Application Note

Bendzynski, Cronin, and Marino
Rowan University

December 22, 2018

1 Creators

Alex Marino - marinoa2@students.rowan.edu
Cameron Bendzynski - benzynsc4@students.rowan.edu
Will Cronin - croninw9@students.rowan.edu

2 Design Overview

The project goal was to create an image sending and receiving device that took a displayed image on one screen and sent that image to another screen wirelessly. The setup for one node had two processors; the first microprocessor was for image displaying, selection and processing and the second was for sending a signal to the other node to determine which image to use, as well as to read the proximity sensor data. The nodes were connected using a WiFi module, which provided a wireless connection.

2.1 Design Features

These are the design features:

- The ability to cycle through a library of emojis
- Select an emoji to send to a paired board
- Receive an emoji from a paired board
- Send and receive over a wireless connection
- View emoji on OLED display
- Determine user proximity using ultrasonic sensor

2.2 Featured Applications

- Emoji messaging
- Network communication
- Bitmap image display
- Proximity Sensing

2.3 Design Resources

The entire project is stored on the team GitHub repository for ease of access. All files can be found here.

2.4 Block Diagram

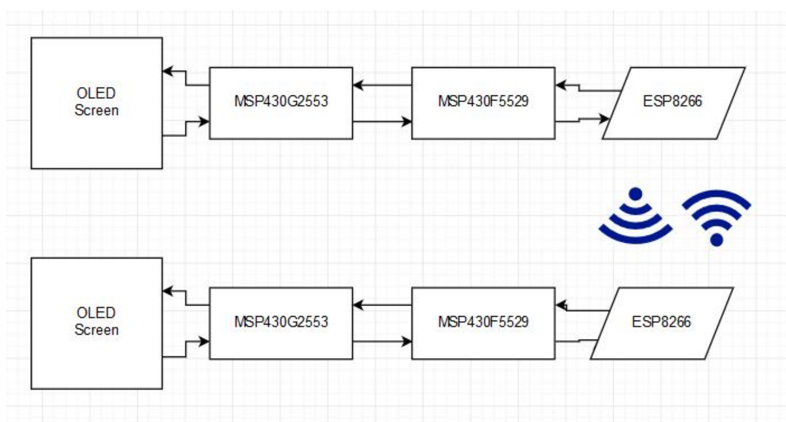

Figure 1: Block Diagram

2.5 Board Image

3 Key System Specifications

4 System Description

The Emoji Sender 3000 is the future of wireless emoji communication. It features a 128 x 64 OLED display with easy-to-use two-button input. Users can cycle through 9 distinct emoji with one button, and send that emoji to another user with the other button. When an emoji is sent, it is delivered wirelessly to another Emoji Sender

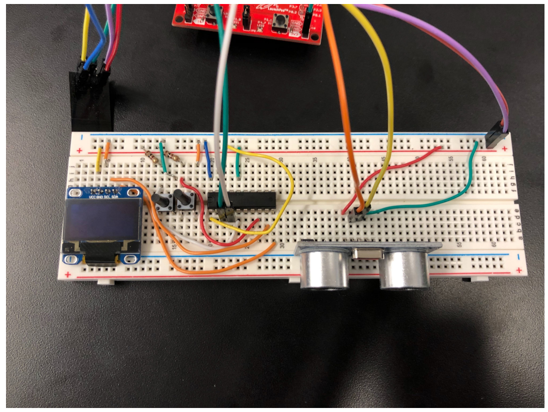

Figure 2: Complete device board for Emoji Sender 3000

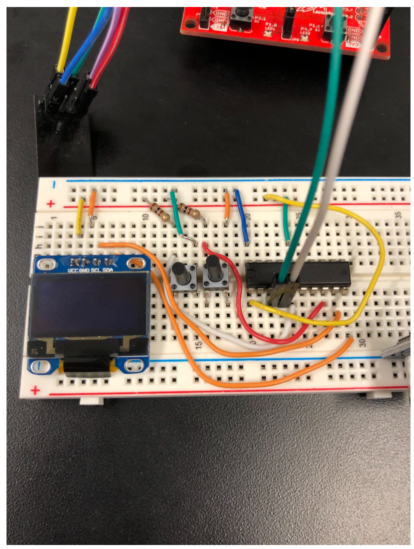

Figure 3: Screen and buttons that control Image selection and sending

3000 user via a peer-to-peer wifi network, meaning no external network connection is required for use. When a user receives an emoji, it is displayed on their screen. In addition, an ultrasonic proximity sensor is installed facing each user. If the receiving user is not present at their device, the sending user will receive a ”user away” message, and the emoji will not be sent, meaning you will never miss an important message.

PARAMETER	SPECIFICATIONS	DETAILS
CCR Limit	0 - 65,535	Limited to 16 bits
Board Input Voltage	5V	
Breadboard Circuit Voltage	5V	
OLED Screen Size	128 x 64	Number of Pixels
UART Rx/Tx Buffer Size	0 - 255	Limited to a byte
User Away Distance	60cm	

Table 1: Key System Specifications

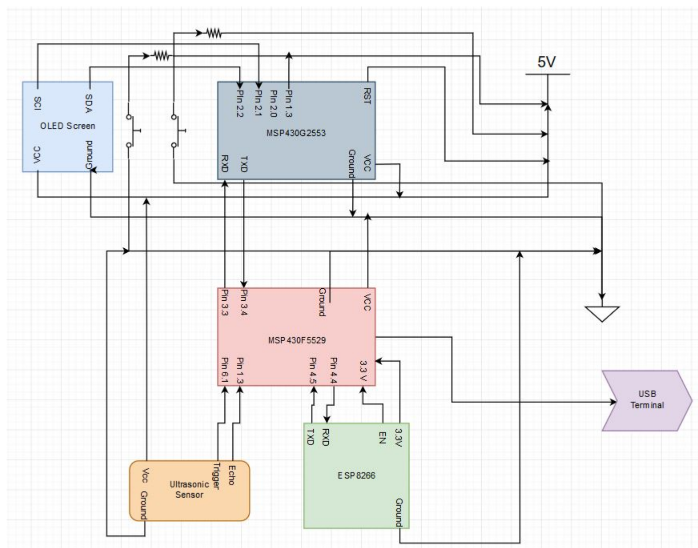

Figure 4: Detailed Block Diagram and Pin Connections of the Emoji Sender 3000

4.1 Detailed Block Diagram

4.2 Highlighted Devices

- MSP430G2553
- MSP430F5529
- ESP8266
- OLED Screen
- HC-SR04 Ultrasonic Proximity Sensor

4.3 OLED Screen: SSD1306

The OLED screen is the primary output of this project. It is driven by the connected G2 as described in the next section. The screen takes a series of hexadecimal values as an input and displays them as pixels that are either on or off as the output. This screen has a resolution of 128 x 64 and runs on either 5V or 3.3V. This implementation used 5V for ease of use, because the proximity sensor described later requires 5V. The screen used in this version of the Emoji Sender 3000 has a yellow strip at the top, while most of the screen is blue. Future versions will either use a monochromatic or RGB screen.

4.4 MSP430G2553

The MSP430G2553 was used as the display driver and interpreter of the system. The SCA and SCL ports were connected to the OLED screen, and the UART connections went to the MSP430F5529, which is explained in the next section. The two input buttons are also connected to the G2. When a user presses the left button, the state of the G2 changes, which cycles to the next emoji to display on the screen. Each emoji was converted from a .bmp image to an array of hex values. These hex values are sent to the screen using Arduino code. The button interrupts are handled with C code. After a user cycles to an emoji they wish to send, pressing the right button will send it. This takes the numerical value of the current emoji and sends it over UART to the MSP430F5529. When this number is received by the other user's G2, a serial communication interrupt written in Arduino changes the current state to display this emoji instead of what was currently being displayed.
The G2 was chosen for this purpose because it is able to be removed from the launchpad and placed in a breadboard. This made it so that the screen and proximity sensor could all be stored on the same breadboard as the G2. Because an addition processor was needed for this implementation, having one of them mounted on the breadboard makes the system much more compact and manageable. The peripherals of the G2 are more limited, but for the display driver, only one UART peripheral and a few general I/O ports were required.

4.5 MSP430F5529

The MSP430F5529 handled UART communication and proximity sensing for the system. Originally, the G2 was chosen for this purpose, so that it could be mounted next to the G2 which was driving the display. However, the F5529 was ultimately chosen because it has two UART peripherals, which was required for this type of implementation. All of the code on the F5529 was written in C. In the main function, a loop is constantly checking whether or not the user is within a certain distance using the proximity sensor. When a message is sent from another user, it arrives on the F5529 via the UCA1 peripheral. The resulting interrupt first checks the status of the proximity sensor. Assuming the user is within range, the received message is sent to the UCA0TXBUF, which is connected to the receive of the connected G2. If the user is not in range, the UCA1TXBUF is populated with the ”user not present” message which is sent back to the sending user, and the sent emoji is never relayed to the connected G2. In this way, the F5529 is the communication middle man, which relays communications based on the status of the proximity sensor. If the F5529 receives a message from its connected G2, it comes through the UCA0 peripheral, and is simply allowed to echo out to the UCA1TXBUF to be processes by the other user's F5529. Communications which leave the F5529 via UCA1TXBUF are sent to the ESP8266 which is described in the next section.

4.6 ESP8266

The ESP8266 is the WiFi module which allows the two Emoji Senders to wirelessly communicate with one another. The device has it's own flash memory, so it only requires 3.3V, ground, and UART receive and transmit connections once the code has been flashed to it. In our project, the ESP8266 modules are used simply as a sort of ”Wireless UART” connection. One system sends its ESP8266 module a number over UART, the module forwards the number to the other module, which then sends that number to its MSP430F5529. The communication is nearly identical from a top-level view, but the underlying principles for the functionality of the ESP8266 network is slightly different for the two modules.

The goal in this project was to configure a peer-to-peer network using the ESP8266 modules to communicate as previously explained. However, ESP8266 modules are typically used as for web servers, and so they are only able to do client-server connections. This was not a problem for our project, however, because there were only ever going to be two devices communicating at once, and a client-server network acts exactly like a peer-to-peer network for two devices when done correctly. One ESP8266 was configured as an access point, setting its own SSID (no password was used for ease of access) so the other ESP8266, which was configured as a station, could connect without the need of an existing network. The access point was also set as the server, which would await a connection from any client and receive/send information once connected. The station ESP8266 was configured as the client, which would always attempt to connect to the server if it wasn't already connected, it wasn't waiting for a message from the server, or it wasn't waiting for a message from its own UART connection.

Once connected to the server, the client and server would both simultaneously await a message from each other or from their own UART connections. Once a message was sent over an ESP8266's UART connection, it would forward this message to the other ESP8266 (first converting it to a string in the format ”/#/”, simply to keep with the syntax of the example code), which would receive it and send it along to the MSP430F5529's UART. After this communication, regardless of which direction, the client would disconnect from the server, loop back to the start of the program, and reconnect to the server. This was done to ensure the connection was updated after every communication. This could then be done indefinitely as long as power was connected to the modules, and communication could travel in both directions.

4.7 HC-SR04

The HC-SR04 is the ultrasonic sensor used in the Emoji Sensor 3000. The sensor is built to detect distance from the sensor to the nearest object up to 60 centimeters away. The ultrasonic sensor is used as a confirmation to send the signal to the other board. If the receiving user is within the distance threshold (60cm by default), the message from the sending user is allowed to transfer. If the receiving user is not within the distance threshold, the sending user will receive a message saying so and the signal will not be transferred.

5 SYSTEM DESIGN THEORY

This system consists of two emoji sending devices. Each device has five main components: G2553, F5529, OLED screen, ESP8266, HC-SR04. This means that the entire system contains two each of these components, because it is designed as a two way peer-to-peer communication system. As described in the following subsections, the overall system has three main requirements: display, sense, and communicate, which it accomplishes using the listed devices. This section goes into more detail about how these design requirements are accomplished through a union of the included devices.

5.1 Design Requirement 1: Display

This requirement is primarily completed through the use of the OLED screen and the MSP430G2553. The G2 stores all of the hex maps of the emojis to be displayed. Each of these is associated with a hexadecimal number, which acts as an address for the hex map array. When an emoji is referenced, the hex map is sent to the screen, and the pixels are told how to light up based on the hex values in the array, thus displaying the requested image. The address is set either by pressing the left button, or when the other user sends an emoji.

5.2 Design Requirement 2: Sense

The requirement to sense is fulfilled by the HC-SR04 which is connected to the MSP430F5529. In this implementation it was used to determine whether or not the receiving user was in front of their Emoji Sender 3000. If the receiver was not in front of their device, the sender would get a ”user not found” message, alerting the sender that the sent emoji did not reach its destination. This way, no one ever misses a message. This simple ultrasonic sensor uses a trigger output and echo input to determine how far away something is. The trigger sends out an ultrasonic pulse which bounces off of any objects in front of it. The echo receives that pulse, which has a different peak to peak value than the sent pulse. These two pulses are compared, and their difference determines how far away the pulse got before it bounced back. In the code, a distance threshold of 60cm was set, such that if the user was not within this range, they would be considered ”away.” This is indicated by the red LED on each respective F5529.

5.3 Design Requirement 3: Communicate

To communicate, the system takes advantage of serial communication on the G2553, 2 UART peripherals on the F5529, and WiFi on the ESP8266. The G2553 communicates with the F5529 with hexadecimal values. These values determine which image will be sent and displayed on the other device. The F5529 uses its second set of UART ports to communicate with the ESP8266 to echo this hexadecimal value and send it to the other device's ESP8266. At this point, the process of transmitting the value goes essentially in reverse ending with the G2 displaying the sent image on the receiving user's screen. When the send button is pressed, the hexidecimal number is sent from the G2 to the F5529 over UART. Once received, the F5529 echos this character to its other UART peripheral, which sends it to the ESP8266. The ESP then sends the character over the peer-to-peer Wifi connection to the other ESP. The peer-to-peer connection is accomplished by one ESP8266 configuring an access point and acting as a server that monitors connections to the set SSID, while the the other ESP8266 connects to the access point as a station and communicates with the server as a client. With only two devices connected, the connection acts exactly like a peer-to-peer network, while it is indeed a client-server network. This ESP then converts the character to a string formatted like ”/#/” (this format was used in the example code for the ESP8266, so this was kept the same for consistency). This string is then converted back to a binary number, and sent over UART to the receiving F5529, which sends it to that user's G2, assuming that user is within proximity. Each G2 is equipped with the same indexed image library, which uses the received value to display the corresponding image.

6 Getting Started/How to use the device

Using the emoji sender is very simple. The first step is to power on the device; the emoji sender uses a USB connection to get power. The next step in using the device is to have another user power on their same emoji sender. The wireless connection will allow the devices to connect automatically. After the power is on and the devices are wirelessly connected, one user can press the left button to cycle through the library of emoticons available and stop on one they wish to send to the other. The user can then press the button on the right to send the message to the other user. The emote will only send if the receiver is close to the device.

7 Getting Started Software/Firmware

The devices are all flashed with all the code needed to function and no other software is needed for the device to operate. However, the MSP430G2553's image library can be updated. The G2 is flashed with Arduino code, but does have some standard C button interrupts. The F5529 is coded entirely in C. Should a user wish to add an image to the library, a regular .png or .jpg must first be converted to a 128x64 .bmp image file. Then, a piece of software which converts from .bmp to hex maps, such as the one found here, should be used to get the hex values. Finally, the header file and case statement in the G2 code should be updated to include the new image.

7.1 Device Specific Information - MSP430G2553's Image Library

While it is not necessary for operation, there is the option of removing or adding more emoticons. This can be done by editing the images.h file. Each image is a 128x64 pixel, monochromatic bitmap represented by 1024 hexadecimal values. By taking an image and converting it through this process, the image(s) can be added to or removed from the image library. The next step would be to open the oled.ino file and add or remove the image name(s) to the array and change the size accordingly. In addition, an additional case in the case statement needs to be added for that location in the array to print that specific image.

8 Test Setup

The components that need to be set up are the F5529 and the ESP8266 for each device. The G2553 is already integrated on the breadboard and does not need to be set up except connecting its RXD and TXD pins to the opposite ones on the F5529. The F5529, in addition to one set of its UART pins being connected to the G2553, is connected to the ultrasonic sensor as well as the ESP8266. The F5529 is also the source of power for the device but needs to be connected to a USB port. The exact pin connections can be seen in the detailed block diagram.

9 Conclusion and Future Work

As the project moves forward, several changes and additions can be made. First, it is likely that a different WiFi adapter would be used. Because the ESP8266 isn't really designed for point to point communication, many problems were discovered and a work-around needed to be implemented. If we wanted to be able to connect more than two users, this implementation would need to change. In addition, the entire circuit is bulky and messy. A PCB should be designed to house all of the necessary components, and the F5529 should be replaced with something that can be surface mounted. The OLED screen should also be replaced with a monochromatic one, and the entire circuit should be battery powered. The precision of the ultrasonic sensors was also questionable during testing, so these may need to be upgraded in a future release as well. Finally, the team would like to add additional emojis, and a kind of ”inbox” where received emojis are stored when the receiving user is detected as being away.

10 Design Files

10.1 Bill of Materials

- 2x MSP430G2553
- 2x MSP430F5529
- 2x ESP8266
- 2x SSD1306 OLED Screen
- 2x HC-SR04 Ultrasonic Proximity Sensor
- 2x Breadboards
- 4x 1kOhm resistors
- 4x Buttons
- Various cables

11 Commented Code

F5529 UART Code:

```
#include <intrinsics.h>
#include <stdint.h>
#include <msp430.h>

#define TRIGGER_PIN BIT1    // P6.1
#define ECHO_PIN BIT3   // P1.3
#define LED_PIN BIT0    // P1.0
#define DISTANCE_THRESHOLD 60   // cm
#define MEASURE_INTERVAL 2048   // ~250 ms
volatile float distance; //current distance
volatile _Bool ON; //bool to check if user is present or not


void triggerMeasurement() {
    static volatile int trigWait;

    // Start trigger
    P6OUT |= TRIGGER_PIN;

    // Wait a small amount of time with trigger high, > 10us required (~10
    clock cycles at 1MHz MCLK)
```

```
    for (trigWait = 0; trigWait < 12; trigWait++) {}

    // End trigger
    P6OUT &= ~TRIGGER_PIN;
}


int main(void)
{
  WDTCTL = WDTPW | WDTHOLD; // stop watchdog timer

  P4SEL |= BIT4 + BIT5;                          //UART A0
    P3SEL |= BIT3 + BIT4;                          // P3.3,4 = USCI_A1 TXD/RXD

    UCA1CTL1 |= UCSWRST;                        // **Put state machine in reset**
    UCA1CTL1 |= UCSSEL_2;                       // SMCLK
    UCA1BR0 = 6;                              // 1MHz 9600 (see User's Guide)
    UCA1BR1 = 0;                                // 1MHz 9600
    UCA1MCTL |= UCBRS_0 + UCBRF_13 + UCOS16;   // Modulation UCBRSx=1, UCBRFx=0
    UCA1CTL1 &= ~UCSWRST;                     // **Initialize USCI state machine**
    UCA1IE |= UCRXIE;                           // Enable USCI_A1 RX interrupt

    UCA0CTL1 |= UCSWRST;                        // **Put state machine in reset**
    UCA0CTL1 |= UCSSEL_2;                       // SMCLK
    UCA0BR0 = 6;                              // 1MHz 9600 (see User's Guide)
    UCA0BR1 = 0;                                // 1MHz 9600
    UCA0MCTL |= UCBRS_0 + UCBRF_13 + UCOS16;   // Modulation UCBRSx=1, UCBRFx=0
    UCA0CTL1 &= ~UCSWRST;                     // **Initialize USCI state machine**
    UCA0IE |= UCRXIE;                           // Enable USCI_A1 RX interrupt

    __bis_SR_register(GIE);        // Enter LPM0, interrupts enabled

    //sensor
    // Configure trigger pin, low to start
    P6DIR |= TRIGGER_PIN;
    P6OUT &= ~TRIGGER_PIN;

    // Configure LED, off to start
    P1DIR |= LED_PIN;
    P1OUT &= ~LED_PIN;

    // Configure echo pin as capture input to TA0CCR2
    P1DIR &= ~ECHO_PIN;
    P1SEL |= ECHO_PIN;

    // Set up TA0 to capture in CCR2 on both edges from P1.3 (echo pin)
    TA0CCTL2 = CM_3 | CCIS_0 | SCS | CAP | CCIE;

    // Set up TA0 to compare CCR0 (measure interval)
    TA0CCR0 = MEASURE_INTERVAL;
    TA0CCTL0 = CCIE;
```

```

    // Set up TA0 with ACLK / 4 = 8192 Hz
    TA0CTL = TASSEL__ACLK | ID__4 | MC__CONTINUOUS | TACLR;

    uint16_t lastCount = 0;
    uint32_t distance = 0;

    while(1)
    {
        triggerMeasurement();

        // Wait for echo start
        __low_power_mode_1();

        lastCount = TA0CCR2;

        // Wait for echo end
        __low_power_mode_1();

        distance = TA0CCR2 - lastCount;
        distance *= 34000; //unit conversion
        distance >>= 14;  // division by 16384 (2 ^ 14)


        if (distance <= DISTANCE_THRESHOLD)
        {
            // Turn on LED
            P1OUT |= LED_PIN;
            ON = 1; //tell MSP user is present
        }
        else
        {
            // Turn off LED
            P1OUT &= ~LED_PIN;
            ON = 0; //tell MSP user is not present
        }

        // Wait for the next measure interval tick
       __low_power_mode_1();
    }

}


#pragma vector=USCI_A1_VECTOR //P4.4 TX, P4.5 RX
__interrupt void USCI_A1_ISR(void)
{

    switch (__even_in_range(UCA1IV, 4))
    {
    case 0:
        break;                              // Vector 0 - no interrupt
    case 2:                                    // Vector 2 - RXIFG
        while (!(UCA1IFG & UCTXIFG));   // USCI_A1 TX buffer ready?
        if (ON) //if user is present
         {
            UCA0TXBUF = UCA1RXBUF; //send message to other UART peripheral
```

```
            }
            else
            {
                UCA1TXBUF = 0x09; //send user not found character
            }

        default:
            break;
    }
}



#pragma vector=USCI_A0_VECTOR //P3.3 TX, P3.4 RX
__interrupt void USCI_A0_ISR(void)
{

    switch (__even_in_range(UCA0IV, 4))
    {
    case 0:
        break;                              // Vector 0 - no interrupt
    case 2:                                 // Vector 2 - RXIFG
        while (!(UCA0IFG & UCTXIFG)); // USCI_A1 TX buffer ready?
        UCA1TXBUF = UCA0RXBUF; //echo receive to other UART peripheral
    default:
        break;
    }
}

#pragma vector = TIMER0_A0_VECTOR
__interrupt void TIMER0_A0_ISR (void) {
    // Measure interval tick
    __low_power_mode_off_on_exit();
    TA0CCR0 += MEASURE_INTERVAL;
}

#pragma vector = TIMER0_A1_VECTOR
__interrupt void TIMER0_A1_ISR (void) {
    // Echo pin state toggled
    __low_power_mode_off_on_exit();
    TA0IV = 0;
}
```

Screen Driver Code:

```
#include <Wire.h>
#include "Font.h"
#include <string.h>
#include "images.h"
#include <msp430.h>
//#include "somethingelse.c"

//extern void somethingelse(void);

#define OLED_Write_Address 0x3C

int i = 0;

```

```
void OLED_Data(char *DATA) /* Function for sending data to OLED */
{
  int len = strlen(DATA);
  for (int g=0; g<len; g++)
  {
    for (int index=0; index<5; index++)
    {
      Wire.beginTransmission(OLED_Write_Address); /* Begin transmission to slave device */
    /* Queue data to be transmitted */
      Wire.write(0x40); /* For Data Transmission, C = 0 and D/C = 1 */
      Wire.write(ASCII[DATA[g] - 0x20][index]);
      Wire.endTransmission(); /* Transmit the queued bytes and end transmission to slave device */
    }
  }
}

void OLED_Command(char DATA) /* Function for sending command to OLED */
{
  Wire.beginTransmission(OLED_Write_Address); /* Begin transmission to slave device */
  /* Queue data to be transmitted */
  Wire.write(0x00); /* For Data Transmission, C = 0 and D/C = 0 */
  Wire.write(DATA);
  Wire.endTransmission(); /* Transmit the queued bytes and end transmission to slave device */
}

void OLED_clear(void) /* Function for clearing OLED */
{
  OLED_setXY(0x00, 0x7F, 0x00, 0x07); /* Column Start Address 0, Column End Address 127, Page Start Address 0, Page End Address 7  */
  for (int k=0; k<=1023; k++)
  {
    Wire.beginTransmission(OLED_Write_Address); /* Begin transmission to slave device */
  /* Queue data to be transmitted */
    Wire.write(0x40); /* For Data Transmission, C = 0 and D/C = 1 */
    Wire.write(0x00);
    Wire.endTransmission(); /* Transmit the queued bytes and end transmission to slave device */
  }
}

void OLED_setXY(char col_start, char col_end, char page_start, char page_end) /* Function for setting cursor for writing data */
{
  Wire.beginTransmission(OLED_Write_Address); /* Begin transmission to slave device */
  /* Queue data to be transmitted */
  Wire.write(0x00); /* For Data Transmission, C = 0 and D/C = 0 */
  Wire.write(0x21); /* Set Column Start and End Address */
  Wire.write(col_start); /* Column Start Address col_start */
  Wire.write(col_end); /* Column End Address col_end */
  Wire.write(0x22); /* Set Page Start and End Address */
  Wire.write(page_start); /* Page Start Address page_start */
```

```
  Wire.write(page_end); /* Page End Address page_end */
  Wire.endTransmission(); /* Transmit the queued bytes and end transmission to slave device */
}

void OLED_init(void) /* Function for initializing OLED */
{
  OLED_Command(0xAE); /* Entire Display OFF */
  OLED_Command(0xD5); /* Set Display Clock Divide Ratio and Oscillator Frequency */
  OLED_Command(0x80); /* Default Setting for Display Clock Divide Ratio and Oscillator Frequency that is recommended */
  OLED_Command(0xA8); /* Set Multiplex Ratio */
  OLED_Command(0x3F); /* 64 COM lines */
  OLED_Command(0xD3); /* Set display offset */
  OLED_Command(0x00); /* 0 offset */
  OLED_Command(0x40); /* Set first line as the start line of the display */
  OLED_Command(0x8D); /* Charge pump */
  OLED_Command(0x14); /* Enable charge dump during display on */
  OLED_Command(0x20); /* Set memory addressing mode */
  OLED_Command(0x00); /* Horizontal addressing mode */
  OLED_Command(0xA1); /* Set segment remap with column address 127 mapped to segment 0 */
  OLED_Command(0xC8); /* Set com output scan direction, scan from com63 to com 0 */
  OLED_Command(0xDA); /* Set com pins hardware configuration */
  OLED_Command(0x12); /* Alternative com pin configuration, disable com left/right remap */
  OLED_Command(0x81); /* Set contrast control */
  OLED_Command(0x80); /* Set Contrast to 128 */
  OLED_Command(0xD9); /* Set pre-charge period */
  OLED_Command(0xF1); /* Phase 1 period of 15 DCLK, Phase 2 period of 1 DCLK */
  OLED_Command(0xDB); /* Set Vcomh deselect level */
  OLED_Command(0x20); /* Vcomh deselect level ~ 0.77 Vcc */
  OLED_Command(0xA4); /* Entire display ON, resume to RAM content display */
  OLED_Command(0xA6); /* Set Display in Normal Mode, 1 = ON, 0 = OFF */
  OLED_Command(0x2E); /* Deactivate scroll */
  OLED_Command(0xAF); /* Display on in normal mode */
}

void OLED_image(const unsigned char *image_data)   /* Function for sending image data to OLED */
{
  OLED_setXY(0x00, 0x7F, 0x00, 0x07);
  for (int k=0; k<=1023; k++)
  {
    Wire.beginTransmission(OLED_Write_Address); /* Begin transmission to slave device */
  /* Queue data to be transmitted */
    Wire.write(0x40); /* For Data Transmission, C = 0 and D/C = 1 */
    Wire.write(image_data[k]);
    Wire.endTransmission(); /* Transmit the queued bytes and end transmission to slave device */
  }
}
```

```
void setup() {
  Wire.begin(); /* Initiate wire library and join I2C bus as a master */
  OLED_init(); /* Initialize OLED */
  delay(100);
  OLED_clear(); /* Clear OLED */
  delay(1000);
  OLED_setXY(0x31, 0x7F, 0x03, 0x02);
  OLED_Data("Emoji");
  OLED_setXY(0x36, 0x7F, 0x04, 0x03);
  OLED_Data("Sender");
  OLED_setXY(0x31, 0x7F, 0x05, 0x04);
  OLED_Data("3000");
  OLED_setXY(0x00, 0x7F, 0x00, 0x08); //Main text that appears when the board is powered
  delay(2000);

  P1DIR |= BIT0;                              // Set P1.0 to output direction
  P1SEL |= 0;

  P1DIR &= ~(BIT3);
  P1IE  |=  BIT3;                             // P1.3 interrupt enabled
  P1IES |= BIT3;                              // P1.3 falling edge
  P1REN |= BIT3;                              // Enable Pull Up on SW2 (P1.3)
  P1IFG &= ~(BIT3);                             // P1.3 IFG cleared

  P2DIR &= ~(BIT0);
  P2IE  |=  BIT0;                             // P2.0 interrupt enabled
  P2IES |= BIT0;                              // P2.0 falling edge
  P2REN |= BIT0;                              // Enable Pull Up on SW2 (P2.0)
  P2IFG &= ~(BIT0);                             // P2.0 IFG cleared

  Serial.begin(9600);
}

void loop()
{
  switch(Serial.read())   //Switch case checks the uart receive and draws the corresponding image to the value received
  {
    case 1:
      delay(1000);
      OLED_image(Eggplant);
      break;
    case 2:
      delay(1000);
      OLED_image(XD);
      break;
    case 3:
      delay(1000);
      OLED_image(MiddleFinger);
      break;
    case 4:
      delay(1000);
      OLED_image(B);
      break;
```

```
    case 5:
      delay(1000);
      OLED_image(Fire);
      break;
    case 6:
      delay(1000);
      OLED_image(Heart);
      break;
    case 7:
      delay(1000);
      OLED_image(Cry);
      break;
    case 8:
      delay(1000);
      OLED_image(Smile);
      break;
    case 9:                    //Receives a 9 if the other user isnt present and prints this message
      delay(1000);
      OLED_setXY(0x31, 0x7F, 0x03, 0x02);
      OLED_Data("Receiver");
      OLED_setXY(0x36, 0x7F, 0x04, 0x03);
      OLED_Data("Not");
      OLED_setXY(0x31, 0x7F, 0x05, 0x04);
      OLED_Data("Found");
      delay(1000);
      OLED_clear();
      break;
    default:
      break;
  }
}


#pragma vector=PORT1_VECTOR
__interrupt void Port_1(void)      //left button interrupt - runs through catalog of images
{

  const unsigned char * image[8] = {Eggplant, XD, MiddleFinger, B, Fire, Heart, Cry, Smile};    //creates arraylist of images
  OLED_image(image[i]);                          //board writes image of a specific location in the arraylist to the screen
  P1IFG &= ~BIT3;                                // P1.3 IFG cleared
  delay(200);
  if(i < 7)                                      //the integer iterates through the list of images
  {
      i++;
  }
  else
  {
    i = 0;
  }
}

#pragma vector=PORT2_VECTOR
```

```
__interrupt void Port_2(void)       //right button interrupt - sends signal to
    the 5529
{
  //P1OUT ^= BIT0;

  Serial.write(i);                  // Sends through the TX pin the value of '
    i'

  i = 0;                            //sets i back to zero so the array starts
    from the beginning

  OLED_clear();
  OLED_setXY(0x31, 0x7F, 0x03, 0x02);
  OLED_Data("Message");
  OLED_setXY(0x36, 0x7F, 0x04, 0x03);
  OLED_Data("Sent");                    //displays a message that the message
    was sent to the 5529

  P2IFG &= ~BIT0;                           // P2.0 IFG cleared
  delay(200);

  OLED_clear();
  OLED_setXY(0x31, 0x7F, 0x03, 0x02);
  OLED_Data("Emoji");
  OLED_setXY(0x36, 0x7F, 0x04, 0x03);
  OLED_Data("Sender");
  OLED_setXY(0x31, 0x7F, 0x05, 0x04);
  OLED_Data("3000");
  OLED_setXY(0x00, 0x7F, 0x00, 0x08);     //default home screen message is
    printed

}


/*#pragma vector=USCI_A0_VECTOR
__interrupt void USCI_A0_ISR(void)
{
  rx = (const unsigned char *) UCA0RXBUF;
  P1OUT ^= BIT0;
  OLED_image(rx);
}*/
```

ESP8266 Client Code:

```
/*  Client
*   This client will turn the LED on or off every 5 seconds
*/
#include <ESP8266WiFi.h>
#include <WiFiClient.h>

const char* ssid = "The_Hardest_Part";  // ssid of access point
const char* password = "";              // password of access point
int port = 1174;                        // port number

byte ip[]= {192,168,4,1};               // gateway address

int ledPin = 2; // GPIO2 of Server ESP8266

```

```
WiFiClient client;  // Declare client

void setup()
{
    // UART configuration
    Serial.begin(9600);
    delay(10);

    pinMode(ledPin, OUTPUT);    // set GPIO 2 as an output for debugging

    WiFi.mode(WIFI_STA);   // set mode to station (client)

    // Connect to WiFi
    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED)
    {
        // Wait for WiFi connection
        delay(500);
    }
}

void loop()
{
    // Connect client to server
    if (client.connect(ip, port))
    {
        digitalWrite(ledPin, LOW); // turn LED off to show connection
    }
    else
    {
        // Retry connection if failure is detected
        return;
    }

    while(!client.available() && !Serial.available())
    {
        // Wait to receive data from server or UART
        delay(500);
        // Return to start if connection breaks
        if(!client.connected() && !client.available())
            return;
    }

    if(Serial.available())
    {
        // Forward received UART character over WiFi
        switch(Serial.read())
        {
            case 1:
                client.println("/1/");
                break;
            case 2:
                client.println("/2/");
                break;
            case 3:
                client.println("/3/");
```

```
                break;
            case 4:
                client.println("/4/");
                break;
            case 5:
                client.println("/5/");
                break;
            case 6:
                client.println("/6/");
                break;
            case 7:
                client.println("/7/");
                break;
            case 8:
                client.println("/8/");
                break;
            case 9:
                client.println("/9/");
                break;
            default:
                break;
        }
    }
    if(client.available())
    {
        // Read the request
        String request = client.readStringUntil('\r');

        // Forward received WiFi character over UART
        if (request.indexOf("/1/") != -1)
        {
            Serial.write(1);
        }
        if (request.indexOf("/2/") != -1)
        {
            Serial.write(2);
        }
        if (request.indexOf("/3/") != -1)
        {
            Serial.write(3);
        }
        if (request.indexOf("/4/") != -1)
        {
            Serial.write(4);
        }
        if (request.indexOf("/5/") != -1)
        {
            Serial.write(5);
        }
        if (request.indexOf("/6/") != -1)
        {
            Serial.write(6);
        }
        if (request.indexOf("/7/") != -1)
        {
            Serial.write(7);
        }
```

```
        if (request.indexOf("/8/") != -1)
        {
            Serial.write(8);
        }
        if (request.indexOf("/9/") != -1)
        {
            Serial.write(9);
        }
    }

    client.stop(); // disconnect from server

    digitalWrite(ledPin, HIGH); // turn LED on to show disconnection

    delay(1);
}
```

ESP8266 Server Code:

```
/* Server
* We write the server first because we need its IP address
*/

#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>

const char* ssid = "The_Hardest_Part";   // ssid of access point
const char* password = "";               // password of access point
int port = 1174;                         // port number
WiFiServer server(port);                 // server configuration

int ledPin = 2; // GPIO2 of Server ESP8266

void setup()
{
    // UART configuration
    Serial.begin(9600);
    delay(10);

    pinMode(ledPin, OUTPUT);   // set GPIO 2 as an output for debugging

    WiFi.mode(WIFI_AP);  // set mode to access point (server)

    // Start WiFi network
    WiFi.softAP(ssid, password);

    // Tell the server to begin listening for incoming connections
    server.begin();
}

void loop()
{
    // Check if a client has connected
    WiFiClient client = server.available();
    if (!client) // if not available, return
    {
        return;
```

```
    }

    // Wait until the client sends some data
    digitalWrite(ledPin, LOW); // turn LED off to show connection
    while(!client.available() & !Serial.available())
    {
        // Wait to receive data from client or UART
        delay(500);
        // Return to start if connection breaks
        if(!client.connected() && !client.available())
            return;
    }

    if(Serial.available())
    {
        // Forward received UART character over WiFi
        switch(Serial.read())
        {
            case 1:
                client.println("/1/");
                break;
            case 2:
                client.println("/2/");
                break;
            case 3:
                client.println("/3/");
                break;
            case 4:
                client.println("/4/");
                break;
            case 5:
                client.println("/5/");
                break;
            case 6:
                client.println("/6/");
                break;
            case 7:
                client.println("/7/");
                break;
            case 8:
                client.println("/8/");
                break;
            case 9:
                client.println("/9/");
                break;
            default:
                break;
        }
    }
    if(client.available())
    {
        // Read the request
        String request = client.readStringUntil('\r');

        // Forward received WiFi character over UART
        if (request.indexOf("/1/") != -1)
        {
```

```
                Serial.write(1);
            }
            if (request.indexOf("/2/") != -1)
            {
                Serial.write(2);
            }
            if (request.indexOf("/3/") != -1)
            {
                Serial.write(3);
            }
            if (request.indexOf("/4/") != -1)
            {
                Serial.write(4);
            }
            if (request.indexOf("/5/") != -1)
            {
                Serial.write(5);
            }
            if (request.indexOf("/6/") != -1)
            {
                Serial.write(6);
            }
            if (request.indexOf("/7/") != -1)
            {
                Serial.write(7);
            }
            if (request.indexOf("/8/") != -1)
            {
                Serial.write(8);
            }
            if (request.indexOf("/9/") != -1)
            {
                Serial.write(9);
            }
        }

        client.stop(); // disconnect client

        digitalWrite(ledPin, HIGH); // turn LED on to show disconnection

        delay(1);
}
```