EEC 172 Lab4

Richard Szeto & Sean Ho

Wednesday, March 6, 2013

1 Objectives

1.1 Part I

The objectives of part I is to generate two valid PWM signals in the LM3S2110 and be able to use the IR remote to alter the PWM signal within the legal bounds. The left and right buttons will control one of the PWM signals, and the up and down buttons will control the other PWM signal.

1.2 Part II

The objectives of part II is to use the PWM signals produced in part I to control the servo by using the IR remote to alter the PWM signal within the legal bounds. The left and right buttons will control the pan rotation, and the up and down buttons will control the tilt rotation.

1.3 Part III

The objectives of part III is to control the PWM signals in the LM3S2110 by using the IR remote to send XBee packets from the LM3S8962 to the LM3S2110. The packets sent will have values corresponding to the pan and tilt rotations.

2 Procedures

2.1 Part I

We used the pwmgen code given in the Stellaris folder as a template. To accomodate for the LM3S2110, we changed the output ports of the PWM signal to PF0 and PF1. An mathematical calculation was used to produce two 1.5ms control pules with frequencies of 50Hz. After confirming that our control signals match these specification through the use of the logic analyzer, we incorporated Lab3's code to allow the use of the IR remote. An Algorithm was used to determine how much of the control signal would be changed each time the up, down, left, and right buttons are pressed. When the up/down buttons are pressed, the PWM signal on channel 1 will decrease/increase by 0.1ms. The PWM signal on channel 1 has an upper bound of 2.0ms and a lower bound of 1.0ms. When the left/right buttons are pressed, the PWM signal on channel 2 will decrease/increase by 0.05ms. The PWM signal on channel 2 has an upper bound of 2.0ms and a lower bound of 1.0ms.

2.2 Part II

We needed to convert the PWM pulse widths to map to the angles available to the servo. The pan rotation angles range from 0°to 360°, and the tilt rotation angles range from 0°to 120°. An algorithm was to convert angles to the respective PWM pulse widths for each signal. If a provided angle maps to a pulse width outside of the servo range, the lower bound or upper bound will be substituted as the pulse width instead to avoid damaging the servo.

2.3 Part III

The Uart handlers were copied from Lab3 to Lab4 to allow the use of XBees. Since the characters from the alphabet are not needed for Lab4, we disabled interpreting those characters when the corresponding button was pressed on the IR remote. Two numbers will have to be sent through the XBees, the pan and tilt angles. The first input will be used for the pan angle, and the second input will be used for the tilt angle. The ' ' character will be used as the delimiter between the two inputs. The servo can be moved by either manually typing the angles of the pan and tilt, or use the arrow buttons to incrementally change the pan or tilt. Initially the default angle is saved to allow the arrow buttons to send angle packets. When the arrow buttons are

pressed, they will increment/decrement the stored angle and send those angles as XBee packets. We were not able to allow continuous angle changes when an arrow button is held.

3 Algorithm

The core of our algorithm is based on receiving two numbers in a string delimited by a space in the 2110 XBEE, then converting the values into appropriate PWM value in the servos. There are 98 different positions the horizontal servos can take and 100 different vertical positions. The 2110 converts incoming strings into integers in degrees, then converts them into the appropriate position and then calculates the correct PWM signal to transmit.

The 8962 takes all IR input and converts them into the correct angle. It then converts the angle into a string which is transmitted via the XBEE. The 8962 keeps track of the 2110 position by keeping track of the strings it transmits, not through any form of feedback mechanism.

4 Screenshots

4.1 PWM signals

Note: The tilt pulse is displayed on the top channel, and the pan pulse is displayed on the bottom channel.

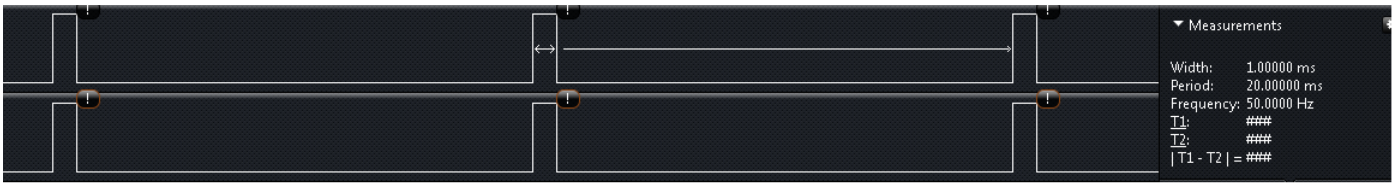

Figure 1: 0°tilt and 0°pan

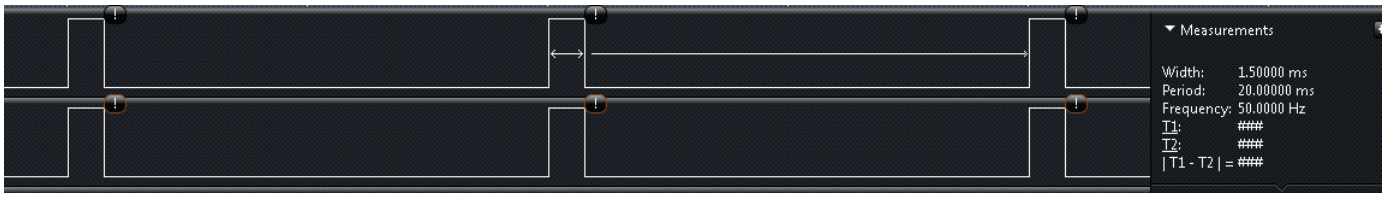

Figure 2: 60°tilt and 180°pan

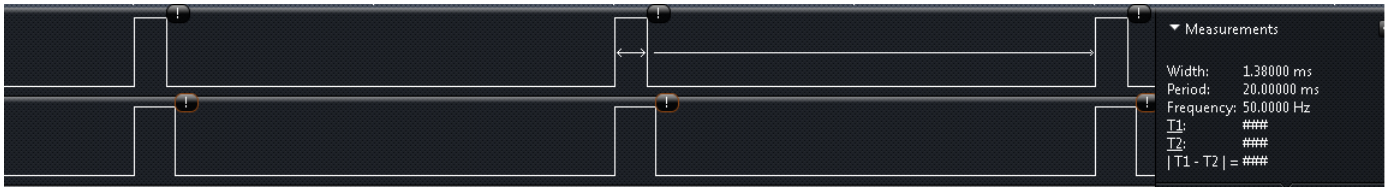

Figure 3: 45°tilt and 270°pan

4.2 XBee

Note: The pan and tilt angles are reversed in the XBee packets. The pan angle precedes the tilt angle.

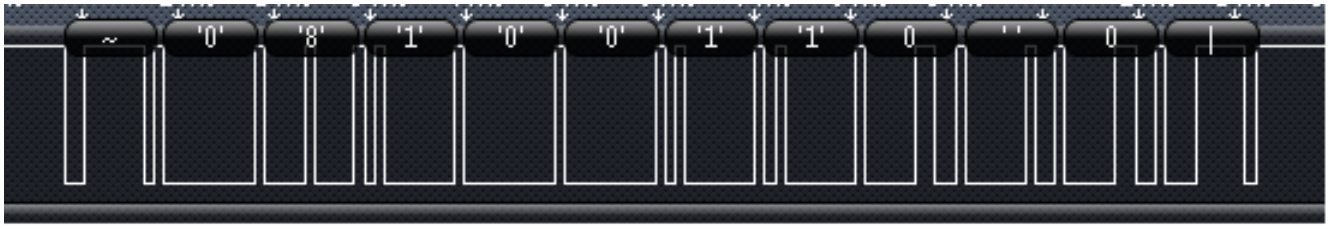

Figure 4: 0°tilt and 0°pan

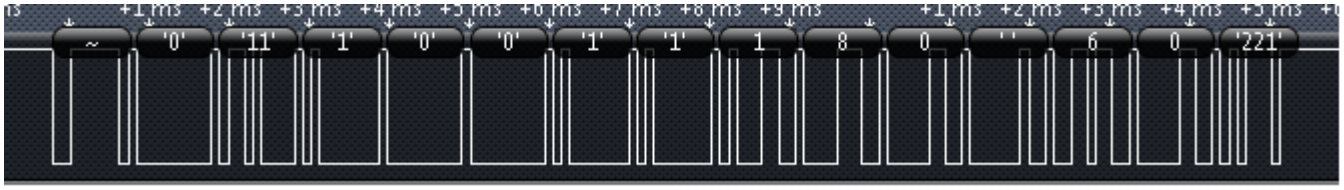

Figure 5: 60°tilt and 180°pan

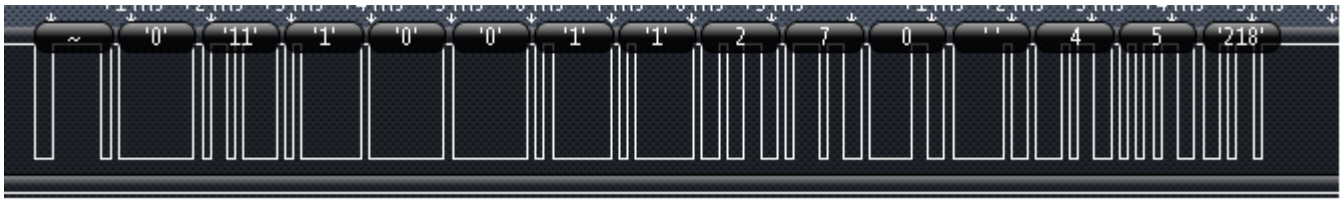

Figure 6: 45°tilt and 270°pan

5 Difficulties

5.1 Part I

The main difficulty for part I was to find a sufficient mathematical calculation that would give us a 1.5ms pulse width and a frequency of 50Hz. Not fully understanding all of the functions provided in the pwmgen code also led to frustration.

5.2 Part II

The main difficulty for part II was to find an Algorithm to convert angles into the correct PMW pulse width.

5.3 Part III

There was a major XBee problem introduced in this lab that wasn't present in Lab3. There was a delay between the LM3S2110 XBee and the actual LM3S2110. The delay caused us to think about what was really going on in the lower level and change our code accordingly. Allowing arrow buttons to change the angles of the servo was also a major problem. We had to increment/decrement the angles everytime an arrow button was pressed. To do this, we had to keep track of the previous angles that were sent to the LM3S2110 and manipulate those angles accordingly. To allow continuous movement of the servo as an arrow button is held, we would needed to change a fundamental part of our code that parsed IR signals. Doing so may drastically change the expectation of our code. We did not want to deal with this problem, so we did not implement this feature.

A Code for Lab4

A.1 LM3S2110

```
// Lab 4 LM3S2110
// Sean Ho & Richard Szeto
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/debug.h"
#include "driverlib/gpio.h"
#include "driverlib/pwm.h"
#include "driverlib/sysctl.h"
#include "drivers/rit128x96x4.h"


#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 pwmOffset1=0;              // tilt pwm offset
volatile int pwmOffset2=0;              // pan pwm offset
int flag = 0;                           // flag for button press delay
int code_offset = 0;                    // placeholder for position in sequence array
char string[CODE_SIZE];                 // array used to convert int to ascii
char display2[CODE_SIZE];               // used to store the payload from the Uart

//prototypes
char decode (char* input);              // interpret sequence data
void checkBounds ();                    // check the bounds of the servo
void move();                            // change pulse width to control servo

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;
}


void SysTickHandler (){
    waitTime += 1;
    waitTime2 += 1;


}
```

```
// adjust the angle of the servo
void adjustAngle (char *str)
{

  float percent, percent2;
  int i, num, angle1=0, angle2=0;
  for(i=0;str[i]!=' ';i++)
  {
    num=angle1*10 + (str[i] - '0');
    angle1=num;
  }
  i++;
  for(i;str[i]!='\0';i++)
  {
    num=angle2*10 + (str[i] - '0');
    angle2=num;
  }

  percent = angle1;
  percent = percent/360 * 100 * 98/100;
  percent -= 49;
  pwmOffset2 = percent;

  percent = angle2;
  percent2 = percent/120 * 100;
  percent = percent2;
  percent -= 50;
  pwmOffset1 = percent;
  if(angle2==121)
    GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_2,0x04);
  move ();
}
//UART interrupt handler
void UARTIntHandler0 ()
{
    unsigned long ulStatus;
    unsigned long c;

    int display_offset = 0;
    int size, i;
    ulStatus = UARTIntStatus(UART0_BASE, true);
    UARTIntClear(UART0_BASE, ulStatus);

    if(UARTCharsAvail(UART0_BASE))
    {
        UARTCharGet(UART0_BASE);//0
        UARTCharGet(UART0_BASE);//1
        size = UARTCharGet(UART0_BASE) - 5;//2
        UARTCharGet(UART0_BASE);//3
        UARTCharGet(UART0_BASE);//4
        UARTCharGet(UART0_BASE);//5
        UARTCharGet(UART0_BASE);//6
        UARTCharGet(UART0_BASE);//7
      i=0;
        for(display_offset=0; display_offset <size; display_offset++)
        {
          i++;
            c = UARTCharGet(UART0_BASE);
            display2[display_offset] = c;
        }
        UARTCharGet(UART0_BASE);
```

```
    }
    display2[size] = '\0';

    adjustAngle(display2);


    while(UARTCharsAvail(UART0_BASE))
        UARTCharGet(UART0_BASE);


    GPIOPinIntClear(GPIO_PORTB_BASE, GPIO_PIN_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;

}

// 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;

}

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();
        }
}

// checks bounds of the servo
void checkBounds ()
{
        if(pwmOffset2>49)
          pwmOffset2 = 49;
        else if(pwmOffset2<-49)
          pwmOffset2 = -49;

        if(pwmOffset1>50)
          pwmOffset1 = 50;
        else if(pwmOffset1<-50)
          pwmOffset1 = -50;
}

// change pulse width to move serve
void move ()
{
        checkBounds();
        PWMPulseWidthSet(PWM_BASE, PWM_OUT_0, SysCtlClockGet() * (0.0015 + 0.00001*pwmOffset1)
            /8);
        PWMPulseWidthSet(PWM_BASE, PWM_OUT_1, SysCtlClockGet() * (0.0015 + 0.00001*pwmOffset2)
            /8);
}

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

        unsigned long ulStatus;
        int i;
        // Reset delay counter
        waitTime=0;
        // skip repeating IR pulse sequences
        if(!checkProtocol())
                return;

        // parse IR pulse sequence data
        getData();


        if(decode(string)=='L')
          pwmOffset2 -= 1;
        else if(decode(string)=='R')
          pwmOffset2 += 1;
        else if (decode(string)=='U')
          pwmOffset1 += 10;
        else if (decode(string)=='D')
          pwmOffset1 -= 10;

        checkBounds ();
        move ();

        // 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;
}

int
main(void)
{
    unsigned long ulPeriod;

    //
    // Set the clocking to run directly from the crystal.
    //
    SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN |
                   SYSCTL_XTAL_8MHZ);
    SysCtlPWMClockSet(SYSCTL_PWMDIV_8);

     //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);

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


      // 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);



    //pwm
    SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_0);
    GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_1);
    PWMGenConfigure(PWM_BASE, PWM_GEN_0,
    PWM_GEN_MODE_DOWN | PWM_GEN_MODE_NO_SYNC);
    ulPeriod = SysCtlClockGet()/50/8;
    PWMGenPeriodSet(PWM0_BASE, PWM_GEN_0, ulPeriod);
    PWMPulseWidthSet(PWM_BASE, PWM_OUT_0, SysCtlClockGet() * 0.0015/8);
```

```
    PWMPulseWidthSet(PWM_BASE, PWM_OUT_1, SysCtlClockGet() * 0.0015/8);
    PWMGenEnable(PWM_BASE, PWM_GEN_0);
    PWMOutputState(PWM_BASE, (PWM_OUT_0_BIT | PWM_OUT_1_BIT), true);



    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';
        }
}
```

A.2 LM3S8962

```
// Lab 4 LM3S8962
// Sean Ho & Richard Szeto

#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 angle1=0;          // tilt angle
volatile int angle2=0;          // pan angle
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 for top half
int dLocx=0;                  // OLED display buffer offset
char buffer[100];             // used to store XBee packet to be sent
char display2[50];            // OLED display buffer for bottom half

//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));
        RIT128x96x4StringDraw(display, 0, 0, 15);

    // 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';
    }
}

void strToDegree (char *str)
{
    int i, num;
    angle1=0;
    angle2=0;
       for(i=0;str[i]!=' ';i++)
    {
       num=angle1*10 + (str[i] - '0');
       angle1=num;
    }
    i++;
    for(i=0;str[i]!='\0';i++)
    {
       num=angle2*10 + (str[i] - '0');
       angle2=num;
    }
```

```
    for(i=0; str[i]!='\0';i++)
    {
        display2[i]=str[i];
    }
 }

// convert button to character output
 void decodeLetter (char input) {
    if (input == '1')
    {
        display[dLocx++] = '1';
    }
    else if (input == 'U')
    {
        //change the tilt
        int temp = angle1 - (angle1/100) + '0';
        angle2 = angle2 + (120.0/100.0);

        display[0] = (temp/100) + '0';
        temp = temp - (temp/100)*100;
        display[1] = (temp/10) + '0';
        temp = temp - (temp/10)*10;
        display[2] = temp + '0';

        display[3] = '.';

        temp = angle2 - (angle2/100) + '0';

        display[4] = (temp/100) + '0';
        temp = temp - (temp/100)*100;
        display[5] = (temp/10) + '0';
        temp = temp - (temp/10)*10;
        display[6] = temp + '0';

        display[7] = '\0';

    }
    else if (input == 'D')
    {
        //change the tilt
    int temp = angle1 - (angle1/100) + '0';
        angle2 = angle2 - (120.0/100.0);

    display[0] = (temp/100) + '0';
        temp = temp - (temp/100)*100;
        display[1] = (temp/10) + '0';
        temp = temp - (temp/10)*10;
        display[2] = temp + '0';

        display[3] = '.';

        temp = angle2 - (angle2/100) + '0';

        display[4] = (temp/100) + '0';
        temp = temp - (temp/100)*100;
        display[5] = (temp/10) + '0';
        temp = temp - (temp/10)*10;
        display[6] = temp + '0';

        display[7] = '\0';
    }
    else if (input == 'L')
```

```
{
    //change the pan
    angle1 = angle1 - (360.0/98.0);
int temp = angle1 - (angle1/100) + '0';

display[0] = (temp/100) + '0';
    temp = temp - (temp/100)*100;
    display[1] = (temp/10) + '0';
    temp = temp - (temp/10)*10;
    display[2] = temp + '0';

    display[3] = '␣';

    temp = angle2 - (angle2/100) + '0';

    display[4] = (temp/100) + '0';
    temp = temp - (temp/100)*100;
    display[5] = (temp/10) + '0';
    temp = temp - (temp/10)*10;
    display[6] = temp + '0';

    display[7] = '\0';
}
else if (input == 'R')
{
    //change the pan
    angle1 = angle1 + (360.0/98.0);
int temp = angle1 - (angle1/100) + '0';

display[0] = (temp/100) + '0';
    temp = temp - (temp/100)*100;
    display[1] = (temp/10) + '0';
    temp = temp - (temp/10)*10;
    display[2] = temp + '0';

    display[3] = '␣';

    temp = angle2 - (angle2/100) + '0';

    display[4] = (temp/100) + '0';
    temp = temp - (temp/100)*100;
    display[5] = (temp/10) + '0';
    temp = temp - (temp/10)*10;
    display[6] = temp + '0';

    display[7] = '\0';
}
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);
                    strToDegree(display);
```

```
            RIT128x96x4Clear();
            RIT128x96x4StringDraw("----------------------", 0, 50, 15);
            RIT128x96x4StringDraw(display2, 0, 60, 15);

      for(i=0; i<dLocx + 1; i++)
                display[i]='\0';
            RIT128x96x4StringDraw(display, 0, 0, 15);
      dLocx=0;
      return;
    }
    else if(input == 'P')
    {
        showError();          // Error
    }
    if (input == '2')
        {
            display[dLocx++] = '2';
        }
        else if (input == '3')
        {
            display[dLocx++] = '3';
        }
        else if (input == '4')
        {
            display[dLocx++] = '4';
        }
        else if (input == '5')
        {
            display[dLocx++] = '5';
        }
        else if (input == '6')
        {
            display[dLocx++] = '6';
        }
        else if (input == '7')
        {
            display[dLocx++] = '7';
        }
        else if (input == '8')
        {
            display[dLocx++] = '8';
        }
        else if (input == '9')
        {
            display[dLocx++] = '9';
        }
        else if (input == '0')
        {
            if (flag == 1)
            {
                if(dLocx != 0)
                {
                    if (display[dLocx - 1] == '0')
                        display[dLocx - 1] = ' ';
                    else if (display[dLocx - 1] == ' ')
                        display[dLocx - 1] = '0';
                    else
                        display[dLocx++] = '0';
                 }
                 else
                    display[dLocx++] = '0';
             }
```

```
            else
                display[dLocx++] = '0';
        }
    display[dLocx] = '\0';
}
```