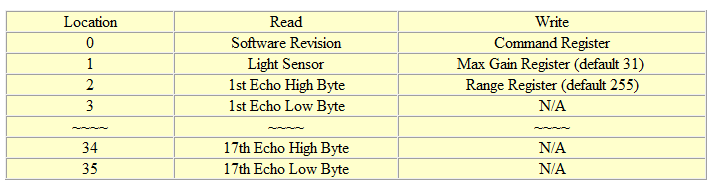
**Connections**

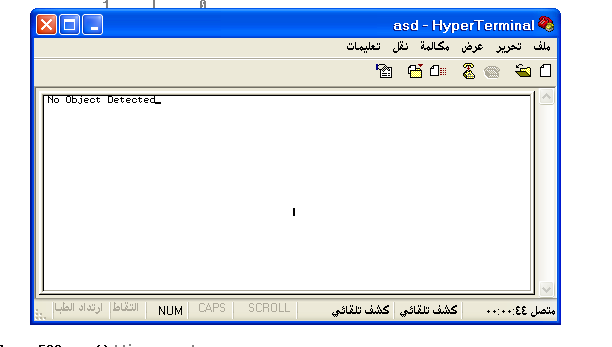
**The "Do Not Connect" pin should be left unconnected. It is actually the CPU MCLR line and is used once only in our workshop to program the PIC16F872 on-board after assembly, and has an internal pull-up resistor. The SCL and SDA lines should each have a pull-up resistor to +5v somewhere on the I2C bus. You only need one pair of resistors, not a pair for every module. They are normally located with the bus master rather than the slaves. The SRF08 is always a slave - never a bus master. If you need them, I recommend 1.8k resistors. Some modules such as the OOPic already have pull-up resistors and you do not need to add any more.**

****

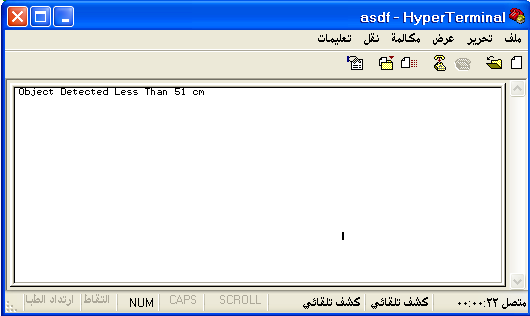
**Registers**

**The SRF08 appears as a set of 36 registers.**

****

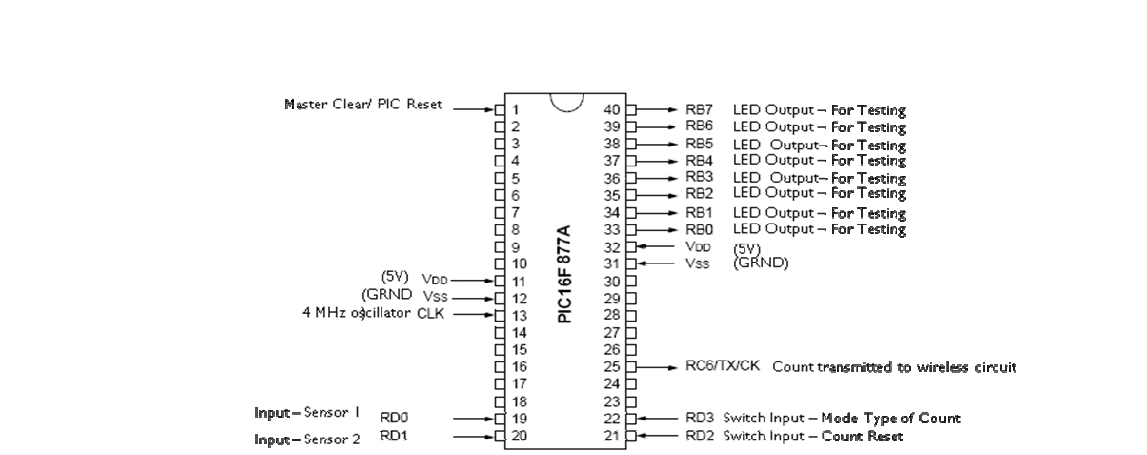
****

****

****

**3.2 Microcontroller**

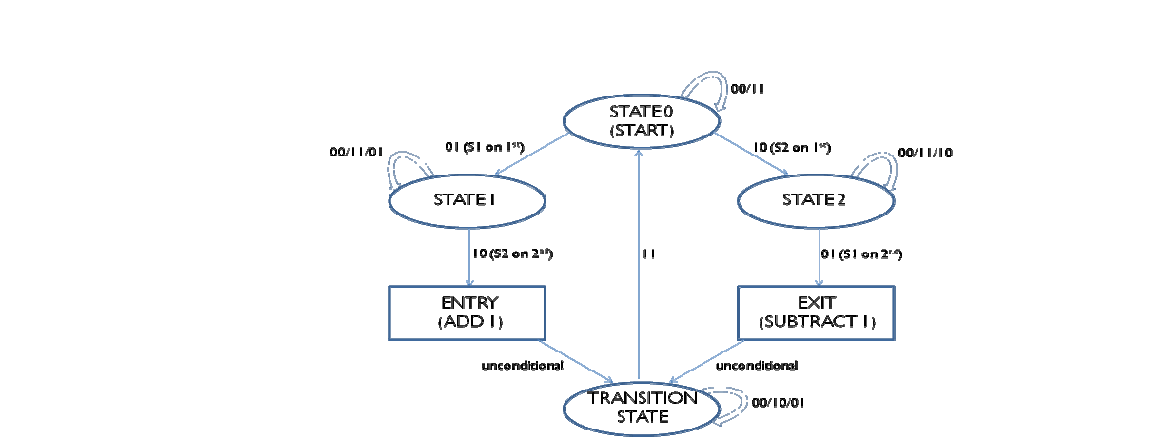
**The decision logic for the project was implemented with a microcontroller, the PIC16F877A. The microcontroller received external inputs and used them to compute the amount of people in the room. It would then output this value to be sent to the transmitter through hard wiring . We had the option of coding the microcontroller’s functionality in C or in assembly language, and we chose to do it in assembly due to run-time efficiency. When writing the logic for the PIC, a relationship between incoming data and the current room occupancy had to be established. Having one sensor detect a person was simple enough, but we needed to know direction to determine whether a person was entering or exiting. So, we decided to place two sensors far enough away from each other so that each sensor had a boundary where only it was on while the other was off. With this sensor system, we could determine if a person had entered a room based on which sensor detected someone first. To do this, one must know past values of the sensors.**

****

**Figure 3**

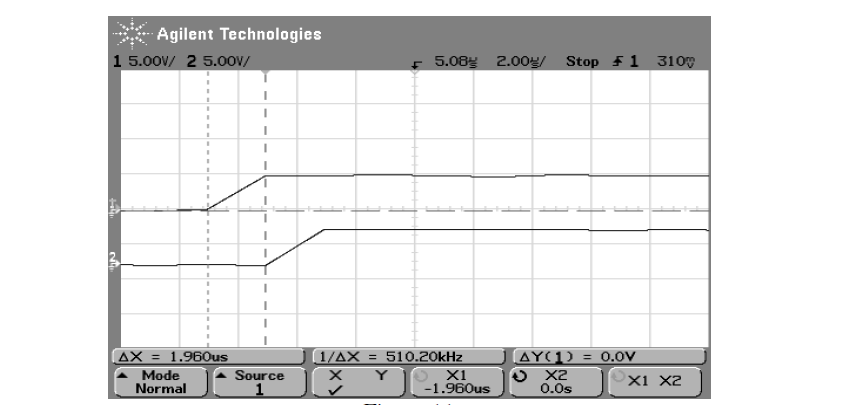
**Originally, we thought of storing previous values in registers or perhaps even D-Latches. During the initial PIC programming, there was a delay between measurements so that they would not record too fast for the person entering the system. Once the data input from the sensors was slowed down enough to achieve slower information into the PIC, the data was stored in memory through registers. The least significant bit would be the most recent reading from the appropriate sensor. From there, the code checked if the second to least significant bit in each register was opposite of the other register and if these lows or highs flipped in the least significant bit (i.e. if sensor one’s latest inputs were either 01 or 10 and sensor two was the opposite). If this was the case, then the count adjusted accordingly, by incrementing or decrementing. In all other cases, the count would remain the same. This concept worked in theory, but not in practice. When simulating with MPLab Sim (a program that allows for assembly debugging), the code worked. It also worked with ideal inputs when we tested it with switches. However, the count did not adjust when the sensors were used as inputs. This was due to two factors. First of all, the waveform property of the sensor output meant that the sensor had to be read at a uniquely appropriate time with the current microcontroller logic. Additionally, the delay was too long and not refined enough to read so precisely. Without being able to achieve the appropriate delay, we could not use the microcontroller logic and it had to be revised.**

**A state machine was a simple, yet effective way of making the PIC logic less sensitive to the exact time the input was being read. In a state machine, we could accommodate for the waveform properties of the sensor, while still being able to hold the knowledge of previous actions recorded by the sensors. An extensive state machine could be made, dissecting the exact properties of each sensor in given regions; however, this would not work with the properties of the pulse width, the sensors output. Because of the sensor’s properties, states where both sensors were on or both were off had to be ignored, because they were such common occurrences. In other words, the logic behind the state machine would be similar to what was created in the original register version of the microcontroller logic. If an entry or exit process was started, the state machine adjusted its state and was ready to receive the next input that would either increment or decrement the count. The final step must be to reach a transition state before going back to the beginning of the state machine in this infinite loop. The transition state made sure the next process in the state machine did not get locked into the opposite action of what just occurred (e.g. it will not initiate exit logic if an entry has just taken place). This concept again worked in simulation. It also worked with ideal inputs and was a lot more reliable and less timing sensitive to the data it received from the switches. It produced a result when placed in the sensors’ system, but it had some flaws that needed to be accounted for**

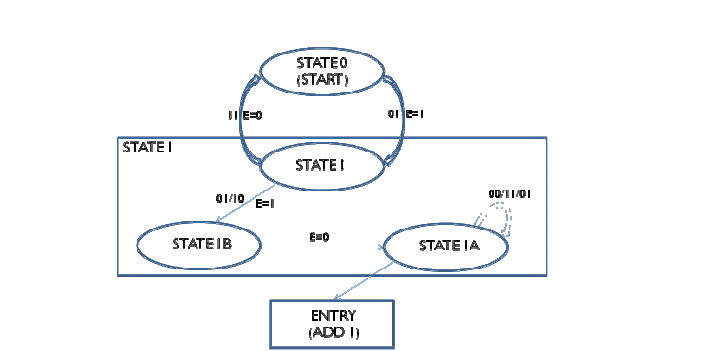
****

**Figure 4**

**One problem with the sensor system was the trigger input of the sensors. It was too fast and resulted in way too many fluctuations in the value of the count before it finally settled. To adjust for this, we tested various frequencies and found a 1 Hz trigger input to yield optimal results. In fact, it practically was the correct system setup. Nevertheless, there was an issue with decrementing the current room occupancy after an exit had taken place. Exiting would unconditionally increment the count rather than subtract one from it. Additionally, if one stood at the second sensor, it would decrement after every clock cycle. This loop was created by a discrepancy between the sensors in timing. As you can see by looking at Figure 10, the sensors’ pulse widths are misaligned by 1.96 microseconds. This causes issues with adding. So, we adjusted State 1 from the state machine shown in Figure 6. This modified state checks to make sure the current count adjustment actually is supposed to be in state 1 rather than state 0. In short, are issues with the sensors’ inputs causing errors? Once a count adjustment enters state 1, the modified version checks to make sure it is still in state 1, since the time delay should be accounted for by then. If the count adjustment genuinely is preparing for an entry then it continues to state 1 in Figure 10. Otherwise, the count adjustment will be reset and start again in state 0.**

****

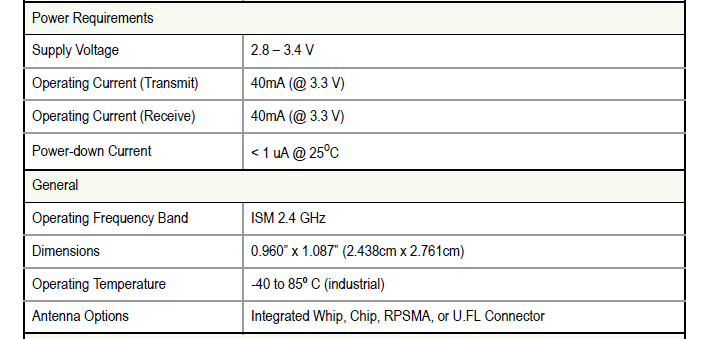
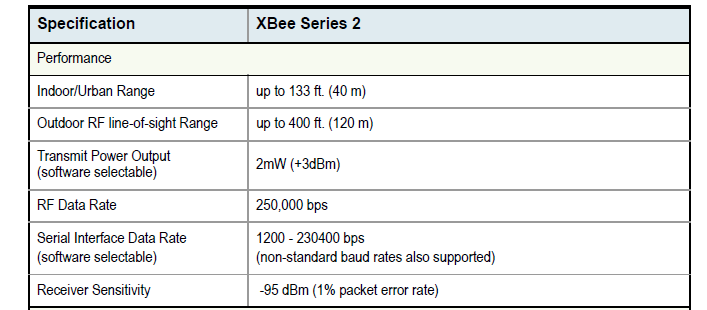
**Figure 5**

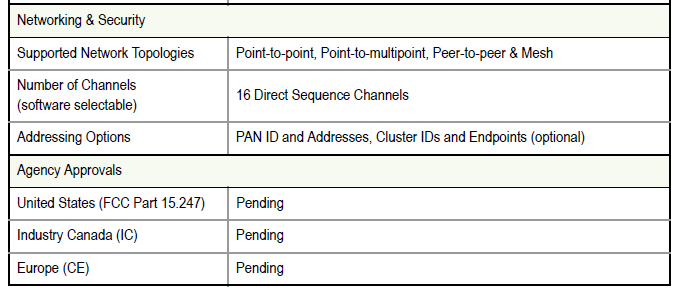
****

**Figure 6**

**3.3 Wireless Transmitter**

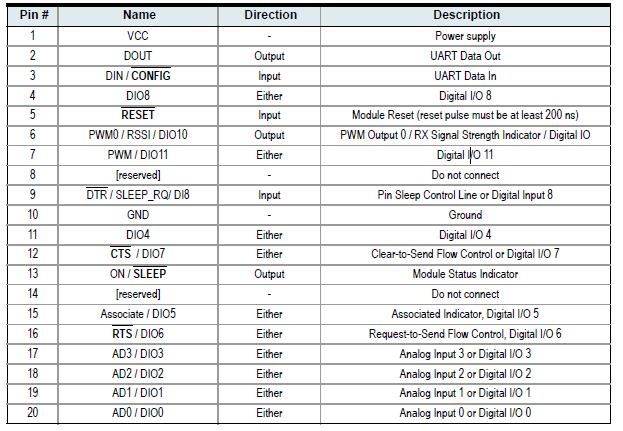
**We used the Xbee (ZigBee S2) transmitter in order to take the output from the microcontroller and to send it to the receiver. This is the very popular 2.4GHz XBee module from Digi (formally Maxstream). These modules take the** [**802.15.4**](http://en.wikipedia.org/wiki/IEEE_802.15.4) **stack (the basis for Zigbee) and wrap it into a simple to use serial command set. These modules allow a very reliable and simple communication between microcontrollers, computers, systems, really anything with a serial port! Point to point and multi-point networks are supported.**

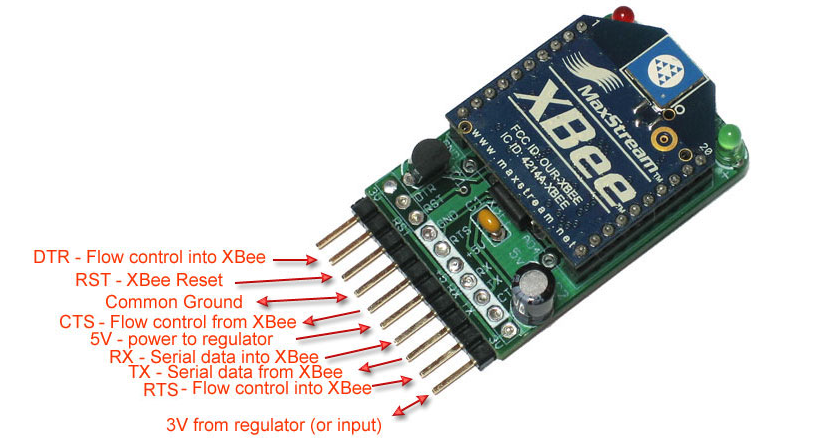
**Features: **

****

**Antenna Options: The ranges specified are typical when using the integrated Whip (1.5 dBi) and Dipole (2.1 dBi) antennas. The Chip antenna option provides advantages in its form factor; however, it typically yields shorter range than the Whip and Dipole antenna options when transmitting outdoors.**

****

****

****

* **3V pin - this is either an input power pin (if 5V is not provided) or an output from the 250mA regulator if 5V is provided**
* **DTR - "Data terminal ready" this is a flow control pin used to tell the XBee that the microcontroller or computer host is ready to communicate.**
* **RST - this pin can be used to reset the XBee. By default it is pulled high by the 10K resistor under the module. To reset, pull this pin low.'**
* **Ground - common ground for power and signal**
* **CTS - "Clear to Send" this is a flow control pin that can be used to determine if there is data in the XBee input buffer ready to be read**
* **5V - this is the power input pin into the 3.3V regulator. Provide up to 6V that will be linearly converted into 3.3V**
* **RX - This is the XBee's serial recieve pin. Serial data is sent on this pin into the XBee to be transmitted wirelessly**
* **TX - This it the XBee's serial transmit pin. Serial data is sent on this pin out of the XBee, after it has been transmitted wirelessly from another module**
* **0 RTS - "Ready to Send" this is a flow control pin that can be used to tell the XBee to signal that the computer or microcontroller needs a break from reading serial data.**
* **see pin #1**

**The DTR, RTS, RESET and RX pins (going into the XBee) pass through a level converter chip that brings the levels to 3.3V. You can use pretty much anywhere between 2.7 to 5.5V data to communicate with the XBee. The breakout pins on the bottom of the board are not level shifted and you should try to keep data going directly into the XBee pins under 3.3V**

**Design Notes:**

• Minimum connections: VCC, GND, DOUT & DIN

• Minimum connections to support firmware upgrades: VCC, GND, DIN, DOUT, RTS & DTR

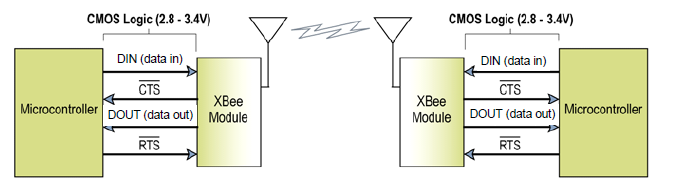
• Signal Direction is specified with respect to the module

• Module includes a 30k Ohm resistor attached to RESET

• Several of the input pull-ups can be configured using the PR command

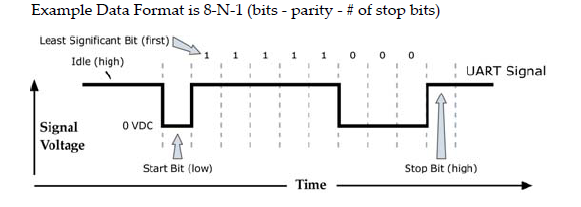
• Unused pins should be left disconnected

**\*\***Devices that have a UART interface can connect directly to the pins of the RF module as shown in the figure below.

****

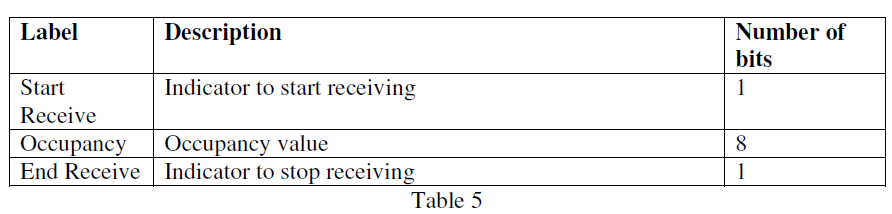
Serial Data

**Data enters the module UART through the DIN (pin 3) as an asynchronous serial signal. The signal should idle high when no data is being transmitted. Each data byte consists of a start bit (low), 8 data bits (least significant bit first) and a stop bit (high). The following figure illustrates the serial bit pattern of data passing through the module.**

****

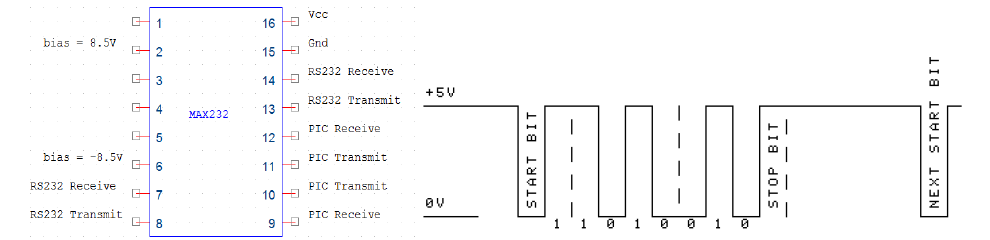
**The module UART performs tasks, such as timing and parity checking, that are needed for data communications. Serial communications depend on the two UARTs to be configured with compatible settings (baud rate, parity, start bits, stop bits, data bits).**

**Continuing the previous example, Table 5 describes the data and receiver configurations.**

****

**3.5 MAX-232**

**The MAX-232 chip was used to convert the higher voltage levels from the computer to lower voltages for TTL logic. The computer outputs 12V and this chip converted it to about 5V. The pin layout is shown in Figure 9. Our design had our PIC output 8 bits of data into the transmitter where the output of the receiver connected to the input of the MAX-232. The output from the MAX-232 was sent into the RS-232 which connected to the computer. We used many of the pins from Figure 9 for debugging where we interfaced the PIC with the RS-232. This will be discussed later in the testing section of the report. Also shown later in the report is the voltage levels on the oscilloscope when obtaining data from the MAX-232 and RS-232. Figure 10 shows what a typical set of data looks like. Our data consisted of a start bit, the 8 bit count \ value (1 byte), and 1 stop bit. There could be a parity bit for error checking and more than 1 additional stop bit, but we chose not to implement it that way for simplicity.**

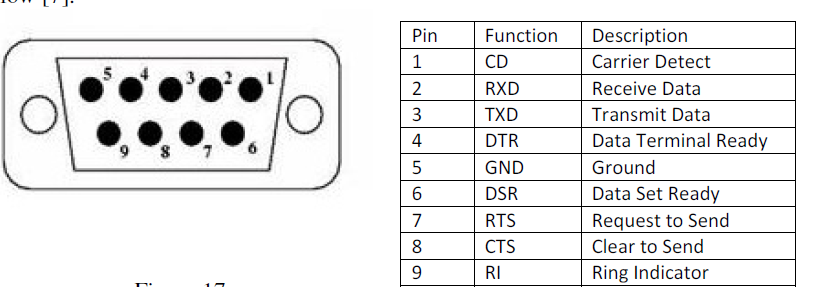
****

**Figure 9 figure10**

**We can see that the voltage levels are clearly at 5V for this data line. This is because of the MAX-232 which converts the voltage from 12V. Also, this is the correct data being sent as opposed to the data coming from the PC . That data is actually inverted which will be shown later.**

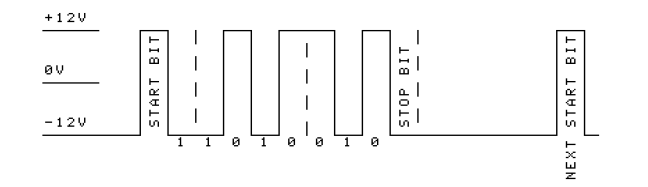
**3.6 RS-232**

**We used an RS-232 serial cable to get our data onto the computer. The pin layout is shown in Figure 11 below .**

****

**Figure 11**

**The only pins we used were pins 2, 3, and 5. The rest of the pins are meant for handshaking purposes only. Handshaking is a process where two devices begin communications between one another . We disabled handshaking because it was not allowed since HyperTerminal forbids it and we have no control over Hyper Terminal. We used HyperTerminal for debugging purposes and to display the count. There was no way to control how this program functioned by writing code, which is why we did not have any handshaking protocols. For our final design, we only used pins 2 and 5, receive data and ground respectively, because it was only one way communication. However, for debugging, we also used pin 3 to transmit data to the PIC. Figure 12 represents a typical signal coming from the PC. Comparing this to Figure 10, one can see the voltage levels are higher at 12V and the bits are inverted. The MAX-232 actually outputs the real data and the PC outputs the inverted data.**

****

**Figure 12**

3.7Conclusions

We were satisfied by the correct functionalities of each of our project’s components. The overall goal was to get each component working as effectively as possible, and to combine them for a working

system.

**REFERENCES**

**[1] M. Stracey, “PIC Tutorial,” [Online Document], 3 August 2002, [cited 4 May 2009], Available HTTP: http://www.mstracey.btinternet.co.uk/pictutorial/picmain.htm**

**[2] Microchip Technology, “Data Sheet PIC16F87XA,” [Online Document], 28 October 2003, [cited 4 May 2009], Available HTTP: http://ww1.microchip.com/downloads/en/DeviceDoc/39582b.pdf**

**[3] Linx Technologies Technical Staff, *HP3 Series Transmitter Module Data Guide Description*, Linx Technologies Inc., 2008.**

**[4] Linx Technologies Technical Staff, *HP3 Series Receiver Module Data Guide Description*, Linx Technologies Inc., 2008.**

**[5] N. Goodwin, “PIC Tutorial Seven – RS232,” [Online Document], 6 May 2006, [cited 4 May 2009], Available HTTP: http://www.winpicprog.co.uk/pic\_tutorial7.htm**

**[6] Texas Instruments, ”Data Sheet MAX232”, [Online Document],15 Sept 2004, [cited 4 May 2009], Available HTTP: http://www.datasheetcatalog.org/datasheet/texasinstruments/max232.pdf**

**[7] Christopher E. Strangio,” The RS232 Standard”,[Online Document],10 Sept 2006, [cited 4 May 2009], Available HTTP: http://www.camiresearch.com/Data\_Com\_Basics/RS232\_standard.html**

**[8] Webopedia,” What is handshaking?”,[Online Document], 4 May 2006, [cited May 2009],**

**Available HTTP:** [**http://webopedia.com/TERM/H/handshaking.html**](http://webopedia.com/TERM/H/handshaking.html)

**Appendix:**

**Code use to change the address**

**#include "Source.h"**

**void main()**

**{**

**byte iDist;**

**int Counter=0;**

**int16 distance;**

**char distance\_L, distance\_H, luz;**

**setup\_adc\_ports(NO\_ANALOGS);**

**setup\_adc(ADC\_OFF);**

**setup\_psp(PSP\_DISABLED);**

**setup\_timer\_0(RTCC\_INTERNAL|RTCC\_DIV\_1);**

**setup\_timer\_1(T1\_DISABLED);**

**setup\_timer\_2(T2\_DISABLED,0,1);**

**i2c\_start();**

**i2c\_write(0xE0);**

**i2c\_write(0x00);**

**i2c\_write(0xA0);**

**i2c\_stop();**

**i2c\_start();**

**i2c\_write(0xE0);**

**i2c\_write(0x00);**

**i2c\_write(0xAA);**

**i2c\_stop();**

**i2c\_start();**

**i2c\_write(0xE0);**

**i2c\_write(0x00);**

**i2c\_write(0xA5);**

**i2c\_stop();**

**i2c\_start();**

**i2c\_write(0xE0);**

**i2c\_write(0x00);**

**i2c\_write(0xE2);**

**i2c\_stop();**

**}**

**TX code :**

**#include "Source.h"**

**void main()**

**{**

**long range;**

**int Counter=0;**

**byte iDist;**

**int16 distance;**

**char distance\_L, distance\_H, light\_intensity ;**

**setup\_adc\_ports(NO\_ANALOGS);**

**setup\_adc(ADC\_OFF);**

**setup\_psp(PSP\_DISABLED);**

**setup\_timer\_0(RTCC\_INTERNAL|RTCC\_DIV\_1);**

**setup\_timer\_1(T1\_DISABLED);**

**setup\_timer\_2(T2\_DISABLED,0,1);**

**/\***

**Enter (E2) ++**

**-----**

**in out**

**--- ---**

**E2 Sensor | E0 Sensor**

**0 | 0**

**1 | 0**

**1 | 1**

**0 | 1**

**Exit (E0) --**

**-----**

**out in**

**--- ---**

**E0 Sensor | E2 Sensor**

**0 | 0**

**1 | 0**

**1 | 1**

**0 | 1**

**\*/**

**while (1)**

**{**

**if(E2Sonar() && !E0Sonar()) Counter++;**

**if(E0Sonar() && !E2Sonar()) Counter--;**

**printf("\fTotal Number of Students=%d ",Counter);**

**restart\_wdt();**

**}//main loop**

**}**

**boolean E2Sonar()//increment**

**{**

**byte iDist;**

**int16 distance;**

**char distance\_L, distance\_H, light\_intensity ;**

**i2c\_start();**

**i2c\_write(0xE2);**

**i2c\_write(0x02);**

**i2c\_write(0x18);**

**i2c\_stop();**

**i2c\_start();**

**i2c\_write(0xE2);**

**i2c\_write(0x00);**

**i2c\_write(0x51);**

**i2c\_stop();**

**delay\_ms(100);**

**i2c\_start();**

**i2c\_write(0xE2);**

**i2c\_write(0x01);**

**i2c\_stop();**

**i2c\_start();**

**i2c\_write(0xE3);**

**light\_intensity = i2c\_read(1);**

**distance\_H = i2c\_read(1);**

**distance\_L = i2c\_read(0);**

**i2c\_stop();**

**distance = make16(distance\_H,distance\_L);**

**// printf("\flight\_intensity : %u Lux\nDist.: %lu cm", light\_intensity , distance);**

**if(distance<90)**

**{**

**return true;**

**}**

**/\*else**

**{**

**output\_low(PIN\_C0);**

**printf("No Object Detected \n\r");**

**}**

**\*/**

**return false;**

**}**

**boolean E0Sonar()//decrement**

**{**

**byte iDist;**

**int Counter=0;**

**int16 distance;**

**char distance\_L, distance\_H, light\_intensity ;**

**i2c\_start();**

**i2c\_write(0xE0);**

**i2c\_write(0x02);**

**i2c\_write(0x18);**

**i2c\_stop();**

**i2c\_start();//Config**

**i2c\_write(0xE0);**

**i2c\_write(0x00);**

**i2c\_write(0x51);**

**i2c\_stop();**

**delay\_ms(100);**

**i2c\_start();//Address of Register to Read**

**i2c\_write(0xE0);**

**i2c\_write(0x01);**

**i2c\_stop();**

**i2c\_start();//Go to Read Mode**

**i2c\_write(0xE1);**

**light\_intensity = i2c\_read(1);**

**distance\_H = i2c\_read(1);**

**distance\_L = i2c\_read(0);**

**i2c\_stop();**

**distance = make16(distance\_H,distance\_L);**

**//printf("\flight\_intensity : %u Lux\nDist.: %lu cm", light\_intensity , distance);**

**if(distance<90)**

**{**

**//output\_high(PIN\_C0);**

**//printf("Object Detected less than 50 cm\n\r");**

**return true;**

**}**

**/\*else**

**{**

**output\_low(PIN\_C0);**

**printf("No Object Detected \n\r");**

**}**

**\*/**

**return false;**

**}**