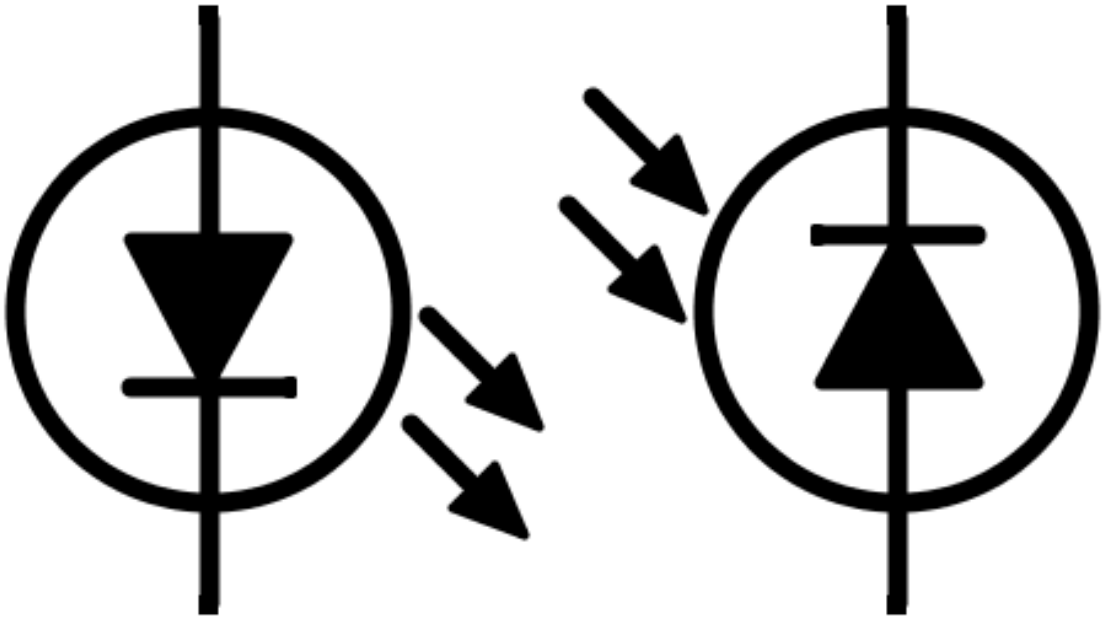
VERSION 1.0

MAY 31, 2014



DATA TRANSFER USING LIGHT

A PROOF OF CONCEPT FOR DATA TRANSMISSION USING LIGHT AS A MEDIUM

RYAN SUAREZ

CALIFORNIA POLYTECHNIC STATE UNIVERSITY, SAN LUIS OBISPO
SENIOR PROJECT SPRING 2014

CONTENTS

DATA TRANSFER USING LIGHT

INTRODUCTION

The idea for this project was taken from a TED talk by Harald Haas titled “Wireless Data from Every Light Bulb”. In the video, Haas describes how an LED can strobe faster than the human eye can detect. He also discusses the widespread use of light bulbs through the current infrastructure of the world. Given these two topics, he argues that everyday ordinary light bulbs can be exchanged for a data transmitting LED infrastructure. This change would allow for more efficient, reliable, and secure data transmission.

SCOPE

The scope of this project is to be a proof of concept for Haas’s idea of LED data transmission. The project seeks to answer the question: Is it possible to strobe an LED faster than the human eye can detect and still maintain a reliable data connection? In order to test these requirements there will be two circuits: a transmitter and a receiver. The transmitter will send reliable packets and the receiver will accept and decode the packets – displaying the encoded message on a LCD screen.

TIMELINE

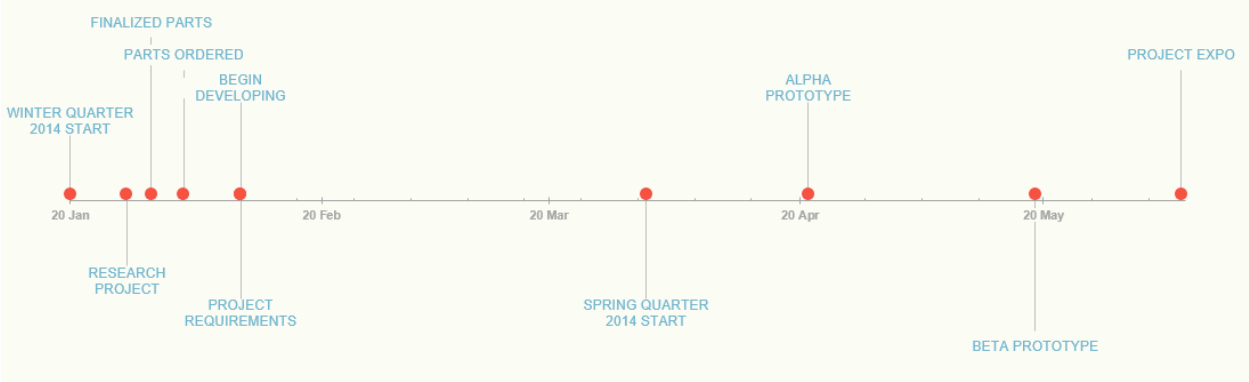


FIGURE 1 TIMELINE The expected timeline and milestones to complete for this project.

DATE	MILESTONE
1/30/2014	Winter Quarter 2014 Start
1/20/2014	Research Project
1/27/2014	Finalized Parts
2/3/2014	Parts Ordered
2/10/2014	Project Requirements
2/10/2014	Begin Developing
4/1/2014	Spring Quarter 2014 Start
4/21/2014	Alpha Prototype
5/19/2014	Beta Prototype
5/31/2014	Project Expo

TABLE 1 TIMELINE The expected timeline and milestones to complete for this project

HIGH LEVEL OVERVIEW

BLOCK DIAGRAM

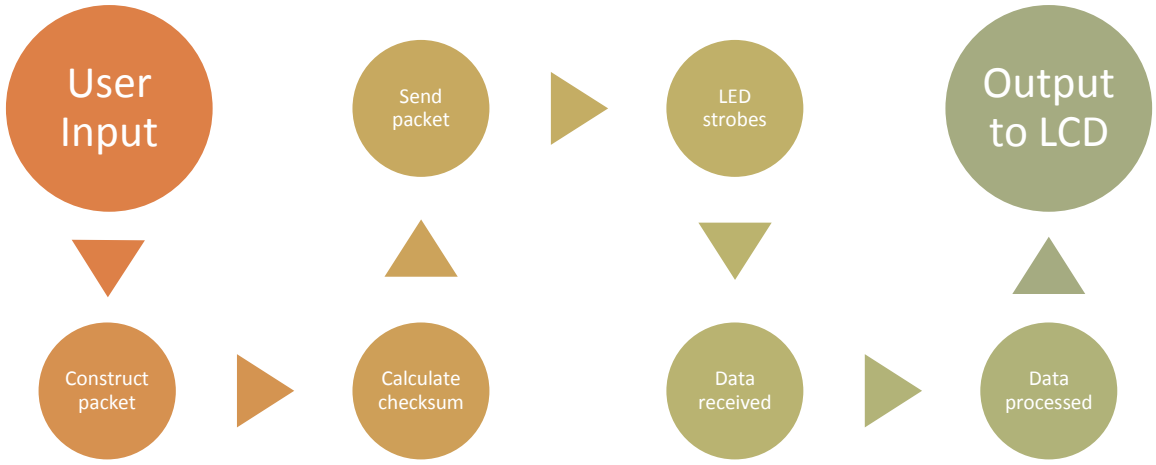


FIGURE 2 HIGH LEVEL BLOCK DIAGRAM Process flow at its highest Level

TRANSMITTER FLOW CHART

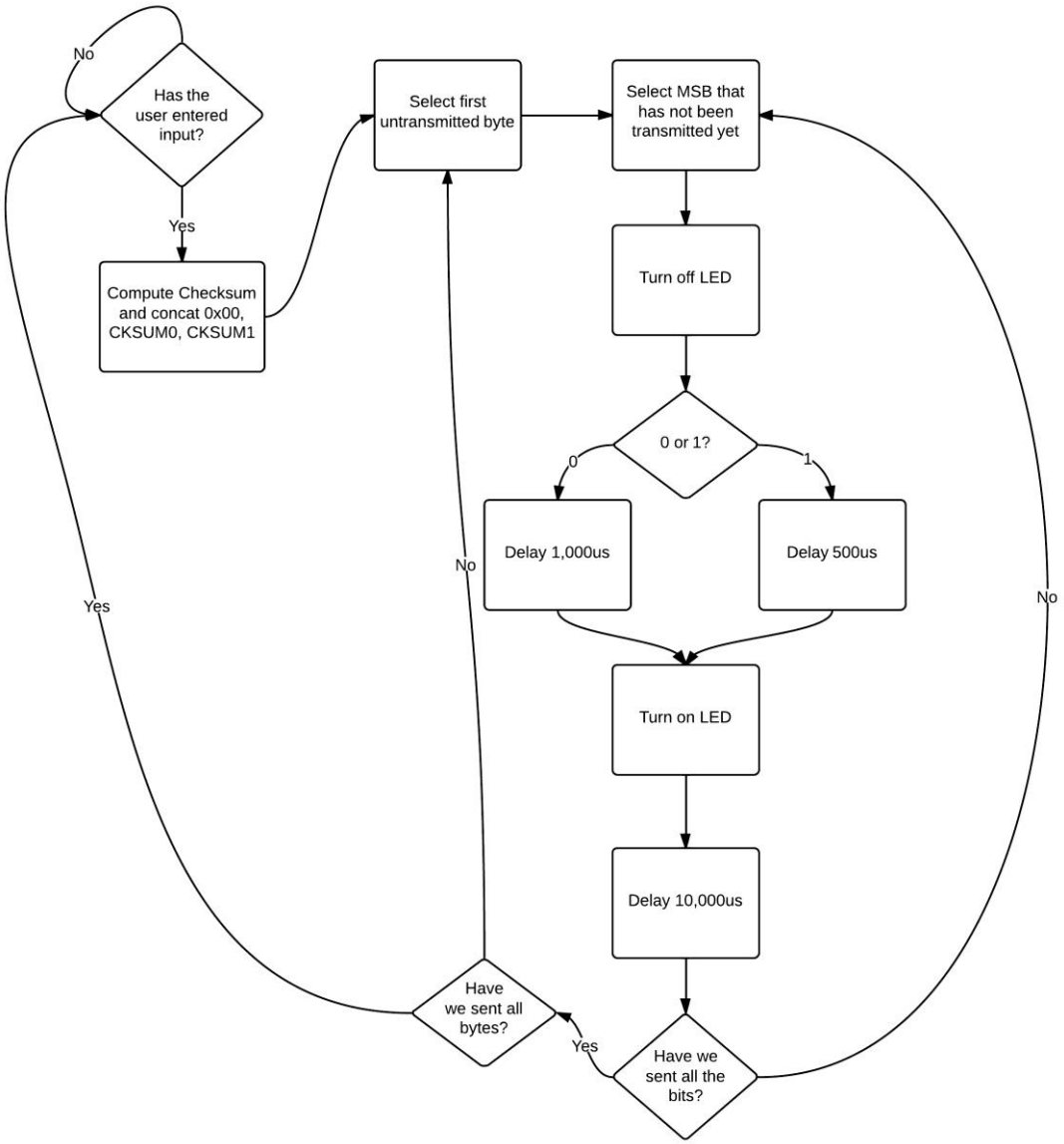


FIGURE 3 TRANSMITTER FLOW CHART Chart displaying the order of operations for the transmitter

RECEIVER FLOW CHART

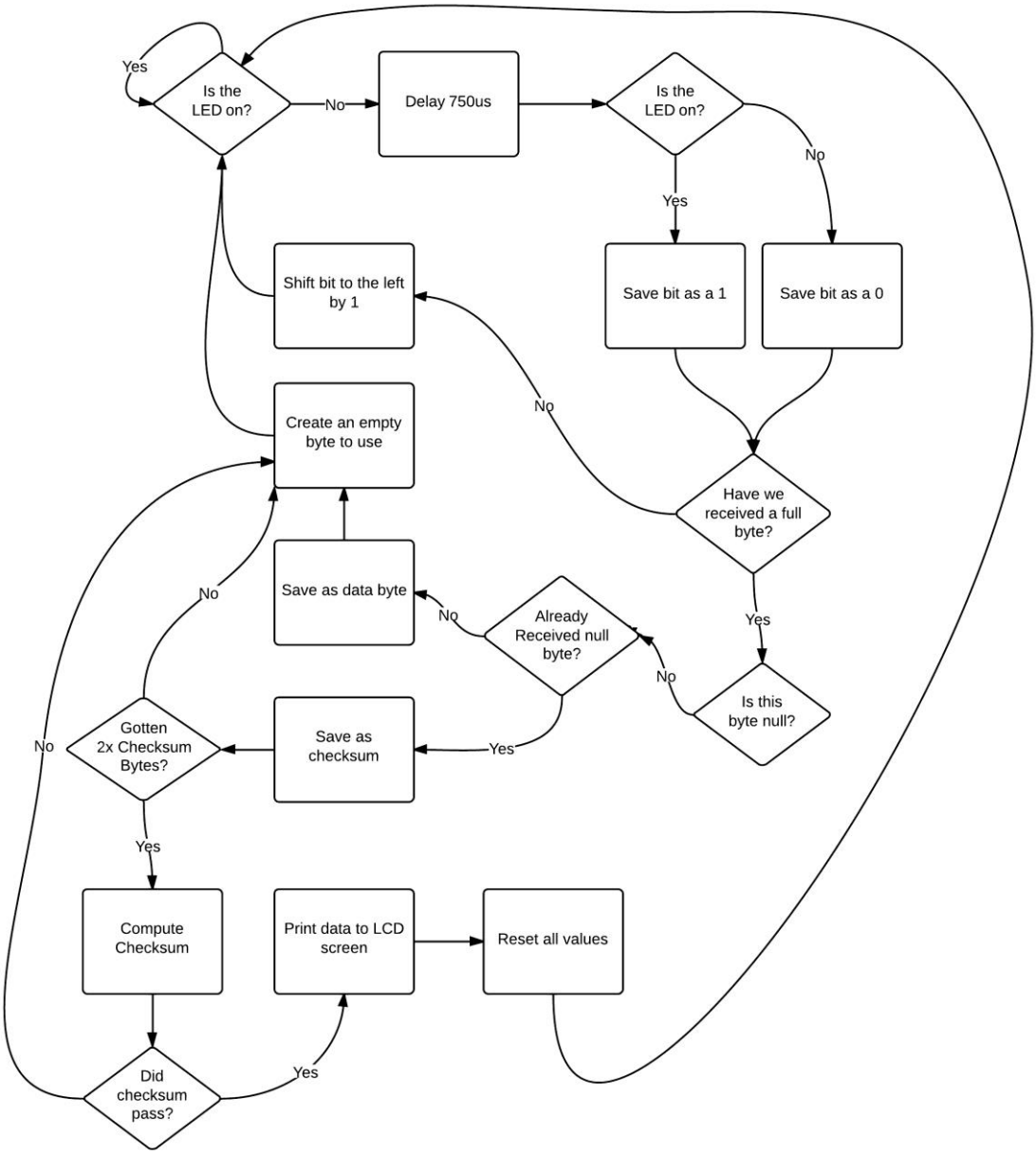


FIGURE 4 RECEIVER FLOW CHART Chart displaying the order of operations for the receiver

TIMING DIAGRAM FOR TRANSMITTER

FIGURE 5 SAMPLE TIMING DIAGRAM Timing Diagram for sending 0x65 (‘e’)

IMPLEMENTATION

MATERIALS AND SUPPLIES

- 2x Arduino Uno R3 (Atmega328)
- 1x OP955 P-N Photodiode
- 2x C503C-WAN-CBADB151 Cool White LED
- 1x 1602A LCD 16x2 Display
- 1x 10Ω - 5MΩ Potentiometer
- 1x 10kΩ Resistor
- 1x 100kΩ Resistor
- 1x 510kΩ Resistor
- Arduino IDE (http://arduino.cc/en/main/software)

TRANSMITTER ELECTRICAL SCHEMATIC

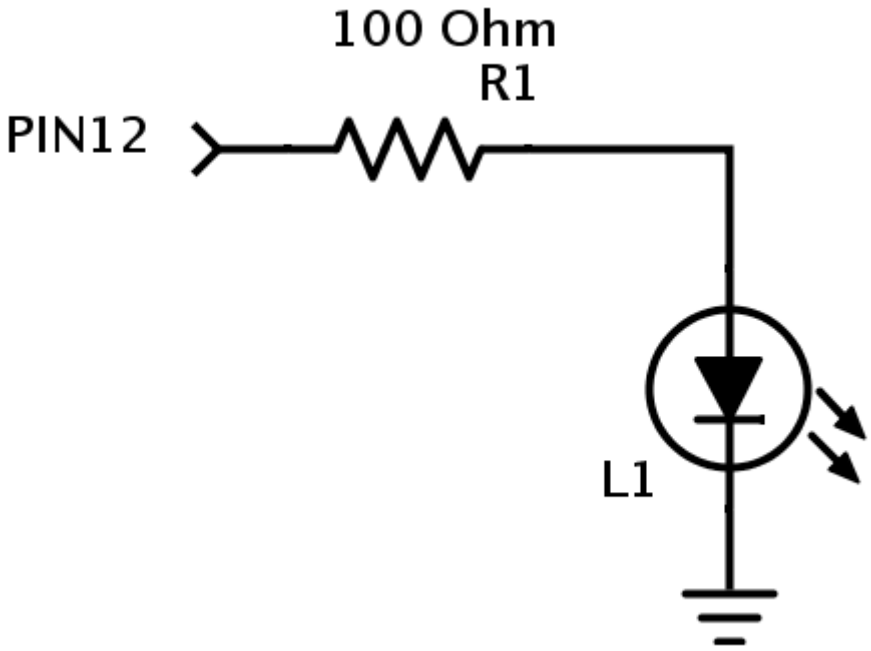


FIGURE 6 TRANSMITTER Electrical Schematic for the Transmitter

NOTE: This circuit will get input PIN12 from only a single Arduino board. It will be a completely separate circuit entirely from the Receiver which will be attached to the second Arduino board.

RECEIVER ELECTRICAL SCHEMATIC

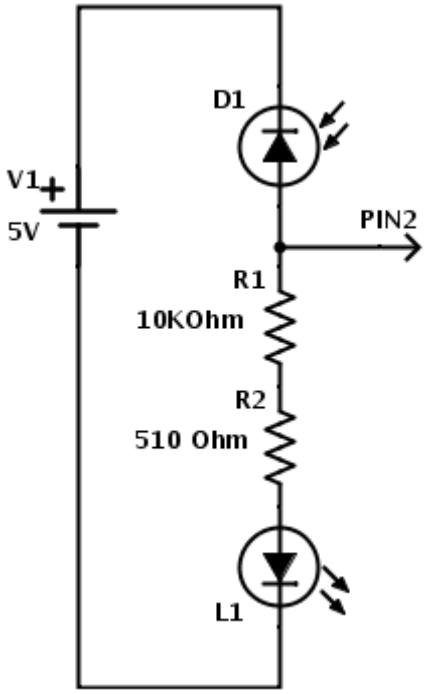


FIGURE 7 RECEIVER Electrical Schematic for the Receiver

NOTE: This circuit will output to PIN2 from only a single Arduino board. It will be a completely separate circuit entirely from the Transmitter which will be attached to the first Arduino board.

LCD SCREEN WIRING SCHEMATIC

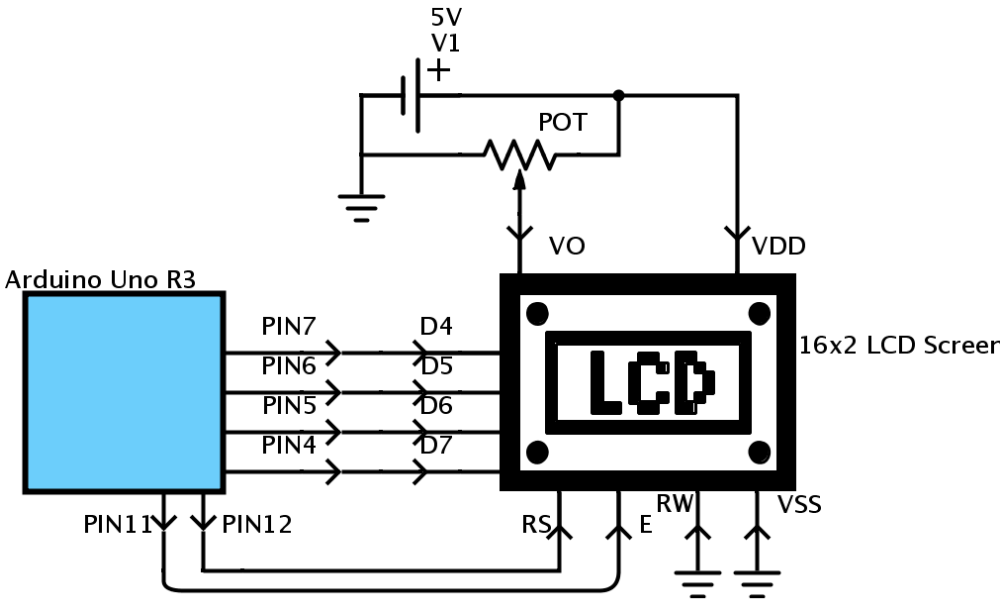


FIGURE 8 WIRING SCHEMATIC FOR LCD SCREEN

PROGRAMMING THE ARDUINO BOARDS

1. Set up the Transmitter according to Figure 5. Use only a single Arduino board to connect PIN12 to the circuit.
2. Plug the Arduino into a USB port on the computer via a USB cable.
3. Load the Transmitter source code into the Arduino IDE (Found in the Appendix).
4. In the Arduino IDE, click **View** -> **Source Port** and record the COM port being used by Arduino.
5. Click the upload button to program the Arduino.
6. Open the **Serial Monitor** by pressing Ctrl + Shift + M.
7. Data can be sent by typing into the pop up console.
8. Set up the Receiver circuit as seen in Figure 6, be sure to position the Photodiode and the LED on the transmitter so that they are facing each other. This step is crucial to correct execution of the system, if the LED is not focused on the photodiode then the photodiode circuit will not be driven properly.
9. Repeat steps 2-6 in a separate Arduino IDE instance. Be sure that this IDE instance is not using the same COM port as the transmitter.
10. Set up the LCD Screen circuit. The pins for this circuit correlate to the pins on the same Arduino board as the Receiver.
11. Adjust the potentiometer in order to have a clear display of characters. (NOTE: Output voltage of 0.8V on the wiper worked for the screen used in this experiment) If there are no characters displayed the voltage is too high on Vo, if there are only squares displayed then the voltage is too low on Vo.
12. Hit reset on the Arduino Transmitter.
13. Hit reset on the Arduino Receiver.
14. You will be greeted with “Welcome!” on the LCD screen if all is set up correctly.
15. Messages typed into the Arduino **Serial Monitor** on the transmitter COM port will be sent to the receiver to process. The LCD screen will display the message.

CONCLUSION

PROJECT SUMMARY

The aspirations of Harald Haas were to replace the entire infrastructure that is set up today around light bulbs with this new LED data transfer technology. While his dream is still years away, this project demonstrates that it is entirely possible to have a network set up based solely around light transfer.

This project was very limited in funding. As a result, the serial data transfer was not as quick or efficient as it could have been. However, given that it was a proof of concept the project was a success. The transmitter was capable of sending reliable data through an LED faster than the human eye can detect. The receiver was also capable of receiving and interpreting this data correctly.

PROJECT EXPANSIONS

If this project were funded, it could easily be expanded upon by upgrading the hardware. The Arduino Uno R3 has a 16MHz clock which translates into a period of 62.5 ns. The LED and photodiode have switching speeds of about 5ns each. The Arduino board could be upgraded to have a processor with a much higher clock speed. The LEDs and Photodiodes used in the experiment and are very generic. There are more expensive options that are optimized for high switching speeds.

Another possibility that is well beyond the scope and budget of this project is to have data be transmitted and received in parallel. Different data could be sent using different wavelengths of light asynchronously and received using a spectrometer. Since white is the usual color of light emitted from light bulbs and white light is actually made up of all the different wavelengths of light the same idea of this replacing a normal every day light bulb is not only possible but would be the optimum solution.

SOURCE CODE

TRANSMITTER

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <string.h>

#define htons(x) ( ((x)<<8) | (((x)>>8)&0xFF) )
#define DELAY_HIGH 500 // us
#define DELAY_LOW 1000 // us
#define DELAY_BETWEEN 10000 // us.  This is the delay between bits.

int ledPin = 12;
int inputPin = 2;

volatile int state = LOW;

byte byteReceived = 0;
int bitsReceived = 0;
void setup()
{
  pinMode(ledPin, HIGH);
  pinMode(inputPin, INPUT);
  Serial.begin(9600);
}

byte incomingByte = 0xDEADBEEF;
void loop()
{
  String data;
  int i;
  // send data only when you receive data:
  if (Serial.available() > 0)
    // read the incoming byte:
    data = Serial.readString();

  // Make sure there are actually characters to send.
  if (data.length() > 0)
  {
    Serial.print(data);

    // Make room, calculate, and concatenate checksum.
    data.concat("\0\0");

    char packet[data.length() + 3];
    data.toCharArray(packet, data.length() + 3);

    packet[data.length()] = 0;
    packet[data.length() + 1] = 0;
    packet[data.length() + 2] = 0;
    unsigned short checksum = (in_cksum((unsigned short *)packet,
sizeof(packet)));
    memcpy(packet + strlen(packet) + 1, &checksum, 2);
```

```
    unsigned short check = in_cksum((unsigned short *)packet,
sizeof(packet));

    // Reverse all bits in the packet...big endian vs little endian.
    for (i = 0; i < sizeof(packet); i++)
    {
      packet[i] = reverse(packet[i]);
    }

    sendPacket(packet, sizeof(packet));
  }
}

// Send a single byte.
void sendByte(byte byteToSend)
{
  int bitsLeft = 8;
  while(bitsLeft-- > 0)
  {
    digitalWrite(ledPin, HIGH);
    delayMicroseconds(DELAY_BETWEEN);

    digitalWrite(ledPin, LOW);
    if (byteToSend & 0x01)
      delayMicroseconds(DELAY_HIGH);
    else
      delayMicroseconds(DELAY_LOW);
    byteToSend >>= 1;
    Serial.print("X");
  }
  Serial.println();
  digitalWrite(ledPin, HIGH);
}

// Send the entire packet / array of bytes.
void sendPacket(char *packet, int len)
{
  int i;
  for (i = 0; i < len; i++)
  {
    //Serial.println(sizeof(packet));
    Serial.println(reverse(packet[i]), HEX);
    sendByte(packet[i]);
  }
}

// Reverse bit order...big endian vs little endian.
unsigned char reverse(unsigned char b) {
  b = (b & 0xF0) >> 4 | (b & 0x0F) << 4;
  b = (b & 0xCC) >> 2 | (b & 0x33) << 2;
  b = (b & 0xAA) >> 1 | (b & 0x55) << 1;
  return b;
}

/*
 * in_cksum --
 *      Checksum routine for Internet Protocol family headers (C Version)
```

```
 */
unsigned short in_cksum(unsigned short *addr,int len)
{
  register int sum = 0;
  unsigned short answer = 0;
  register unsigned short *w = addr;
  register int nleft = len;

  /*
         * Our algorithm is simple, using a 32 bit accumulator (sum), we add
   * sequential 16 bit words to it, and at the end, fold back all the
   * carry bits from the top 16 bits into the lower 16 bits.
   */
  while (nleft > 1)  {
    sum += *w++;
    nleft -= 2;
  }

  /* mop up an odd byte, if necessary */
  if (nleft == 1) {
    *(unsigned char *)(&answer) = *(unsigned char *)w ;
    sum += answer;
  }

  /* add back carry outs from top 16 bits to low 16 bits */
  sum = (sum >> 16) + (sum & 0xffff);     /* add hi 16 to low 16 */
  sum += (sum >> 16);                     /* add carry */
  answer = ~sum;                          /* truncate to 16 bits */
  return(answer);
}
```

RECEIVER

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <LiquidCrystal.h>

#define DELAY_MIDDLE 750 // us

int pbIn = 0;                   // Interrupt 0 is on DIGITAL PIN 2!
int ledOut = 13;                 // The output LED pin
volatile int state = LOW;       // The input state toggle
int bitReady = 0;

LiquidCrystal lcd(12, 11, 7, 6, 5, 4);

String previousPacket = "";

// the setup routine runs once when you press reset:

void setup()
{
  // Set up the digital pin 2 to an Interrupt and Pin 4 to an Output
  lcd.begin(16, 2);
  lcd.print("Welcome!");
  pinMode(ledOut, OUTPUT);
  pinMode(2, INPUT);
  Serial.begin(9600);
  //Attach the interrupt to the input pin and monitor for ANY Change
  attachInterrupt(pbIn, stateChange, FALLING);
}


char receivedCharacter = 0;
int bitsReceived = 0;
int endOfPacket = 0;
String packet;
void loop()
{
  // Check first to see that we have a bit ready to be received.
  if (bitReady == 1) {
    bitReady = 0;
    delayMicroseconds(DELAY_MIDDLE);
    state = digitalRead(2);

    receivedCharacter <<= 1;
    receivedCharacter |= state;
    bitsReceived++;

    // If we have a full byte.
    if (bitsReceived == 8) {

      // If this is the end of data.
      if (receivedCharacter == '\0')
        packet.concat('|');
      else
        packet.concat(receivedCharacter);
```

```
      // Set flag for end of packet.
      // Else if we already have it then this is the first checksum byte
      if (receivedCharacter == '\0')
      {
        endOfPacket = 1;
      }
      else if (endOfPacket)
      {
        // If we have the first, this is the second.
        // Else begin processing.
        if (endOfPacket == 1)
        {
          endOfPacket++;
        }
        else
        {
           unsigned char packetArrayTemp[packet.length()];
           packet.getBytes(packetArrayTemp, packet.length() + 1);

           // Substitute our temp character back to null.
           int j = 0;
           for (j = 0; j < sizeof(packetArrayTemp); j++)
           {
             if (packetArrayTemp[j] == '|')
               packetArrayTemp[j] = '\0';
           }

           // Proper casting.
           char *packetArray = (char *)packetArrayTemp;
           unsigned char *u_packetArray = (unsigned char *) packetArrayTemp;

           // If we pass the checksum.
           if (in_cksum((unsigned short *)u_packetArray,
sizeof(packetArray)))
           {
             int i = 0;

             // Clear LCD > Write last packet to top > get ready to write to
bottom.
             lcd.clear();
             lcd.setCursor(0,0);
             lcd.print(previousPacket);
             lcd.setCursor(0,1);

             previousPacket = "";

             while (packetArray[i] != 0)
             {
               previousPacket += packetArray[i];
               lcd.print(packetArray[i++]);
             }
             previousPacket = packetArray;

             packet = String("");
             endOfPacket = 0;
           }
```

```
            else
            {
              packet = String("");
              endOfPacket = 0;
            }
          }
        }
        bitsReceived = 0;
      }
    }
  }
}

// This is the ISR.  Will change bit ready when it senses the LED is off.
void stateChange()
{
  bitReady = 1;
}

/*
 * in_cksum --
 *      Checksum routine for Internet Protocol family headers (C Version)
 */
unsigned short in_cksum(unsigned short *addr,int len)
{
  register int sum = 0;
  unsigned short answer = 0;
  register unsigned short *w = addr;
  register int nleft = len;

  /*
         * Our algorithm is simple, using a 32 bit accumulator (sum), we add
   * sequential 16 bit words to it, and at the end, fold back all the
   * carry bits from the top 16 bits into the lower 16 bits.
   */
  while (nleft > 1)  {
    sum += *w++;
    nleft -= 2;
  }

  /* mop up an odd byte, if necessary */
  if (nleft == 1) {
    *(unsigned char *)(&answer) = *(unsigned char *)w ;
    sum += answer;
  }

  /* add back carry outs from top 16 bits to low 16 bits */
  sum = (sum >> 16) + (sum & 0xffff);     /* add hi 16 to low 16 */
  sum += (sum >> 16);                     /* add carry */
  answer = ~sum;                          /* truncate to 16 bits */
  return(answer);
}
```