

Project Final Report

Encrypted Messenger

April 29, 2008

Brian Chan, bobes@gwu.edu

Project Abstract

The Encrypted Messenger takes input data from a keyboard connected to one Z8 development board, encrypts the data and transmits it to another Z8 board where the message is decrypted and then displayed. The boards communicate over an RS232 link (using three wires) and utilize the AES 128 bit encryption scheme. The message is displayed on the LED grid built into the Z8 board.

Status

Originally, the project set out to communicate over a wireless link. The Linx 433-LC modules were used in the original design of the project. However, I was unable to get them to work. The transmitter was connected to the TX pin on the UART and the receiver was connected to the RX pin. The rest of the connections were made according to the provided schematics from the manufacturer. Using the same code, the wireless transmitter/receiver pair were taken out and the TX from one board was connected to the RX of the other and vice versa to create a connection from the UART of one board to the other. In this setup, the program worked as it was designed. So the problem lied with the wireless transmitters.

To get around this, the two Z8 boards were just connected using three wires, RX, TX and ground. The RS232 connection requires a shared ground in order to transmit the information.

The other issue was with the LCD display. The specifications provided were inadequate. I found a second data sheet for the LCD controller chip which provided the needed instructions for operating the display. But by the time I found the new data sheet, it was too late. As a work around, the message is scrolled across the LED grids on the Z8 boards.

Specification

The final project used two Z8 Encore Development boards with the Z8F6403 microcontroller. The Zilog chip has plenty of IO PINs, a DUART, and edge triggered interrupts, all of which were used in this project. Other hardware included a PS2 female connector and a PS2 keyboard circa 2002.

The transmitter/receiver pair I had planned to use was the Linx TXM-433-LC and the Linx RXM-433-LC. These modules operated at 433MHz and connected to the UART

Project Final Report

at a maximum baud rate of 5,000 bps. Amplitude modulation using On-Off Keying was utilized by the wireless modules to transmit a digital signal.

The display was a Varitronix MDLS16166D-01 Liquid Crystal Display. It could display 2 rows with 16 characters on two rows. The LCD module utilized a Novatek NT3881DH-91 LCD controller. The Varitronix data sheet did contain pertinent information such as the electrical specifications and timing diagrams for the parallel inputs. However, it omitted the command bytes to turn the display on and off as well as cursor movement. This information was available in the Novatek datasheet.

For encryption, 128 bit AES algorithm was used. Information on AES was found on Wikipedia. Test cases were generated using a Javascript implementation of AES found online (<http://people.eku.edu/styere/Encrypt/JS-AES.html>).

Two main programs were written, one for the receiving board, one for the transmitting board. Starting on the transmitter side (the more complicated side) there were 5 main modules. The keyboard scan, message transmission, message receive, encryption, and some debug features.

The keyboard scan module basically detects a key press on the keyboard, converts the scan code to ASCII and stores it in a character array. The keyboard module for this project only covers lower case characters and the basics for sending a simple text message. Tab, ctrl, alt, backspace, and delete are all not recognized.

The encryption module should take in a message and a key, and encrypt the message using the encryption key. The project currently only supports a hard coded encryption key. However, the program can be modified to generate a random key value and send it to the receiver.

The message transmission module transmits a packet over the RS232 connection. It also calculates the checksum of the packet. More of the packet fields will be described in the Implementation section.

Though the transmitter board's main task is to send a message, it still must receive acknowledgements from the receiver board. So the message receive module must take in a message over the RS232 link. The messages that the transmitter would receive are ACK and NACK packets.

For debugging, the LED grid displays the status of the program. When a message is sent, but before a response is received, the LED grids will display "Send". If an acknowledgment is received, it will display "Ack". If a not acknowledge is received, it will display "Nack". Also, a clear LED function was put in to make sure the program wasn't caught in a display loop for scrolled messages.

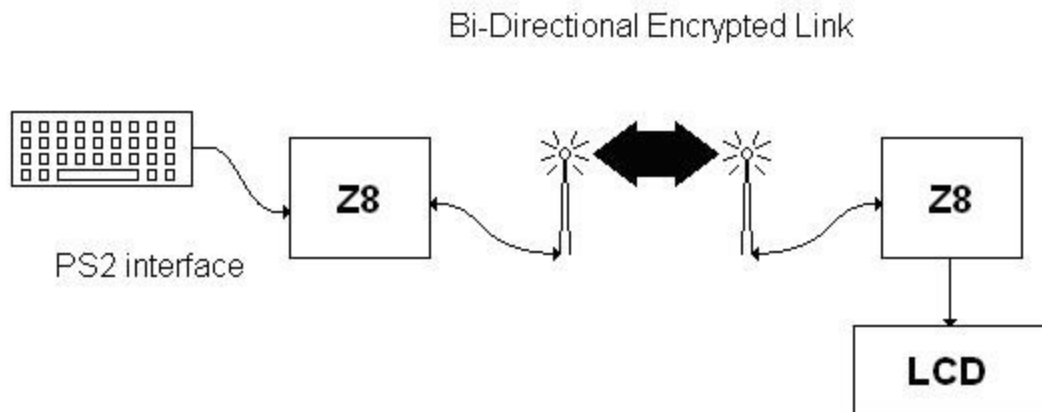
The receiver board had 4 main modules: display, message transmission, message received, and decryption. The display module basically managed the scrolling of a

received message on the LED grid. The message scrolls across the 4 LED grids, letter by letter, until the end of the message is reached. At that point, the LED display will show the first 4 letters of the message.

Message transmission was far simpler, but similar to the module in the transmitter. Only NACK and ACK packets needed to be sent from the receiver. Received messages however would be the encrypted messages. So as bytes came in over the RS232 port, they needed to be parsed and the encrypted message isolated. The decrypt module just decrypted the received message using the encryption key (since AES is a symmetric encryption algorithm, the same key must be used).

Implementation & Construction

The original hardware setup consisted of the two Z8 boards each with a transmitter/receiver pair (for bidirectional communication). The receiver board had the display connected to it and the transmitter board would be connected to the keyboard via a female PS2 connector.



Preliminary Hardware Diagram

The actual completed project consisted of the two Z8 boards connected by 3 wires (TX to RX, RX to TX and GND to GND). The transmitter board still had the keyboard connected but instead of using the LCD module, the project utilized the LED grid built into the Z8 development board.

The keyboard connector has four connections that need to be made. The clock line from the PS2 connector was connected to an interrupt PIN. The data line on the PS2 was connected to a general purpose IO PIN. Both were set as inputs to the Z8 chip. The last two connections were to power and ground. The keyboard sends over an 11 bit message for each button pressed. The message begins with a start bit, followed by the 8 bit scan code. The next bit is the parity bit. The AT keyboard uses an odd parity. The last bit is a stop bit indicating the end of a message.

The input line connect to the clock pin was set to trigger and interrupt on a falling edge. At this point, the data line could be sampled to find the value of the incoming bit. The clock line was only active when a key is pressed. The other thing to note is that for each button pressed, a minimum of 3 scan codes are sent. One indicates the button pressed. The next code, F0, indicates that the button has been released. After the release code, the scan code is sent again to indicate exactly which keyboard button has been released. Once a key press was determined, the scan code was converted to an ASCII code and the character was loaded into an array. When the array was full, 16 characters, or the enter key was pressed, the message was encrypted, encapsulated into a packet and sent over the serial link.

For the transmission of data, the UART handles some of the flow control (with a start and stop bit) and does have the ability to do error checking (parity), however the error checking feature was disabled. However, with the wireless link, there are a few other things to consider. First of all, On-Off Keying only has two states, on and off; which is almost perfect for digital signals. However, a bit of interference can trigger the receiving UART into thinking a message was incoming. So for the communication a bit of a protocol stack was implemented. The data would be transmitted into packets. Each byte of the packet would be fed into the UART and send in 1 byte chunks. The UART was set to transmit and receive was 2400 baud.

The packet would contain the following fields: start code, command, length, message, and checksum. The start code serves a dual purpose. If the receiver does not see the start code, then no message was transmitted. This prevents a little bit of interference from causing the receiver to think a message was coming in. Also, a signal transmitter can address receivers by changing the start code in puts in packets.

The command would allow for different functions to take advantage of the same packet. Commands implemented were: message, acknowledged (ACK) and not acknowledged (NACK). After receiving a message, the receiver will either send an ACK or NACK packet. The ACK will mean that the message was received, the command was proper and the checksum (error checking, more on that later) worked out. The NACK would mean that some error occurred in transmission (a start code was seen but something else was wrong) and that a retransmission is necessary. Unfortunately, the auto retransmit was not implemented in this project. The transmitter will simply display to the user that a NACK was received.

The length packet in this protocol does not signify how many bytes are in the message field. It instead describes how many bytes are significant in the message field. In the AES protocol, the messages must be in 128 bit (16 byte) chunks. If a message is less than 16 bytes, then the message is padded with 0's and then encrypted. So the length field indicates how many characters are in the message field, even though the message field is always 16 bytes long. When the message is decrypted and displayed, the display doesn't show extra characters if the message is less than 16 characters.

The checksum uses a simple bitwise XOR operation on each byte in the packet. Each byte of the packet is XORed with the previous byte to create a 1 byte checksum. This checksum is appended to the end of the packet when the message is sent out. The receiver performs the same operation, XORing the bytes as they come in. When the checksum byte is XORed with the rest of the packet, it should result in 0x00 for a transmission with no errors. If the value is anything else, then there was an error in transmission and a retransmit should occur. The start code was the only byte that was left out of the XOR calculation. If an error occurred on the start byte, the receiver would just ignore the message.

On the receiver end, the program would listen for a byte on the UART. When a message came in, it would check the start byte. If the start byte was correct, it would receive the rest of the message. If it was incorrect, the message was discarded. If a message was received, the receiver board would decrypt the message and display it on the LED grids. For the scrolling display, each character of the message would move over by one grid until the end of the message was reached. At that point, the first four characters of the message would stay illuminated on the screen. The message scrolled over one place about every .4 seconds.

To test the serial link, I wrote a simple ping program. The transmitter would send one byte out on the UART and display PING on the display. The receiver would look for that byte and when it was received the display would show PONG. One button on each board sent the ping message and another button cleared out the display.

To test the encryption, I used the online Javascript AES implementation to create test cases. I set the Z8 board to output encrypted messages out on the UART into a terminal on my PC and then compared the data. When I wrote the decryption algorithm, I just encrypted a message, displayed the cipher text, then passed it back to decryption function and outputted that data to the UART.

Retrospective

There were a few changes that needed to be made for the project. First of all, some of the hardware components was either not working, or the datasheets did not provide enough information. I would have liked to have had maybe an example program to test out the hardware, or some good application notes to have a base of information to build on.

The not being able to use the transmitter/receiver pair did make things difficult. However, since they had a serial interface, it was easier to transition to a backup plan of connecting the two boards via a serial link. Instead of changing the code, I just needed to change the connection. However, troubleshooting the serial link and making that work without a serial cable was a bit tricky. I didn't realize that the link required a shared ground connection between the two boards. Simply connecting RX to TX would not work.

Project Final Report

The LCD display not working was much more difficult to get around. Since I was unable to get that component working, I needed to use the LEDs as a backup plan. Coding for the LEDs was much different than coding for the LCD display and required some extra code and testing from what I had already written.

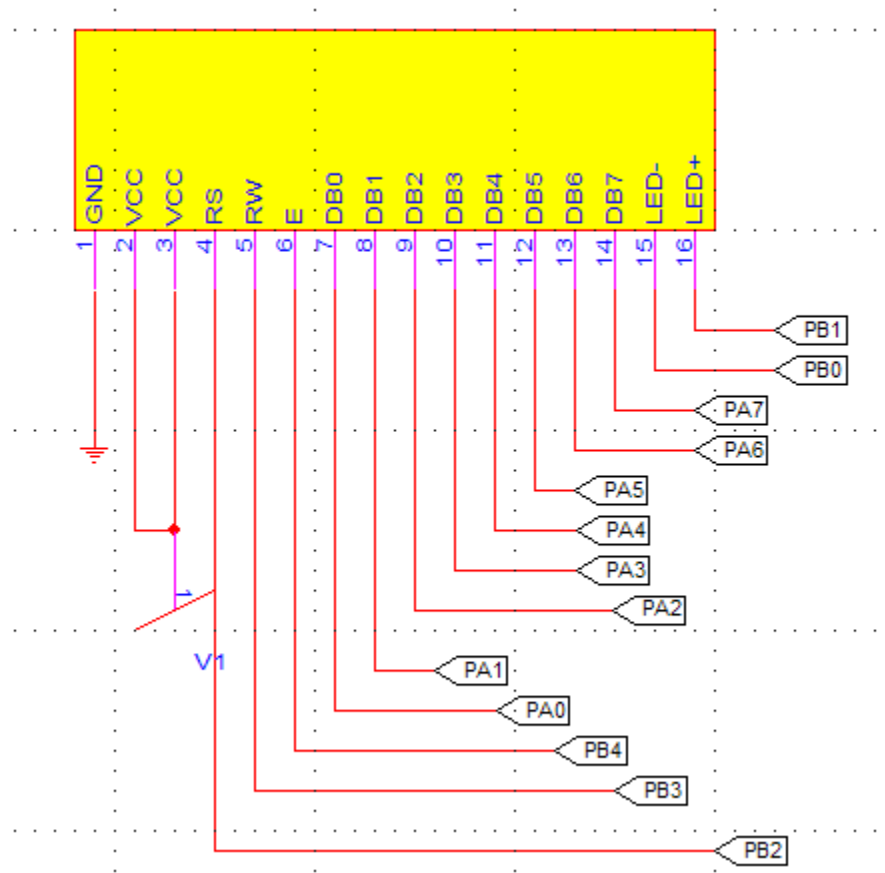
On the software side, I would have liked to have the transmitter auto retransmit the message to the receiver on a NACK. To test this I also would have liked one of the buttons to control the mode. For example, pressing SW2 would put the transmitter into error mode where it will purposely send the entered message with a bad checksum. The receiver in this case would send the NACK and then the transmitter would auto retransmit the corrected packet.

I would have also liked to create a random number generator or pseudo-random number generator to create new encryption keys for encrypting the data. If the transmitter wanted to use a new encryption key, it would send the new key in a packet encrypted with the old key and with a new command to signify that the data in the message field is a new encryption key, not a message to be displayed. All further transmissions would use this new key.

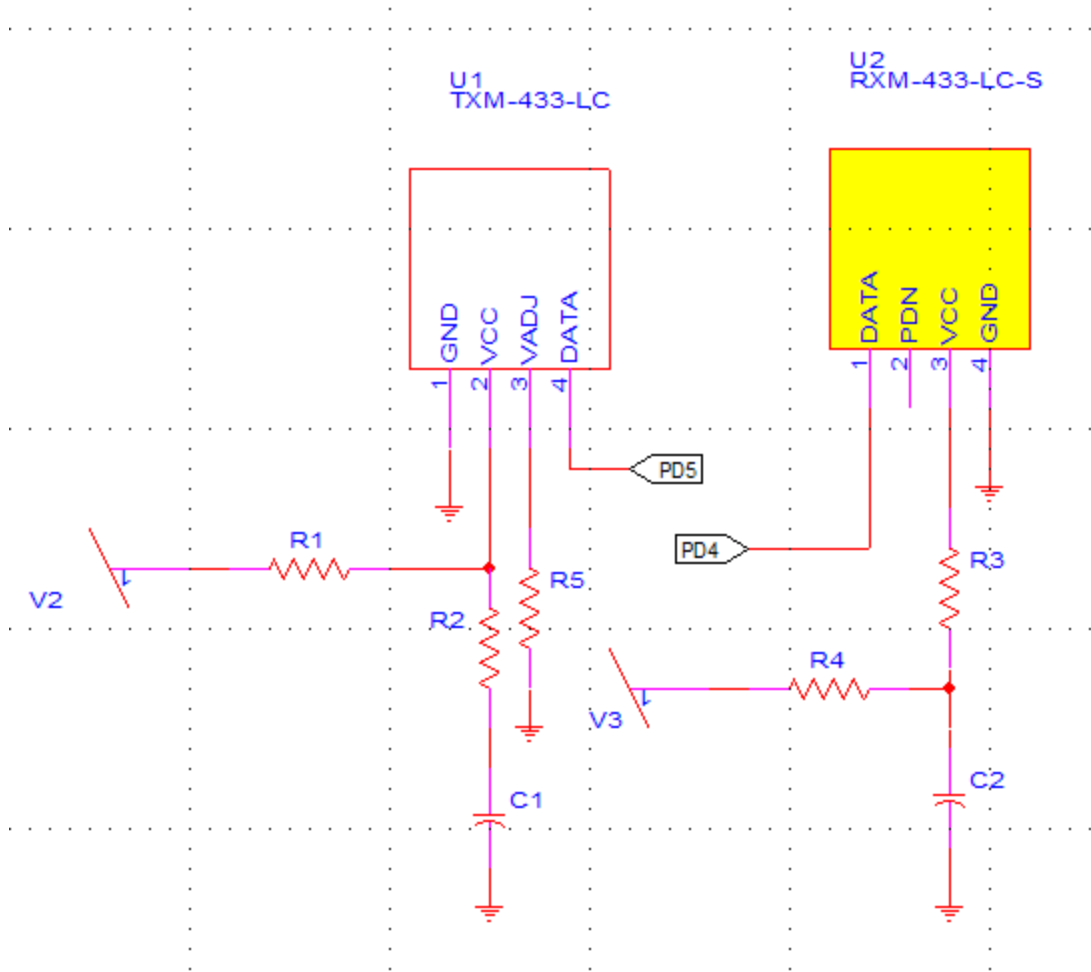
Also, for the ACK and NACK there isn't really a way to ensure that the response packet was sent from the receiver. An eavesdropper might pick up on the protocol and insert a NACK and force a resend to get more data. Using the random number generator, a nonce could be inserted into the message packet. The receiver would transmit the ACK (or NACK) with the nonce encrypted in the message field of the ACK packet.

If I knew then what I knew now, I would have planned for more things to go wrong with the hardware interfaces. I would have definitely asked for more help more frequently, and I would have spent more time researching the parts and making backup plans.

Attachments



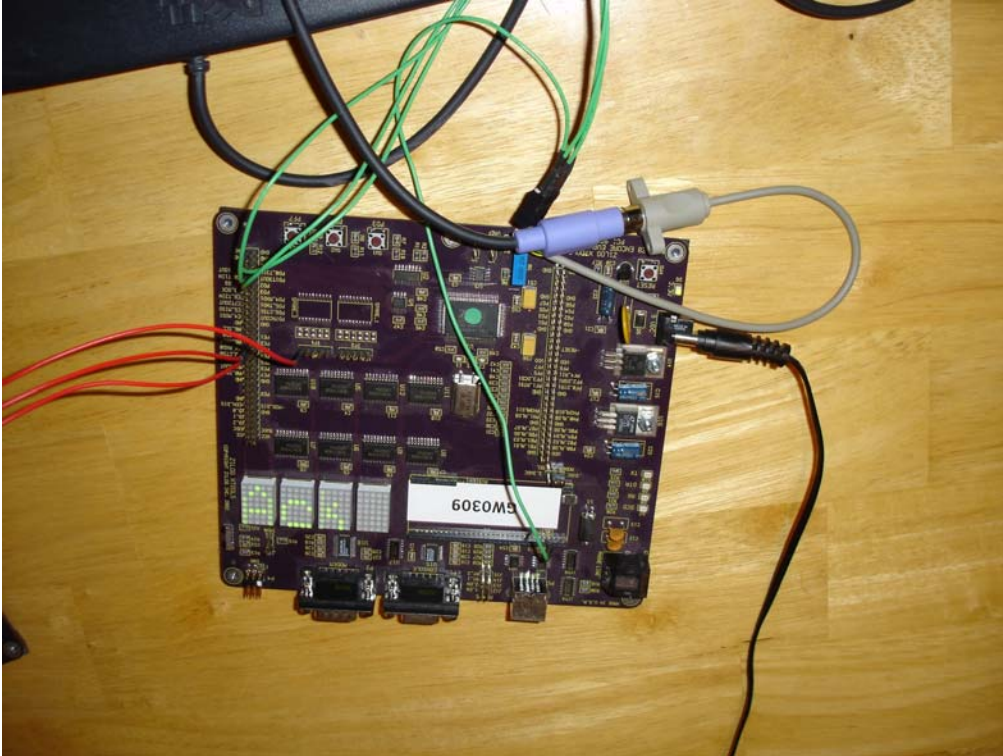
Hardware Schematic for LCD screen



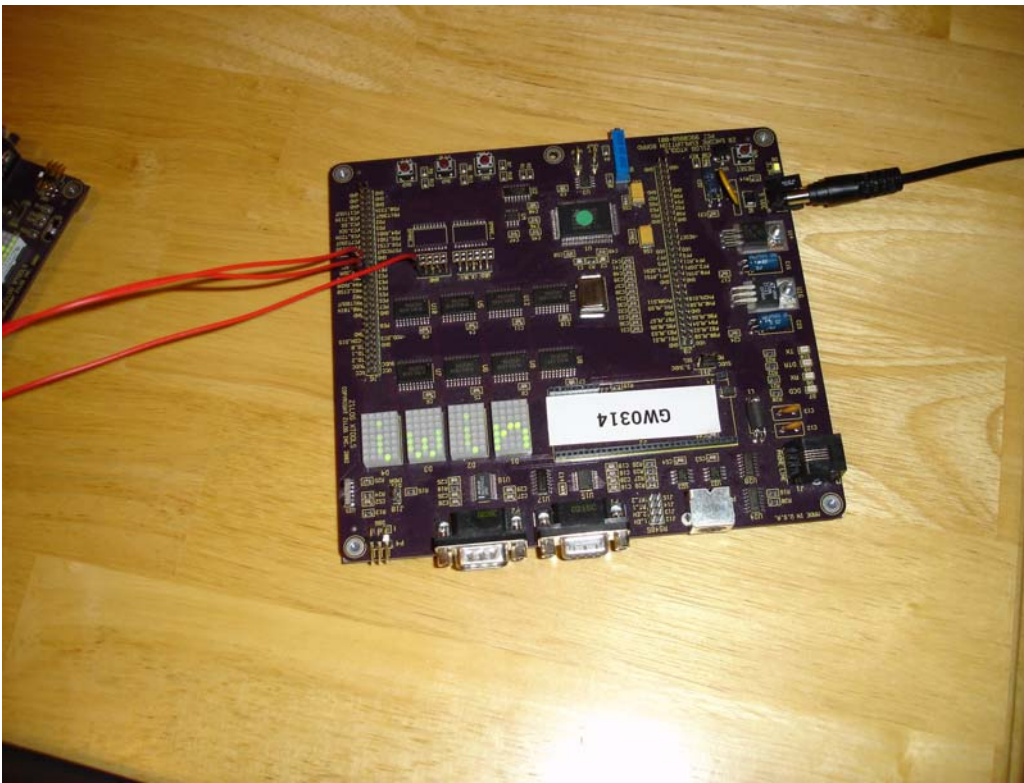
Hardware Schematic for the Linx Transmitter/Receiver pair

All schematics created with PCB123 software.

Project Final Report

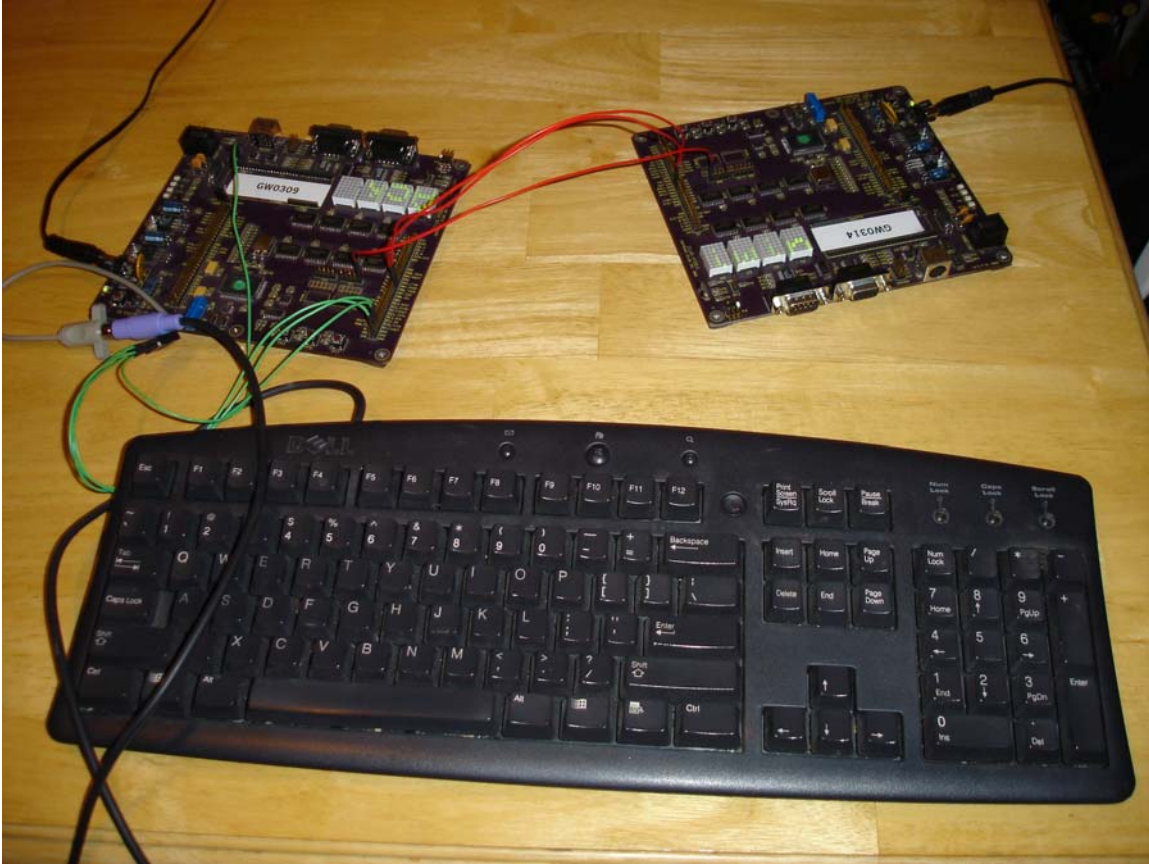


Transmitter board



Receiver Board

Project Final Report



Final Setup