EEC 172 Lab3

Richard Szeto & Sean Ho

February 22, 2013

1 Objectives

1.1 Part A

The objectives for Part A is to modify the uart echo code to be able to echo back the data that is inputted into X-CTU to the remote XBee. The input and echo data will be displayed on the X-CTU terminal.

1.2 Part B

The objectives for Part B is to successfully send data from the XBee connected to the LM3S8962 to the XBee connected to the LM3S2110. The input will be from the X-CTU terminal. The status LED from the LM3S2110 will turn on when it receives a '1', and will turn off when it receives a '0'.

1.3 Part C

The objectives for Part C is to send text messages back and forth between the LM3S8962 and the LM3S2110. The IR remotes will be used as the typing medium, and the OLED display of the LM3S8962 will display the messages of the LM3S8962 and the LM3S2110 on a split screen. The LM3S8962 will output the Morse code corresponding to the message it receives.

2 Procedures

2.1 Part A

We modified the uart echo code to fit the specifications of the XBee. We used the X-CTU to test the modified uart echo code. Once X-CTU successfully connected to the XBee, we set both XBees to use our designated PAN and channel numbers. That way the two XBees can communicate with each other without interfering with XBees used by other groups. To verify that the two XBees can communicate with each other, we placed a wire going from Data Out to Data In in the remote XBee. This causes the remote XBee to send out the same data as it receives. If the XBee communication works correctly, the XBee connected to X-CTU should also receive the same data as it sent out and the X-CTU terminal should display and echo of the data that was sent.

2.2 Part B

Now that we know both XBees can communicate with each other, we can test if the XBee connected to the X-CTU can communicate with the LM3S2110 through the remote XBee. The LM3S2110 is to be programmed such that the status light turns on if the remote XBee receives a '1', and the status light turns off if the remote XBee receives a '0'. This can be done by having the LM3S2110 parse in the data coming from the Data Out of the remote XBee.

2.3 Part C

To send a sequence of characters from one XBee to another delimited by the `ENTER` key on the IR remote, we needed to set both XBees to API mode and designate each XBee with a different source address. The XBee connected to the LM3S8962 would take the 0x0 address and the XBee connected to the LM3S2110 would take the 0x1 address. Since both XBees are in API mode, we need to send packets conforming to the `Transmit Request:16-bit address` format and receive packets conforming to the `Receive Packet:16-bit address` format. To correctly send a packet, the sender must create a packet that conforms to the `Transmit Request:16-bit address` format and send it to the Data In of the connected XBee. The connected XBee will then automatically create a packet that conforms to the `Receive Packet:16-bit address` format, and send the packet to the Data Out of that XBee and will also send the same packet out into the air. The receiver

XBee will accept the packet from the air and output that packet to the Data Out of that XBee. The connected device will parse through the packet to find the payload. The payload will contain the text message that was sent. The LM3S8962 will display both messages that are sent and received by that device by splitting the OLED display into two halves. The top half will represent the text message that is sent by the LM3S8962 and the bottom half will represent the text message that is received by the LM3S8962 (also the same message that was sent by the LM3S2110). The LM3S2110 will display morse code through the use of the status light for messages that are sent and received by that device.

3 Implementation

Note: Only our Part C code is included in this lab report.

3.1 Part A

3.1.1 LM3S8962

We modified the uart echo code by adding an extra uart interrupt handler for UART1. The original uart echo code already had a uart interrupt handler for UART0, so we did not have to do much modifying for that port. For each uart port, we set the appropriate baud rate, parity, etc. the reflected the specifications of the XBee. We also enabled the hardcoded ports used by UART0 and UART1. To send data out to the XBee, we had to pass data from the UART0 interrupt handler to UART1. To view the echoed data in X-CTU, we had to pass data from the UART1 interrupt handler to UART0. For the UART1 interrupt handler, we did something similar to what was described above.

3.2 Part B

3.2.1 LM3S8962

The same code that was used in Part A was used for Part B.

3.2.2 LM3S2110

We basically copied the code that was used by the LM3S8962 to the LM3S2110. We changed the hardcoded ports that were used by UART0, and there was no UART1 in the LM3S2110, so we removed the corresponding interrupt handler and any of the functions that referred to it. In the while loop of the UART0 interrupt handler, we added an if statement to check for the '1' and '0' characters. If the current character was a '1', then the status light would turn on. If the current character was a '0', then the status light would turn off.

3.3 Part C

3.3.1 LM3S8962

Since Part C involved the IR remote, we integrated our code from Lab2 into our current code for Part C. We modified our lab2 code to include the `LAST` button, since our TV code does not support the `ENTER` button. To reflect API mode, we had to create a character array to store all the pieces of information as seen in the Logic Analyzer picture of the Lab3 prompt. To reduce any problems, we removed the code that sent data to UART0. Instead of using the X-CTU, we used the Logic Analyzer for debugging sent and received packets. To extract data from a packet, we would have to find the "size" and "payload" fields of the packet. The payload field size reflected the "size" field we found in the packet. To output the text messages onto the OLED display, we displayed any sent messages from the LM3S8962 onto the top half of the display, and displayed the payload that was received from the LM3S2110 on the bottom half of the display.

3.3.2 LM3S2110

What was described above for the LM3S8962 will be about the same for the LM3S2110, except we display Morse code on the status light instead of displaying text on a display.

4 Screenshots of XBee Packets

4.1 LM3S8962 is the Sender

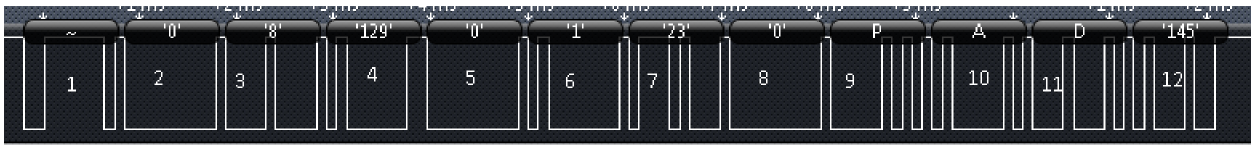

Figure 1: LM3S8962 XBee Data In

Legend

- (1) Start Delimiter
- (2 - 3) Length of Frame Data
- (4 - 11) Frame Data
  - (4) API Identifier
  - (5 - 11) Identifier-specific Data
    - ∗ (5) Frame ID
    - ∗ (6 - 7) Destination Address
    - ∗ (8) Options
    - ∗ (9 - 11) RF Data (Payload)
- (12) Checksum

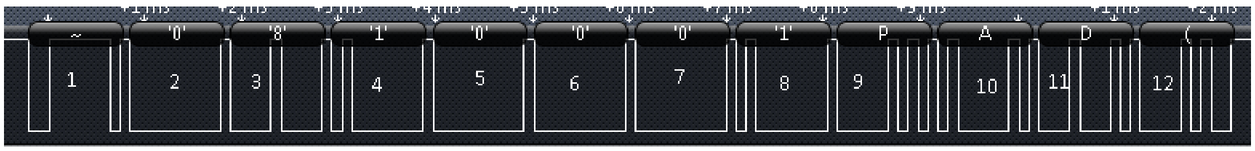

Figure 2: LM3S2110 XBee Data Out

Legend

- (1) Start Delimiter
- (2 - 3) Length of Frame Data
- (4 - 11) Frame Data
  - (4) API Identifier
  - (5 - 11) Identifier-specific Data
    - ∗ (5 - 6) Source Address
    - ∗ (7) Received Signal Strength Indicator

    * (8) Options
    * (9 - 11) RF Data (Payload)

- (12) Checksum

4.2 LM3S2110 is the Sender

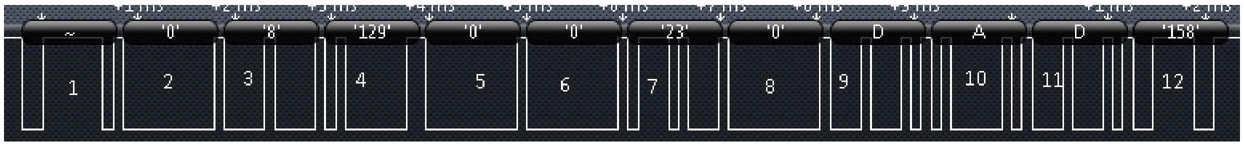

Figure 3: LM3S2110 XBee Data In

Legend

- (1) Start Delimiter
- (2 - 3) Length of Frame Data
- (4 - 11) Frame Data
  - (4) API Identifier
  - (5 - 11) Identifier-specific Data
    * (5) Frame ID
    * (6 - 7) Destination Address
    * (8) Options
    * (9 - 11) RF Data (Payload)
- (12) Checksum

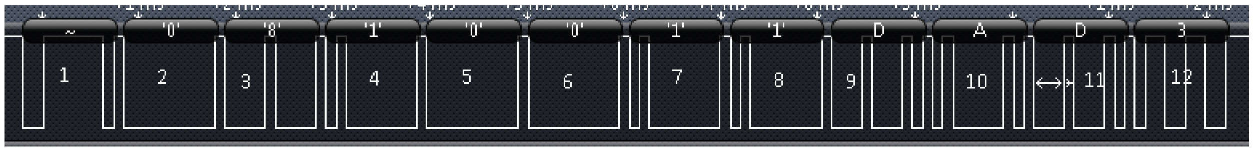

Figure 4: LM3S8962 XBee Data Out

Legend

- (1) Start Delimiter
- (2 - 3) Length of Frame Data
- (4 - 11) Frame Data
  - (4) API Identifier
  - (5 - 11) Identifier-specific Data
    * (5 - 6) Source Address
    * (7) Received Signal Strength Indicator
    * (8) Options
    * (9 - 11) RF Data (Payload)
- (12) Checksum

5 Difficulties

5.1 Part A

Initially we were unsure of the ports that were used by UART0 and UART1 on the LM3S8962. We did not know at the time that Uart ports are hard coded, so we arbitrarily chose 4 unused ports to map Uart to. We later found out that UART0 was mapped to the mini-USB and UART1 was mapped to ports A01 and A02 on the LM3S8962.

5.2 Part B

After finding the correct Uart ports for the LM3S8962, we searched for the hardcoded Uart ports on the LM3S2110. Initially we found that the Uart ports are mapped on the same pins as the Ulink. We found it inconvenient to constantly add and remove the Ulink from the board whenever we wanted to program or test the board. Later we found out that the same Uart ports found in the pins used by the Ulink can also be found in other parts of the chip. Finding those pins were vital to minimize the time it took to finish this portion of the lab.

5.3 Part C

We initially had no clue how the `Transmit Request:16-bit address` and the `Receive Packet:16-bit address` formats worked. The use of trial and error before we understood the required formats wasted a large amount of valuable time. After realizing that the Logic Analyzer picture on the Lab3 prompt represented the correct format, we finally understood what needed to be done. We later found some inconsistent assumptions. The `UARTCharGet` function used in the LM3S8962 returned the bytes in the payload without first going through the bytes before the payload. However, the `UARTCharGet` function used in the LM3S2110 returned the bytes expected from the format. The confusion led to a heated discussion and more wasted time. In the end, we accepted this inconsistency and integrated it into our final code.

6 Miscellaneous

The LM3S8962 OLED display will display an error message corresponding to the device that produced the error. The LM3S8962 is directed connected to the OLED display, so the error message will be displayed in real time if there is an invalid button press. The LM3S8962 does not know about any error button presses on the LM3S2110 until a message is sent to the LM3S8962, so an error message can only be displayed when a message is sent from the LM3S2110 to the LM3S8962.

A Code for Lab3

A.1 LM3S8962

```
//        Lab3 Code for LM3S8962
//        Richard Szeto & Sean Ho

#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/debug.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "drivers/rit128x96x4.h"
```

```
#include "inc/lm3s8962.h"
#include "driverlib/systick.h"

#define CODE_SIZE 16                        // size of sequence from controller

volatile int waitTime;                      // used to determine IR delay
volatile int waitTime2;                     // used to determine delay between button presses
volatile int screenX = 0;                   // OLED display x coordinate
volatile int screenY = 0;                   // OLED display y coordinate
int flag = 0;                               // flag for button press delay
int code[CODE_SIZE];                        // array to hold sequence from controller
int code_offset = 0;                        // placeholder for position in sequence array
char string[CODE_SIZE];                     // array used to convert int to ascii
char display[50];                           // OLED display buffer
int dLocx=0;                                // OLED display buffer offset
char buffer[100];
char display2[50];

//prototypes
char decode (char* input);                  // interpret sequence data
void decodeLetter (char input);             // output corresponding character to OLED display
int getDigit (void);                        // return binary number corresponding to pulse delay


//UART1 interrupt handler
void
UARTIntHandler1 () {
                unsigned long ulStatus;
                unsigned long c;
                int display_offset = 0;
                int i;
                int errorFlag = 0;

    // Get the interrrupt status.
    ulStatus = UARTIntStatus(UART1_BASE, true);


    // Clear the asserted interrupts.
    UARTIntClear(UART1_BASE, ulStatus);

    // Loop while there are characters in the receive FIFO.
    while(UARTCharsAvail(UART1_BASE))
    {
                                c = UARTCharGet(UART1_BASE);


                                display2[display_offset++] = (char)c;

    }

                display2[display_offset-1] = '\0';

                for(i = 0; i < (display_offset - 1); i++)
                {
                        if(display2[i] == 'p')
                        {
                                        errorFlag = 1;
                        }
                }

                if(errorFlag == 1)
                {
                        RIT128x96x4Clear();
                        RIT128x96x4StringDraw("——————————————", 0, 50, 15);
                        RIT128x96x4StringDraw("Error", 50, 75, 15);
                        RIT128x96x4StringDraw(display, 0, 0, 15);
                }
                else
                {
                        RIT128x96x4Clear();
                        RIT128x96x4StringDraw("——————————————", 0, 50, 15);
                        RIT128x96x4StringDraw(display, 0, 0, 15);
                        RIT128x96x4StringDraw(display2, 0, 60, 15);
                }

}

//UART0 interrupt handler
void
UARTIntHandler0(void)
{
    unsigned long ulStatus;

    // Get the interrrupt status.
    ulStatus = UARTIntStatus(UART0_BASE, true);

    // Clear the asserted interrupts.
    UARTIntClear(UART0_BASE, ulStatus);


}


//sends data through UART and then XBee
void
UARTSend(const unsigned char *pucBuffer, unsigned long ulCount)
{

        unsigned char p;
        while(ulCount--)
        {
                p = *pucBuffer;
                *pucBuffer++;


        UARTCharPut(UART1_BASE, p);
        }
}

//create buffer array to send through XBee
void createBuffer ()
{
        unsigned char sum=0, size;
        int i;
        size = dLocx;

        buffer[0]=0x7e;//delimiter
        buffer[1]=0x00;//length
        buffer[2]=size + 5;
        buffer[3]=0x01;//API identifier
        buffer[4]=0x00;//frame id
        buffer[5]=0x00;//desintation address
        buffer[6]=0x01;//desintation address
        buffer[7]=0x01;//options
        for(i=0; i<size; i++)
                buffer[i+8]=display[i];




        for(i=3; i<8+size; i++)
        {
                sum += buffer[i];
        }
        buffer[size+8]= 0xff - sum;

}

//displays Error
void showError (void) {                     // OLED display

        RIT128x96x4StringDraw("Error", 50, 25, 15);

}

// compare last 8 bits of IR pulse sequence
int myStrCmp(char string2[], char given[])
{
        int i;
        for(i=0; (i < 8) && (string2[i] == given[i]); i++);

        if(i && (i == 8) && (given[i]=='\0'))
                        return 1;
        else
                        return 0;
}


//increments waitTime and waitTime2 every period
void SysTickHandler (){
    waitTime += 1;
    waitTime2 += 1;
}


int checkProtocol(){                        // skip repeating pulse sequences

        while(GPIOPinRead(GPIO_PORTB_BASE, GPIO_PIN_1))
        {
        }//90
        if(waitTime <80)
                return 0;
        waitTime=0;
        while (!GPIOPinRead(GPIO_PORTB_BASE, GPIO_PIN_1))
        {
        }
        waitTime=0;
        while(GPIOPinRead(GPIO_PORTB_BASE, GPIO_PIN_1))
        {
        }//24
        if(waitTime <20)
                return 0;
        else
                return 1;
```

```
}

//get data from XBee
void getData()                                  // collect binary numbers
{
        int i, k;
        k =getDigit();
        code_offset=0;
        for(i=0; i<16;i++)
        {
                if(i>=8)
                        string[code_offset++]=k + '0';

                k=getDigit();
        }
}

//get Digit from IR
int getDigit ()                                 // determine binary number from pulse delay
{
        waitTime=0;
        while(!GPIOPinRead(GPIO_PORTB_BASE,GPIO_PIN_1))
        {
        }
        if(waitTime >6)
                return 2;
        waitTime=0;
        while(GPIOPinRead(GPIO_PORTB_BASE,GPIO_PIN_1))
        {
        }
        if(waitTime>14)
                return 1;
        else
                return 0;

}
void PortBIntHandler (void)     // Interrupt handler called upon IR receive
{
        // Reset delay counter
        waitTime=0;

        // Turn on Status light upon first IR receive
        GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_0,0x01);

        // skip repeating IR pulse sequences
        if(!checkProtocol())
                return;

        // 2 second delay between button presses
        if(waitTime2 < 20000)
        {
                flag = 1;
        }
        else
        {
                flag = 0;
        }

        // parse IR pulse sequence data
        getData();

        // concatenate corresponding character to display buffer
        decodeLetter(decode(string));


        // clear interrupt
        GPIOPinIntClear(GPIO_PORTB_BASE, GPIO_PIN_1);

        // reset delay between button presses
        waitTime2=0;
}
int
main(void)
{
        display[0] = '\0';
        display2[0] = '\0';
        // Set the clocking to run directly from the crystal.
        SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN |
                   SYSCTL_XTAL_8MHZ);

        // Initialize the OLED display and write status.
        RIT128x96x4Init(1000000);
        RIT128x96x4StringDraw("----------------------", 0, 50, 15);

        // Enable the peripherals used by this example.

        SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
        SysCtlPeripheralEnable(SYSCTL_PERIPH_UART1);
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);

        // Status
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
        GPIOPadConfigSet(GPIO_PORTF_BASE,GPIO_PIN_0,GPIO_STRENGTH_4MA,GPIO_PIN_TYPE_STD);
        GPIODirModeSet(GPIO_PORTF_BASE,GPIO_PIN_0,GPIO_DIR_MODE_OUT);

        //PB1
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
        GPIOPadConfigSet(GPIO_PORTB_BASE,GPIO_PIN_1,GPIO_STRENGTH_4MA,GPIO_PIN_TYPE_STD);
        GPIODirModeSet(GPIO_PORTB_BASE,GPIO_PIN_1,GPIO_DIR_MODE_IN);

        GPIOPortIntRegister(GPIO_PORTB_BASE, PortBIntHandler);
        GPIOIntTypeSet(GPIO_PORTB_BASE, GPIO_PIN_1, GPIO_BOTH_EDGES);
        GPIOPinIntEnable(GPIO_PORTB_BASE, GPIO_PIN_1);

        IntPrioritySet(INT_GPIOB,0x80);
        SysTickIntRegister(SysTickHandler);
        SysTickPeriodSet(SysCtlClockGet()/10000);       // 0.1ms
        SysTickIntEnable();
        waitTime = 0;                                   // initialize
        waitTime2 = 0;
        SysTickEnable();


        // Enable processor interrupts.
        IntMasterEnable();

        // Set GPIO A0 and A1 as UART pins.
        GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
        GPIOPinTypeUART(GPIO_PORTD_BASE, GPIO_PIN_2 | GPIO_PIN_3);

        // Configure the UART for 115,200, 8-N-1 operation.
        UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 9600,
                        (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
                                UART_CONFIG_PAR_NONE));

        UARTConfigSetExpClk(UART1_BASE, SysCtlClockGet(), 9600,
                                (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
                                UART_CONFIG_PAR_NONE));

         // Enable the UART interrupt.
        IntEnable(INT_UART0);
        UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT);
                IntEnable(INT_UART1);
        UARTIntEnable(UART1_BASE, UART_INT_RX | UART_INT_RT);



        while(1)
        {
        }
}

// interpret sequence data
char decode (char* input) {

        if (myStrCmp(input,"10000100"))
        {
                return '1';
        }
        else if (myStrCmp(input,"01000100"))
        {
           return '2';
        }
        else if (myStrCmp(input,"11000100"))
        {
           return '3';
        }
        else if (myStrCmp(input,"00100100"))
        {
           return '4';
        }
        else if (myStrCmp(input,"10100100"))
        {
           return '5';
        }
        else if (myStrCmp(input,"01100100"))
        {
           return '6';
        }
        else if (myStrCmp(input,"11100100"))
        {
           return '7';
        }
        else if (myStrCmp(input,"00010100"))
        {
           return '8';
        }
        else if (myStrCmp(input,"10010100"))
        {
           return '9';
        }
        else if (myStrCmp(input,"00000100"))
        {
           return '0';
        }
        else if (myStrCmp(input,"10011000"))
        {
           return 'U';
        }
        else if (myStrCmp(input,"11111000"))
        {
           return 'L';
        }
        else if (myStrCmp(input,"01111000"))
        {
           return 'R';
        }
        else if (myStrCmp(input,"00011000"))
        {
           return 'D';
        }
        else if (myStrCmp(input,"01100111")) // Delete
        {
           return 'T';
        }
        else if (myStrCmp(input,"11001001")) // Last
        {
                        return 'S';
        }
        else
        {
                return 'P';
        }
 }

// convert button to character output
 void decodeLetter (char input) {
        if (input == '1')
        {
                display[dLocx++] = '1';
        }
        else if (input == 'U')
        {
                screenY--;
        }
        else if (input == 'D')
        {
                screenY++;
        }
        else if (input == 'L')
        {
                screenX--;
        }
        else if (input == 'R')
        {
                screenX++;
        }
        else if (input == 'T')
        {
                if(dLocx != 0)
                {
                        dLocx--;
                        display[dLocx] = '\0';
                }
        }
        else if (input == 'S')
        {
```

```
                    // output display buffer to OLED display
                    int i;
                    createBuffer();
    UARTSend (buffer, dLocx+9);

                    RIT128x96x4Clear();
                    RIT128x96x4StringDraw("———————————————", 0, 50, 15);
                    RIT128x96x4StringDraw(display, 0, 0, 15);
                    RIT128x96x4StringDraw(display2, 0, 60, 15);

    for(i=0; i<dLocx + 1; i++)
                            display[i]='\0';
    dLocx=0;
    return;
      }
      else if(input == 'P')
      {
              showError();            // Error
      }

      if (dLocx != 0)
      {
              if (input == '2')
              {
                      if (flag == 1)
                      {
                              if (display[dLocx - 1] == 'A')
                                      display[dLocx - 1] = 'B';
                              else if (display[dLocx - 1] == 'B')
                                      display[dLocx - 1] = 'C';
                              else if (display[dLocx - 1] == 'C')
                                      display[dLocx - 1] = '2';
                              else if (display[dLocx - 1] == '2')
                                      display[dLocx - 1] = 'A';
                              else
                                      display[dLocx++] = 'A';

                      }
                      else
                      {
                              display[dLocx++] = 'A';
                      }
              }
              else if (input == '3')
              {
                      if (flag == 1)
                      {
                              if (display[dLocx - 1] == 'D')
                                      display[dLocx - 1] = 'E';
                              else if (display[dLocx - 1] == 'E')
                                      display[dLocx - 1] = 'F';
                              else if (display[dLocx - 1] == 'F')
                                      display[dLocx - 1] = '3';
                              else if (display[dLocx - 1] == '3')
                                      display[dLocx - 1] = 'D';
                              else
                                      display[dLocx++] = 'D';
                      }
                      else
                      {
                              display[dLocx++] = 'D';
                      }
```

```
        }
        else if (input == '4')
        {
                if (flag == 1)
                {
                        if (display[dLocx - 1] == 'G')
                                display[dLocx - 1] = 'H';
                        else if (display[dLocx - 1] == 'H')
                                display[dLocx - 1] = 'I';
                        else if (display[dLocx - 1] == 'I')
                                display[dLocx - 1] = '4';
                        else if (display[dLocx - 1] == '4')
                                display[dLocx - 1] = 'G';
                        else
                                display[dLocx++] = 'G';
                }
                else
                {
                        display[dLocx++] = 'G';
                }
        }
        else if (input == '5')
        {
                if (flag == 1)
                {
                        if (display[dLocx - 1] == 'J')
                                display[dLocx - 1] = 'K';
                        else if (display[dLocx - 1] == 'K')
                                display[dLocx - 1] = 'L';
                        else if (display[dLocx - 1] == 'L')
                                display[dLocx - 1] = '5';
                        else if (display[dLocx - 1] == '5')
                                display[dLocx - 1] = 'J';
                        else
                                display[dLocx++] = 'J';
                }
                else
                {
                        display[dLocx++] = 'J';
                }
        }
        else if (input == '6')
        {
                if (flag == 1)
                {
                        if (display[dLocx - 1] == 'M')
                                display[dLocx - 1] = 'N';
                        else if (display[dLocx - 1] == 'N')
                                display[dLocx - 1] = 'O';
                        else if (display[dLocx - 1] == 'O')
                                display[dLocx - 1] = '6';
                        else if (display[dLocx - 1] == '6')
                                display[dLocx - 1] = 'M';
                        else
                                display[dLocx++] = 'M';
                }
                else
                {
                        display[dLocx++] = 'M';
                }
        }
        else if (input == '7')
```

```
{
        if (flag == 1)
        {
                if (display[dLocx - 1] == 'P')
                        display[dLocx - 1] = 'Q';
                else if (display[dLocx - 1] == 'Q')
                        display[dLocx - 1] = 'R';
                else if (display[dLocx - 1] == 'R')
                        display[dLocx - 1] = 'S';
                else if (display[dLocx - 1] == 'S')
                        display[dLocx - 1] = '7';
                else if (display[dLocx - 1] == '7')
                        display[dLocx - 1] = 'P';
                else
                        display[dLocx++] = 'P';
        }
        else
        {
                display[dLocx++] = 'P';
        }
}
else if (input == '8')
{
        if (flag == 1)
        {
                if (display[dLocx - 1] == 'T')
                        display[dLocx - 1] = 'U';
                else if (display[dLocx - 1] == 'U')
                        display[dLocx - 1] = 'V';
                else if (display[dLocx - 1] == 'V')
                        display[dLocx - 1] = '8';
                else if (display[dLocx - 1] == '8')
                        display[dLocx - 1] = 'T';
                else
                        display[dLocx++] = 'T';
        }
        else
        {
                display[dLocx++] = 'T';
        }
}
else if (input == '9')
{
        if (flag == 1)
        {
                if (display[dLocx - 1] == 'W')
                        display[dLocx - 1] = 'X';
                else if (display[dLocx - 1] == 'X')
                        display[dLocx - 1] = 'Y';
                else if (display[dLocx - 1] == 'Y')
                        display[dLocx - 1] = 'Z';
                else if (display[dLocx - 1] == 'Z')
                        display[dLocx - 1] = '9';
                else if (display[dLocx - 1] == '9')
                        display[dLocx - 1] = 'W';
                else
                        display[dLocx++] = 'W';
        }
        else
        {
                display[dLocx++] = 'W';
        }
```

```
                }
                else if (input == '0')
                {
                        if (flag == 1)
                        {
                                if (display[dLocx - 1] == '␣')
                                        display[dLocx - 1] = '0';
                                else if (display[dLocx - 1] == '0')
                                        display[dLocx - 1] = '␣';
                                else
                                        display[dLocx++] = '␣';
                        }
                        else
                        {
                                display[dLocx++] = '␣';
                        }
                }
        }
        else
        {
                if (input == '2')
                {
                        display[dLocx++] = 'A';
                }
                else if (input == '3')
                {
                        display[dLocx++] = 'D';
                }
                else if (input == '4')
                {
                        display[dLocx++] = 'G';
                }
                else if (input == '5')
                {
                        display[dLocx++] = 'J';
                }
                else if (input == '6')
                {
                        display[dLocx++] = 'M';
                }
                else if (input == '7')
                {
                        display[dLocx++] = 'P';
                }
                else if (input == '8')
                {
                        display[dLocx++] = 'T';
                }
                else if (input == '9')
                {
                        display[dLocx++] = 'W';
                }
                else if (input == '0')
                {
                        display[dLocx++] = '␣';
                }
        }
        display[dLocx] = '\0';
}
```

A.2 LM3S2110

```
//  Lab3 Code for LM3S2110
//  Richard Szeto & Sean Ho

#include "inc/lm3s8962.h"
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/debug.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/sysctl.h"
#include "drivers/rit128x96x4.h"
#include "driverlib/systick.h"
#include "driverlib/uart.h"

#define CODE_SIZE 16                        // size of sequence from controller

volatile int waitTime;                      // used to determine IR delay
volatile int waitTime2;                     // used to determine delay between button presses
volatile int morseCounter;        // used to time morse code
volatile int screenX = 12;                  // OLED display x coordinate
volatile int screenY = 60;                  // OLED display y coordinate
int flag = 0;                               // flag for button press delay
int code[CODE_SIZE];                        // array to hold sequence from controller
int code_offset = 0;                        // placeholder for position in sequence array
char string[CODE_SIZE];                     // array used to convert int to ascii
char display[50];                           // OLED display buffer
char display2[50];
char dLocx=0;                               // OLED display buffer offset
char buffer[100];

//prototypes
char decode (char* input);                  // interpret sequence data
void decodeLetter (char input);             // output corresponding character to OLED display
int getDigit (void);                        // return binary number corresponding to pulse delay
void morseLetter (char a);                  //convert letter to morse

//create a space in between dots and dashes
void space ()
{

        morseCounter = 0;
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2,0x00);
        while(morseCounter <10000)
        {
        }

}

//status light dash
void dash ()
{

        morseCounter=0;
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2,0x04);
        while(morseCounter < 15000)
        {
        }

        space();
```

```
}

//status light dot
void dot ()
{


        morseCounter=0;
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2,0x04);
        while(morseCounter<5000)
        {
        }

        space();
}



int myStrCmp(char string2[], char given[])      // compare last 8 bits of IR pulse sequence
{
        int i;
        for(i=0; (i < 8) && (string2[i] == given[i]); i++);

        if(i && (i == 8) && (given[i]=='\0'))
                        return 1;
        else
                        return 0;
}

//converts sting to display
void morseDisplay (char *string, int length)
{
        int i;
        for (i=0; i<length; i++)
        {

                morseLetter(string[i]);
        }
        space();
        space();

}

//increments waitTime and waitTime2 every period
void SysTickHandler (){
    waitTime += 1;
                waitTime2 += 1;
                morseCounter +=1;
}




//create buffer array to be sent through Xbee
void createBuffer ()
{
        unsigned char sum=0, size;
        int i;
        size = dLocx;

        buffer[0]=0x7e;//delimiter
        buffer[1]=0x00;//length msb
        buffer[2]=size + 5;//length lsb
        buffer[3]=0x01;//API identifier
        buffer[4]=0x00;//frame id
        buffer[5]=0x00;//desintation address
        buffer[6]=0x00;//desintation address
        buffer[7]=0x01;//options
        for(i=0; i<size; i++)
                buffer[i+8]=display[i];







        for(i=3; i<8+size; i++)
        {
                sum += buffer[i];
        }
        buffer[size+8]= 0xff - sum;

}
int checkProtocol(){                    // skip repeating pulse sequences

        while(GPIOPinRead(GPIO_PORTB_BASE, GPIO_PIN_1))
        {
        }//90
        if(waitTime <80)
                return 0;
        waitTime=0;
        while(!GPIOPinRead(GPIO_PORTB_BASE, GPIO_PIN_1))
        {
        }
        waitTime=0;
        while(GPIOPinRead(GPIO_PORTB_BASE, GPIO_PIN_1))
        {
        }//24
        if(waitTime <20)
                return 0;
        else
                return 1;

}
void getData()                                  // collect binary numbers
{
        int i, k;
        k =getDigit();
        code_offset=0;
        for(i=0; i<16;i++)
        {
                if(i>=8)
                        string[code_offset++]=k + '0';

                k=getDigit();
        }
}

// determine binary number from pulse delay
int getDigit ()
{
        waitTime=0;
        while(!GPIOPinRead(GPIO_PORTB_BASE, GPIO_PIN_1))
        {
        }
        if(waitTime >6)
                return 2;
        waitTime=0;
        while(GPIOPinRead(GPIO_PORTB_BASE, GPIO_PIN_1))
        {
        }
        if(waitTime>14)
                return 1;
        else
                return 0;

}

//sends data through UART and then through XBee
void
UARTSend(const unsigned char *pucBuffer, unsigned long ulCount)
{
    //
    // Loop while there are more characters to send.
    //
                int i;
                int size = pucBuffer[2] - 5;
                char stringArray[50];
                int errorFlag = 0;



                for(i = 8; i < size + 8; i++)
                {
                        stringArray[i - 8] = pucBuffer[i];
                        if(pucBuffer[i] == 'p')
                                errorFlag = 1;
                }



    while(ulCount--)
    {

        //
        // Write the next character to the UART.
        //
       UARTCharPut(UART0_BASE, *pucBuffer++);
    }
                if(errorFlag == 0)
                        morseDisplay(stringArray,size);
}

// Interrupt handler called upon IR receive
void PortBIntHandler (void)
{

        unsigned long ulStatus;
        // Reset delay counter
        waitTime=0;

        // skip repeating IR pulse sequences
        if(!checkProtocol())
                return;

        // 2 second delay between button presses
        if(waitTime2 < 20000)
        {
                flag = 1;
        }
        else
        {
                flag = 0;
        }


        // parse IR pulse sequence data
        getData();

        // concatenate corresponding character to display buffer
        decodeLetter(decode(string));


        // clear interrupt
        GPIOPinIntClear(GPIO_PORTB_BASE, GPIO_PIN_1);
     ulStatus = UARTIntStatus(UART0_BASE, true);
    UARTIntClear(UART0_BASE, ulStatus);

        // reset delay between button presses
        waitTime2=0;
}

//UART interrupt handler
void UARTIntHandler0 ()
{
                unsigned long ulStatus;
                unsigned long c;
                int display_offset = 0;
                int buffer_offset = 0;
                int size;

    //
    // Get the interrrupt status.
    //
    ulStatus = UARTIntStatus(UART0_BASE, true);

    //
    // Clear the asserted interrupts.
    //
    UARTIntClear(UART0_BASE, ulStatus);

    //
    // Loop while there are characters in the receive FIFO.
    //
    while(UARTCharsAvail(UART0_BASE))
    {
                                c = UARTCharGet(UART0_BASE);

                                //
        // Read the next character from the UART and write it back to the UART.
        //
                                if(buffer_offset == 2)
                                {
                                        size = c - 5;
                                }

                                if(buffer_offset >= 8)
                                {
                                                display2[display_offset++] = (char)c;
                                        }
                                        buffer_offset++;

    }
                display2[size] = '\0';

                morseDisplay(display2, size);

                GPIOPinIntClear(GPIO_PORTB_BASE, GPIO_PIN_1);
}



int
main(void)
{



        //set clock
        SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN |
                    SYSCTL_XTAL_8MHZ);





      //PB1
      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
      GPIOPadConfigSet(GPIO_PORTB_BASE, GPIO_PIN_1, GPIO_STRENGTH_4MA, GPIO_PIN_TYPE_STD);
      GPIODirModeSet(GPIO_PORTB_BASE, GPIO_PIN_1, GPIO_DIR_MODE_IN);
      GPIOPortIntRegister(GPIO_PORTB_BASE, PortBIntHandler);
      GPIOIntTypeSet(GPIO_PORTB_BASE, GPIO_PIN_1, GPIO_BOTH_EDGES);
      GPIOPinIntEnable(GPIO_PORTB_BASE, GPIO_PIN_1);


    // Status
                SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    GPIOPadConfigSet(GPIO_PORTF_BASE, GPIO_PIN_2, GPIO_STRENGTH_4MA, GPIO_PIN_TYPE_STD);
        GPIODirModeSet(GPIO_PORTF_BASE, GPIO_PIN_2, GPIO_DIR_MODE_OUT);

    //UART
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
    UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 9600,
    (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
                    UART_CONFIG_PAR_NONE));
    IntEnable(INT_UART0);
    UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT);

    IntPrioritySet(INT_UART0, 0x7F);

    IntPrioritySet(INT_GPIOB,0x80);
    IntMasterEnable();
    SysTickIntRegister(SysTickHandler);
    SysTickPeriodSet(SysCtlClockGet()/10000);    // 0.1ms
    SysTickIntEnable();
    waitTime = 0;                    // initialize
    waitTime2 = 0;
    SysTickEnable();
    while(1)
```

```
   {


   }

}

// interpret sequence data
char decode (char* input) {

        if (myStrCmp(input,"10000100"))
        {
                return '1';
        }
        else if (myStrCmp(input,"01000100"))
        {
           return '2';
        }
        else if (myStrCmp(input,"11000100"))
        {
           return '3';
        }
        else if (myStrCmp(input,"00100100"))
        {
           return '4';
        }
        else if (myStrCmp(input,"10100100"))
        {
           return '5';
        }
        else if (myStrCmp(input,"01100100"))
        {
           return '6';
        }
        else if (myStrCmp(input,"11100100"))
        {
           return '7';
        }
        else if (myStrCmp(input,"00010100"))
        {
           return '8';
        }
        else if (myStrCmp(input,"10010100"))
        {
           return '9';
        }
        else if (myStrCmp(input,"00000100"))
        {
           return '0';
        }
        else if (myStrCmp(input,"10011000"))
        {
           return 'U';
        }
        else if (myStrCmp(input,"11111000"))
        {
           return 'L';
        }
        else if (myStrCmp(input,"01111000"))
        {
           return 'R';
        }
        else if (myStrCmp(input,"00011000"))
        {
           return 'D';
        }
        else if (myStrCmp(input,"01100111")) // Delete
        {
           return 'T';
        }
        else if (myStrCmp(input,"11001001")) // Last
        {
                         return 'S';
        }
        else
        {
                return 'P';
        }
 }

// convert button to character output
 void decodeLetter (char input) {
         int i;
        if (input == '1')
        {
                display[dLocx++] = '1';
        }
        else if (input == 'U')
        {
                screenY--;
        }
        else if (input == 'D')
        {
                screenY++;
        }
        else if (input == 'L')
        {
                screenX--;
        }
        else if (input == 'R')
        {
                screenX++;
        }
        else if (input == 'T')
        {
                if(dLocx != 0)
                {
                        dLocx--;
                        display[dLocx] = '\0';
                }
        }
        else if (input == 'S')
        {
                createBuffer();
                UARTSend (buffer, dLocx+9);
                for(i=0; i<dLocx + 1; i++)
                        display[i]='\0';
                dLocx=0;
                return;
        }
        else if(input == 'P')
        {
                                // Error
                display[dLocx++] = 'p';
        }

        if (dLocx != 0)
        {
                if (input == '2')
                {
                        if (flag == 1)
                        {
                                if (display[dLocx - 1] == 'A')
                                        display[dLocx - 1] = 'B';
                                else if (display[dLocx - 1] == 'B')
                                        display[dLocx - 1] = 'C';
                                else if (display[dLocx - 1] == 'C')
                                        display[dLocx - 1] = '2';
                                else if (display[dLocx - 1] == '2')
                                        display[dLocx - 1] = 'A';
                                else
                                        display[dLocx++] = 'A';

                        }
                        else
                        {
                                display[dLocx++] = 'A';
                        }
                }
                else if (input == '3')
                {
                        if (flag == 1)
                        {
                                if (display[dLocx - 1] == 'D')
                                        display[dLocx - 1] = 'E';
                                else if (display[dLocx - 1] == 'E')
                                        display[dLocx - 1] = 'F';
                                else if (display[dLocx - 1] == 'F')
                                        display[dLocx - 1] = '3';
                                else if (display[dLocx - 1] == '3')
                                        display[dLocx - 1] = 'D';
                                else
                                        display[dLocx++] = 'D';
                        }
                        else
        efficency               {
                                display[dLocx++] = 'D';
                        }
                }
                else if (input == '4')
                {
                        if (flag == 1)
                        {
                                if (display[dLocx - 1] == 'G')
                                        display[dLocx - 1] = 'H';
                                else if (display[dLocx - 1] == 'H')
                                        display[dLocx - 1] = 'I';
                                else if (display[dLocx - 1] == 'I')
                                        display[dLocx - 1] = '4';
                                else if (display[dLocx - 1] == '4')
                                        display[dLocx - 1] = 'G';
                                else
                                        display[dLocx++] = 'G';
                        }
                        else
                        {
```

```
                        display[dLocx++] = 'G';
                }
        }
        else if (input == '5')
        {
                if (flag == 1)
                {
                        if (display[dLocx - 1] == 'J')
                                display[dLocx - 1] = 'K';
                        else if (display[dLocx - 1] == 'K')
                                display[dLocx - 1] = 'L';
                        else if (display[dLocx - 1] == 'L')
                                display[dLocx - 1] = '5';
                        else if (display[dLocx - 1] == '5')
                                display[dLocx - 1] = 'J';
                        else
                                display[dLocx++] = 'J';
                }
                else
                {
                        display[dLocx++] = 'J';
                }
        }
        else if (input == '6')
        {
                if (flag == 1)
                {
                        if (display[dLocx - 1] == 'M')
                                display[dLocx - 1] = 'N';
                        else if (display[dLocx - 1] == 'N')
                                display[dLocx - 1] = 'O';
                        else if (display[dLocx - 1] == 'O')
                                display[dLocx - 1] = '6';
                        else if (display[dLocx - 1] == '6')
                                display[dLocx - 1] = 'M';
                        else
                                display[dLocx++] = 'M';
                }
                else
                {
                        display[dLocx++] = 'M';
                }
        }
        else if (input == '7')
        {
                if (flag == 1)
                {
                        if (display[dLocx - 1] == 'P')
                                display[dLocx - 1] = 'Q';
                        else if (display[dLocx - 1] == 'Q')
                                display[dLocx - 1] = 'R';
                        else if (display[dLocx - 1] == 'R')
                                display[dLocx - 1] = 'S';
                        else if (display[dLocx - 1] == 'S')
                                display[dLocx - 1] = '7';
                        else if (display[dLocx - 1] == '7')
                                display[dLocx - 1] = 'P';
                        else
                                display[dLocx++] = 'P';
                }
                else
                {
                    display[dLocx++] = 'P';
                }
            }
            else if (input == '8')
            {
                if (flag == 1)
                {
                    if (display[dLocx - 1] == 'T')
                        display[dLocx - 1] = 'U';
                    else if (display[dLocx - 1] == 'U')
                        display[dLocx - 1] = 'V';
                    else if (display[dLocx - 1] == 'V')
                        display[dLocx - 1] = '8';
                    else if (display[dLocx - 1] == '8')
                        display[dLocx - 1] = 'T';
                    else
                        display[dLocx++] = 'T';
                }
                else
                {
                    display[dLocx++] = 'T';
                }
            }
            else if (input == '9')
            {
                if (flag == 1)
                {
                    if (display[dLocx - 1] == 'W')
                        display[dLocx - 1] = 'X';
                    else if (display[dLocx - 1] == 'X')
                        display[dLocx - 1] = 'Y';
                    else if (display[dLocx - 1] == 'Y')
                        display[dLocx - 1] = 'Z';
                    else if (display[dLocx - 1] == 'Z')
                        display[dLocx - 1] = '9';
                    else if (display[dLocx - 1] == '9')
                        display[dLocx - 1] = 'W';
                    else
                        display[dLocx++] = 'W';
                }
                else
                {
                    display[dLocx++] = 'W';
                }
            }
            else if (input == '0')
            {
                if (flag == 1)
                {
                    if (display[dLocx - 1] == '␣')
                        display[dLocx - 1] = '0';
                    else if (display[dLocx - 1] == '0')
                        display[dLocx - 1] = '␣';
                    else
                        display[dLocx++] = '␣';
                }
                else
                {
                    display[dLocx++] = '␣';
                }
            }
        }
        else
        {
                if (input == '2')
                {
                        display[dLocx++] = 'A';
                }
                else if (input == '3')
                {
                        display[dLocx++] = 'D';
                }
                else if (input == '4')
                {
                        display[dLocx++] = 'G';
                }
                else if (input == '5')
                {
                        display[dLocx++] = 'J';
                }
                else if (input == '6')
                {
                        display[dLocx++] = 'M';
                }
                else if (input == '7')
                {
                        display[dLocx++] = 'P';
                }
                else if (input == '8')
                {
                        display[dLocx++] = 'T';
                }
                else if (input == '9')
                {
                        display[dLocx++] = 'W';
                }
                else if (input == '0')
                {
                        display[dLocx++] = ' ';
                }
        }
        display[dLocx] = '\0';
}

void morseLetter (char a)
{
        if(a=='A')
        {
                dot();
                dash();
        }else if (a=='B')
        {
                dash();
                dot();
                dot();
                dot();

        }else if (a=='C')
        {
                dash();
                dot();
                dash();
                dot();
        }else if (a=='D')
        {
                dash();
                dot();
                dot();

        }else if (a=='E')
        {
                dot();
        }else if (a=='F')
        {
                dot();
                dot();
                dash();
                dot();
        }else if (a=='G')
        {
                dash();
                dash();
                dot();
        }else if (a=='H')
        {
                dot();
                dot();
                dot();
                dot();
        }else if (a=='I')
        {
                dot();
                dot();
        }else if (a=='J')
        {
                dot();
                dash();
                dash();
                dash();
        }else if (a=='K')
        {
                dash();
                dot();
                dash();
        }else if (a=='L')
        {
                dot();
                dash();
                dot();
                dot();
        }else if (a=='M')
        {
                dash();
                dash();
        }else if (a=='N')
        {
                dash();
                dot();
        }else if (a=='O')
        {
                dash();
                dash();
                dash();
        }else if (a=='P')
        {
                dot();
                dash();
                dash();
                dot();
        }else if (a=='Q')
        {
                dash();
                dash();
                dot();
                dash();
        }else if (a=='R')
        {
                dot();
                dash();
                dot();
        }else if (a=='S')
        {
                dot();
                dot();
                dot();
        }else if (a=='T')
        {
                dash();
        }else if (a=='U')
        {
                dot();
                dot();
                dash();
        }else if (a=='V')
        {
                dot();
                dot();
                dot();
                dash();
        }else if (a=='W')
        {
                dot();
                dash();
                dash();
        }else if (a=='X')
        {
                dash();
                dot();
                dot();
                dash();
        }else if (a=='Y')
        {
                dash();
                dot();
                dash();
                dash();
        }else if (a=='Z')
        {
                dash();
                dash();
                dot();
                dot();
        }else if (a=='1')
        {
                dot();
                dash();
                dash();
                dash();
                dash();
        }else if (a=='2')
        {
                dot();
                dot();
                dash();
                dash();
                dash();
        }else if (a=='3')
        {
                dot();
                dot();
                dot();
                dash();
                dash();
        }else if (a=='4')
        {
                dot();
                dot();
                dot();
                dot();
                dash();
        }else if (a=='5')
        {
                dot();
                dot();
                dot();
                dot();
                dot();
        }else if (a=='6')
        {
                dash();
                dot();
                dot();
                dot();
                dot();
        }else if (a=='7')
        {
                dash();
                dash();
                dot();
                dot();
                dot();
        }else if (a=='8')
        {
                dash();
                dash();
                dash();
                dot();
                dot();
        }else if (a=='9')
        {
                dash();
                dash();
                dash();
                dash();
                dot();
        }else if (a=='0')
        {
                dash();
                dash();
                dash();
                dash();
                dash();
        }
        else if (a == '␣')
        {
                space();
                space();
        }

}
```