1 Lab2 - Polling, Button

```c
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "inc/hw_gpio.h"
int32_t ButtonState = 0;
uint32_t g_w_delay = 2000000;
uint32_t g_RGB_delay = 20000000;
uint8_t g_flash_LED = 0;

void read_Switches(void) {
    if (GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_4)) {
        g_flash_LED = 0;
    } else {
        g_flash_LED = 1;
    }
}

void RGB_FSM(void) {
    static enum {ST_RED, ST_GREEN, ST_BLUE, ST_OFF} next_state;
    if (g_flash_LED == 0) {
        switch (next_state) {
            case ST_RED:
                //Control_RGB_LEDs(1, 0, 0);
                GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|
    GPIO_PIN_3, GPIO_PIN_1);
                SysCtlDelay(g_RGB_delay);
                next_state = ST_BLUE;
                break;
            case ST_BLUE:
                //Control_RGB_LEDs(0, 1, 0);
                GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|
    GPIO_PIN_3, GPIO_PIN_2);
                SysCtlDelay(g_RGB_delay);
                next_state = ST_GREEN;
                break;
            case ST_GREEN:
                //Control_RGB_LEDs(0, 0, 1);
                GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|
    GPIO_PIN_3, GPIO_PIN_3);
                SysCtlDelay(g_RGB_delay);
                next_state = ST_RED;
                break;

            default:
                next_state = ST_RED;
                break;
        }
    }
}

void flash_FSM(void) {
    static enum {ST_WHITE, ST_BLACK} next_state;
```

```
53      if (g_flash_LED == 1) {
54          switch (next_state) {
55              case ST_WHITE:
56                  //Control_RGB_LEDs(1, 1, 1);
57                  GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|
    GPIO_PIN_3, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);
58                  SysCtlDelay(g_w_delay);
59                  next_state = ST_BLACK;
60                  break;
61              case ST_BLACK:
62                  //Control_RGB_LEDs(0, 0, 0);
63                  GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|
    GPIO_PIN_3, 0x00);
64                  SysCtlDelay(g_w_delay);
65                  next_state = ST_WHITE;
66                  break;
67              default:
68                  next_state = ST_WHITE;
69                  break;
70          }
71      }
72  }
73
74  void flash(void) {
75      read_Switches();
76      flash_FSM();
77      RGB_FSM();
78
79  }
80
81  int main(void)
82  {
83      SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|
    SYSCTL_OSC_MAIN);
84          SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
85          GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|
    GPIO_PIN_3);
86          //We use pin4 to control SW1
87          GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, GPIO_PIN_4);
88          //Since the button needs some sort of pull-up, we set pin4 as
    weak pull-up
89          GPIOPadConfigSet(GPIO_PORTF_BASE, GPIO_PIN_4, GPIO_STRENGTH_2MA
    , GPIO_PIN_TYPE_STD_WPU);
90
91          while (1) {
92              flash();
93          }
94  }
```

2 Lab3 - Timer Interrupts - one shot, Button, FSM

```c
#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include "inc/tm4c123gh6pm.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/timer.h"
#include "driverlib/interrupt.h"

#define W_DELAY 150
#define RGB_DELAY 1500

uint8_t g_flash_LED = 0;
bool timer1finish = 1;
bool timer0finish = 1;

uint32_t ui32Period;

void Timer1IntHandler(void);
void Timer0IntHandler(void);
void PORTF_IRQHandler(void);

void initialization(void)
{
    SysCtlClockSet(SYSCTL_SYSDIV_5 | SYSCTL_USE_PLL | SYSCTL_XTAL_16MHZ | SYSCTL_OSC_MAIN);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3);
    GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, GPIO_PIN_4 | GPIO_PIN_0);
    GPIOPadConfigSet(GPIO_PORTF_BASE, GPIO_PIN_4, GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);

    GPIOIntTypeSet(GPIO_PORTF_BASE, GPIO_PIN_4, GPIO_BOTH_EDGES);
    GPIOIntEnable(GPIO_PORTF_BASE, GPIO_PIN_4);
    GPIOIntRegister(GPIO_PORTF_BASE, PORTF_IRQHandler);

    // Configure Timer1
    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1);
    TimerConfigure(TIMER1_BASE, TIMER_CFG_ONE_SHOT);
    IntEnable(INT_TIMER1A);
    TimerIntRegister(TIMER1_BASE, TIMER_A, Timer1IntHandler);
    TimerIntEnable(TIMER1_BASE, TIMER_TIMA_TIMEOUT);

    ui32Period = SysCtlClockGet() / 1000;
    TimerLoadSet(TIMER1_BASE, TIMER_A, RGB_DELAY * ui32Period);

    // Configure Timer0
    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
    TimerConfigure(TIMER0_BASE, TIMER_CFG_ONE_SHOT);
    IntEnable(INT_TIMER0A);
    TimerIntRegister(TIMER0_BASE, TIMER_A, Timer0IntHandler);
```

```
53      TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
54
55      TimerLoadSet(TIMER0_BASE, TIMER_A, W_DELAY * ui32Period);
56
57      IntMasterEnable();
58  }
59
60  // TODO: Using GPIO Interrupt to replace the function
        Read_Switches_Timer()
61  void PORTF_IRQHandler(void)
62  {
63      // Check if the interrupt was triggered by PD4
64      if (GPIOIntStatus(GPIO_PORTF_BASE, true) & GPIO_INT_PIN_4)
65      {
66          // Toggle the LED or perform your action
67          if (GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_4))
68              g_flash_LED = 0; // Button released
69          else
70              g_flash_LED = 1; // Button pressed
71
72      }
73      // Clear the interrupt flag
74      GPIOIntClear(GPIO_PORTF_BASE, GPIO_INT_PIN_4);
75  }
76
77  void Timer1IntHandler(void)
78  {
79      TimerIntClear(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
80      timer1finish = 1;
81  }
82
83  void Timer0IntHandler(void)
84  {
85      TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
86      timer0finish = 1;
87  }
88
89  void Flash_Timer(void)
90  {
91      // TODO: Implement this function similar to RGB_Timer()
92      static enum {W, W_wait, OFF, OFF_wait} next_state;
93      if (g_flash_LED == 1) {
94          switch (next_state) {
95              case W:
96                  GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 |
        GPIO_PIN_3, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3);
97                  timer0finish = 0;
98                  TimerEnable(TIMER0_BASE, TIMER_A);
99                  next_state = W_wait;
100                 break;
101             case W_wait:
102                 if (timer0finish)
103                     next_state = OFF;
104                 break;
105             case OFF:
106                 GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 |
        GPIO_PIN_3, 0);
107                 timer0finish = 0;
```

```
108                 TimerEnable(TIMER0_BASE, TIMER_A);
109                 next_state = OFF_wait;
110                 break;
111             case OFF_wait:
112                 if (timer0finish)
113                     next_state = W;
114                 break;
115             default:
116                 next_state = W;
117                 break;
118         }
119     }
120 }
121
122 void RGB_Timer(void)
123 {
124     if (g_flash_LED == 0)
125     {
126         static enum { R,
127                       R_wait,
128                       G,
129                       G_wait,
130                       B,
131                       B_wait } next_state;
132         switch (next_state)
133         {
134         case R:
135             GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 |
    GPIO_PIN_3, GPIO_PIN_1);
136             timer1finish = 0;
137             TimerEnable(TIMER1_BASE, TIMER_A);
138             next_state = R_wait;
139             break;
140         case R_wait:
141             if (timer1finish)
142                 next_state = G;
143             break;
144         case G:
145             GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 |
    GPIO_PIN_3, GPIO_PIN_2);
146             timer1finish = 0;
147
148             TimerEnable(TIMER1_BASE, TIMER_A);
149             next_state = G_wait;
150             break;
151         case G_wait:
152             if (timer1finish)
153                 next_state = B;
154             break;
155         case B:
156             GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 |
    GPIO_PIN_3, GPIO_PIN_3);
157             timer1finish = 0;
158             TimerEnable(TIMER1_BASE, TIMER_A);
159             next_state = B_wait;
160             break;
161         case B_wait:
162             if (timer1finish)
```

```
163                 next_state = R;
164             break;
165         default:
166             next_state = R;
167             break;
168         }
169     }
170 }
171
172 int main(void)
173 {
174     initialization();
175     g_flash_LED = 0;
176     while (1)
177     {
178         RGB_Timer();
179         Flash_Timer();
180     }
181 }
```

3 Lab4 - LCD, Keypad

```
#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include "inc/tm4c123gh6pm.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/interrupt.h"
#include "driverlib/gpio.h"
#include "driverlib/timer.h"
#define RS_PIN GPIO_PIN_5   // select pin 5 for RS
#define RW_PIN GPIO_PIN_6   // select pin 6 for RS
#define EN_PIN GPIO_PIN_7   // select pin 7 for EN
#define DB_PIN GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3 |
    GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 | GPIO_PIN_7
            // select pins 0~7 for DB
#define ROW GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4
     // select pins 1~4 for ROW
#define COL GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6
     // select pins 4~6 for COL

void flushInput(uint32_t ui32Port, uint8_t ui8Pins);
void delayUs(int n);
void LCD_command(bool rs, bool rw, unsigned char data);
char *message_str1 = "Please Enter:";
char *message_str2 = "                    ";
int flag;
int n;
int cursor_pos = 0xC0;
int inputEnable = 1;
int main(void)
{
begin:
    SysCtlClockSet(SYSCTL_SYSDIV_5 | SYSCTL_USE_PLL | SYSCTL_XTAL_16MHZ
    | SYSCTL_OSC_MAIN);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
    GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, DB_PIN);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, RS_PIN | RW_PIN | EN_PIN);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
    GPIOPinTypeGPIOOutput(GPIO_PORTE_BASE, ROW);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
    GPIOPinTypeGPIOInput(GPIO_PORTC_BASE, COL);
    GPIOPadConfigSet(GPIO_PORTC_BASE, COL, GPIO_STRENGTH_4MA,
    GPIO_PIN_TYPE_STD_WPU);
    /* LCD configuration */
    LCD_command(0, 0, 0x38); // SET FUNCTION: specify 8-bit interface,
    2 line display, and 5x7 dots (font)
    LCD_command(0, 0, 0x08); // DISPLAY OFF: set display off
    LCD_command(0, 0, 0x01); // DISPLAY CLEAR: set display clear
    LCD_command(0, 0, 0x06); // ENTRY MODE SET: set cursor move
    direction as decreasing & display is not shifted
    LCD_command(0, 0, 0x0F); // DISPLAY ON/OFF: set display on, cursor
    on, & cursor blinking on
    LCD_command(0, 0, 0x80); // SET DD DRAM ADDRESS: set cursor to the
```

```
    first line
47   n = 0;
48   while (message_str1[n] != '\0')
49   {
50       LCD_command(1, 0, message_str1[n]); // WRITE DATA: display
    message_str1[n] on the LCD
51       n++;
52   }
53   LCD_command(0, 0, cursor_pos); // SET DD DRAM ADDRESS: set cursor
    to the first line
54
55   while (1)
56   {
57
58       GPIOPinWrite(GPIO_PORTE_BASE, ROW, 0x1C); // first row
59       if (!GPIOPinRead(GPIO_PORTC_BASE, GPIO_PIN_4) && inputEnable)
60       {
61           LCD_command(0, 0, cursor_pos++); // display behind "Please
    Enter:"
62           LCD_command(1, 0, '1');
63           flushInput(GPIO_PORTC_BASE, GPIO_PIN_4);
64       }
65       else if (!GPIOPinRead(GPIO_PORTC_BASE, GPIO_PIN_5) &&
    inputEnable)
66       {
67           LCD_command(0, 0, cursor_pos++);
68           LCD_command(1, 0, '2');
69           flushInput(GPIO_PORTC_BASE, GPIO_PIN_5);
70       }
71       else if (!GPIOPinRead(GPIO_PORTC_BASE, GPIO_PIN_6) &&
    inputEnable)
72       {
73           LCD_command(0, 0, cursor_pos++);
74           LCD_command(1, 0, '3');
75           flushInput(GPIO_PORTC_BASE, GPIO_PIN_6);
76       }
77
78       GPIOPinWrite(GPIO_PORTE_BASE, ROW, 0x1A); // second row
79       if (!GPIOPinRead(GPIO_PORTC_BASE, GPIO_PIN_4) && inputEnable)
80       {
81           LCD_command(0, 0, cursor_pos++);
82           LCD_command(1, 0, '4');
83           flushInput(GPIO_PORTC_BASE, GPIO_PIN_4);
84       }
85       else if (!GPIOPinRead(GPIO_PORTC_BASE, GPIO_PIN_5) &&
    inputEnable)
86       {
87           LCD_command(0, 0, cursor_pos++);
88           LCD_command(1, 0, '5');
89           flushInput(GPIO_PORTC_BASE, GPIO_PIN_5);
90       }
91       else if (!GPIOPinRead(GPIO_PORTC_BASE, GPIO_PIN_6) &&
    inputEnable)
92       {
93           LCD_command(0, 0, cursor_pos++);
94           LCD_command(1, 0, '6');
95           flushInput(GPIO_PORTC_BASE, GPIO_PIN_6);
96       }
```

```
97
98          GPIOPinWrite(GPIO_PORTE_BASE, ROW, 0x16); // third row
99          if (!GPIOPinRead(GPIO_PORTC_BASE, GPIO_PIN_4) && inputEnable)
100         {
101             LCD_command(0, 0, cursor_pos++);
102             LCD_command(1, 0, '7');
103             flushInput(GPIO_PORTC_BASE, GPIO_PIN_4);
104         }
105         else if (!GPIOPinRead(GPIO_PORTC_BASE, GPIO_PIN_5) &&
    inputEnable)
106         {
107             LCD_command(0, 0, cursor_pos++);
108             LCD_command(1, 0, '8');
109             flushInput(GPIO_PORTC_BASE, GPIO_PIN_5);
110         }
111         else if (!GPIOPinRead(GPIO_PORTC_BASE, GPIO_PIN_6) &&
    inputEnable)
112         {
113             LCD_command(0, 0, cursor_pos++);
114             LCD_command(1, 0, '9');
115             flushInput(GPIO_PORTC_BASE, GPIO_PIN_6);
116         }
117
118         GPIOPinWrite(GPIO_PORTE_BASE, ROW, 0x0E); // forth row
119         if (!GPIOPinRead(GPIO_PORTC_BASE, GPIO_PIN_4)) // *
120         {
121             LCD_command(0, 0, 0x01); // DISPLAY CLEAR: set display
    clear
122             cursor_pos = 0xC0;
123             inputEnable = 1;
124             flushInput(GPIO_PORTC_BASE, GPIO_PIN_5);
125             goto begin;
126         }
127         else if (!GPIOPinRead(GPIO_PORTC_BASE, GPIO_PIN_5) &&
    inputEnable)
128         {
129             LCD_command(0, 0, cursor_pos++);
130             LCD_command(1, 0, '0');
131             flushInput(GPIO_PORTC_BASE, GPIO_PIN_5);
132         }
133         else if (!GPIOPinRead(GPIO_PORTC_BASE, GPIO_PIN_6))
134         {
135             LCD_command(0, 0, 0b01100); // DISPLAY ON/OFF: set display
    on, cursor on, & cursor blinking on
136             inputEnable = 0;
137             flushInput(GPIO_PORTC_BASE, GPIO_PIN_6);
138         }
139
140         if (cursor_pos > 0xCF) {
141             cursor_pos = 0xC0;
142         }
143     }
144 }
145
146 void flushInput(uint32_t ui32Port, uint8_t ui8Pins)
147 {
148     /* wait until the key is release to avoid redundant inputs. */
149     while (!GPIOPinRead(ui32Port, ui8Pins))
```

```
150     {
151         delayUs(100000);
152     }
153 }
154 
155 void delayUs(int n)
156 {
157     SysCtlDelay((n * 40) / 3);
158 }
159 void LCD_command(bool rs, bool rw, unsigned char data)
160 {
161     if (rs == 0)                                        // L: Command H:
    Data
162         GPIOPinWrite(GPIO_PORTA_BASE, RS_PIN, 0x00); // set RS as L
163     else
164         GPIOPinWrite(GPIO_PORTA_BASE, RS_PIN, 0x20); // set RS as H
165     if (rw == 0)                                        // L: Write mode;
    H: Read mode
166         GPIOPinWrite(GPIO_PORTA_BASE, RW_PIN, 0x00); // set RW as L
167     else
168         GPIOPinWrite(GPIO_PORTA_BASE, RW_PIN, 0x40); // set RW as H
169     delayUs(1);
170     GPIOPinWrite(GPIO_PORTA_BASE, EN_PIN, 0x80); // set H to enable
    signal EN
171     GPIOPinWrite(GPIO_PORTB_BASE, DB_PIN, data); // assign DB0~DB7 with
     "data"
172     delayUs(1);
173     GPIOPinWrite(GPIO_PORTA_BASE, EN_PIN, 0x00); // set H->L to enable
    signal EN
174     delayUs(1);
175     if (rs == 0) // L: Command
176     {
177         if ((data == 0x01) | (data == 0x02) | (data == 0x03))
178             delayUs(1640); // Clear Display & Display/Cursor Home take
    1.64ms
179         else
180             delayUs(40); // all the others commands require only 40us
    to execute
181     }
182     else
183         delayUs(40); // Data Write takes 40us to execute
184 }
```

4 Lab5 - UART, Bluetooth, Interrupts

main_at.c

```
/*
### UART communication with bluetooth HC-05 and PC ###
UART0 used to communicate with PC
UART5 used to communicate with BLUETOOTH HC-05
UART5IntHandler to handle interrupt

Hardware connection:
RXD -> PE5
TXD -> PE4
GND -> GND
VCC -> VBUS

Steps:
1. Enable AT mode and configure the bluetooth module
    1.1 Connect bluetooth-EN to Tiva-VCC to enable AT mode. AT mode is correctly enabled if the LED blinks slowly (around 2-second period)
    1.2 Use the following AT commands: AT+UART? / AT+ROLE? / AT+ADDR? to check the current configuration
    1.3 Set the baud rate (AT+UART) of both the master and the slave 38400
    1.4 Set the role (AT+ROLE) of the master 1, the slave 0
    1.5 Enable the fixed-address mode (AT+CMODE) of the master
    1.6 Bind (AT+BIND) the destination of the master to the slave address (AT+ADDR?)
2. Disconnect bluetooth-EN and Tiva-VCC to disable AT mode
3. Complete, compile and run the code
*/

#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "utils/uartstdio.h"
#include "inc/hw_ints.h"
#include "driverlib/interrupt.h"

void UART0IntHandler(void);
void UART5IntHandler(void);

int main(void) {
    // set clock
    SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN | SYSCTL_XTAL_16MHZ);

    // enable UART0 and GPIOA to communicate with PC
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    // configure PA0 for RX, PA1 for TX
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
```

```
50      // set PA0 and PA1 as type UART
51      GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
52      // set UART0 base address, clock and baud rate
53      UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200,
54          (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
    UART_CONFIG_PAR_NONE));
55
56      // enable UART5 and GPIOE to communicate with BLUETOOTH
57      SysCtlPeripheralEnable(SYSCTL_PERIPH_UART5);
58      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
59      // configure PE4 for RX, PE5 for TX
60      GPIOPinConfigure(GPIO_PE4_U5RX);
61      GPIOPinConfigure(GPIO_PE5_U5TX);
62      // set PE4 and PE5 as type UART
63      GPIOPinTypeUART(GPIO_PORTE_BASE, GPIO_PIN_4 | GPIO_PIN_5);
64      // set UART5 base address, clock and baud rate
65      UARTConfigSetExpClk(UART5_BASE, SysCtlClockGet(), 38400,
66          (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
    UART_CONFIG_PAR_NONE));
67
68      // configure interrupts
69      IntMasterEnable();
70      IntEnable(INT_UART0);
71      UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT);
72      UARTIntRegister(UART0_BASE, UART0IntHandler);
73      IntEnable(INT_UART5);
74      UARTIntEnable(UART5_BASE, UART_INT_RX | UART_INT_RT);
75      UARTIntRegister(UART5_BASE, UART5IntHandler);
76
77      // UART0 connection indicator
78      // UART0 connected if the serial monitor displays `UART0 connected
    !`
79      UARTCharPut(UART0_BASE, 'U');
80      UARTCharPut(UART0_BASE, 'A');
81      UARTCharPut(UART0_BASE, 'R');
82      UARTCharPut(UART0_BASE, 'T');
83      UARTCharPut(UART0_BASE, '0');
84      UARTCharPut(UART0_BASE, ' ');
85      UARTCharPut(UART0_BASE, 'C');
86      UARTCharPut(UART0_BASE, 'o');
87      UARTCharPut(UART0_BASE, 'n');
88      UARTCharPut(UART0_BASE, 'n');
89      UARTCharPut(UART0_BASE, 'e');
90      UARTCharPut(UART0_BASE, 'c');
91      UARTCharPut(UART0_BASE, 't');
92      UARTCharPut(UART0_BASE, 'e');
93      UARTCharPut(UART0_BASE, 'd');
94      UARTCharPut(UART0_BASE, '!');
95      UARTCharPut(UART0_BASE, '\n');
96
97      while (true) {}
98  }
99
100 // handler when Tiva receives data from UART0
101 void UART0IntHandler(void)
102 {
103     // get interrupt status
104     uint32_t ui32Status = UARTIntStatus(UART0_BASE, true);
```

```
105     // clear the interrupt signal
106     UARTIntClear(UART0_BASE, ui32Status);
107     // receive data from UART0
108     while (UARTCharsAvail(UART0_BASE))
109     {
110         // forward the characters from UART0 to UART5 and back to UART0
111         char a = UARTCharGet(UART0_BASE);
112         UARTCharPut(UART5_BASE, a);
113         UARTCharPut(UART0_BASE, a);
114     }
115 }
116
117 // handler when Tiva receives data from UART5
118 void UART5IntHandler(void)
119 {
120     // get interrupt status
121     uint32_t ui32Status = UARTIntStatus(UART5_BASE, true);
122     // clear the interrupt signal
123     UARTIntClear(UART5_BASE, ui32Status);
124     // receive data from UART5
125     while (UARTCharsAvail(UART5_BASE))
126     {
127         // forward the characters from UART5 to UART0
128         char b = UARTCharGet(UART5_BASE);
129         UARTCharPut(UART0_BASE, b);
130     }
131 }
```

main_master.c

```
1  /*
2  ### UART communication with bluetooth HC-05 and PC ###
3  UART0 used to communicate with PC
4  UART5 used to communicate with BLUETOOTH HC-05
5
6  Hardware connection:
7  RXD -> PE5
8  TXD -> PE4
9  GND -> GND
10 VCC -> VBUS
11
12 Steps:
13 1. Enable AT mode and configure the bluetooth module
14     1.1 Connect bluetooth-EN to Tiva-VCC to enable AT mode. AT mode is
       correctly enabled if the LED blinks slowly (around 2-second period)
15     1.2 Use the following AT commands: AT+UART? / AT+ROLE? / AT+ADDR?
       to check the current configuration
16     1.3 Set the baud rate (AT+UART) of both the master and the slave
       38400
17     1.4 Set the role (AT+ROLE) of the master 1, the slave 0
18     1.5 Enable the fixed-address mode (AT+CMODE) of the master
19     1.6 Bind (AT+BIND) the destination of the master to the slave
       address (AT+ADDR?)
20 2. Disconnect bluetooth-EN and Tiva-VCC to disable AT mode
21 3. Complete, compile and run the code
22 */
23
24 #include <stdint.h>
```

```c
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "utils/uartstdio.h"
#include "inc/hw_ints.h"
#include "driverlib/interrupt.h"


void PORTF_IRQHandler(void);

int main(void) {
    // set clock
    SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN | SYSCTL_XTAL_16MHZ);

    // TODO: YOUR UART INITIALIZATION PROCEDURE
    // enable UART5 and GPIOE to communicate with BLUETOOTH
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART5);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
    // configure PE4 for RX, PE5 for TX
    GPIOPinConfigure(GPIO_PE4_U5RX);
    GPIOPinConfigure(GPIO_PE5_U5TX);
    // set PORTE pin4 and pin5 as type UART
    GPIOPinTypeUART(GPIO_PORTE_BASE, GPIO_PIN_4 | GPIO_PIN_5);
    // set UART5 base address, clock and baud rate
    UARTConfigSetExpClk(UART5_BASE, SysCtlClockGet(), 38400,
        (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));

    // enable SW1 of GPIOF for button control
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, GPIO_PIN_4);
    GPIOPadConfigSet(GPIO_PORTF_BASE, GPIO_PIN_4, GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);

    // TODO: YOUR BUTTON CONTROL PROCEDURE
    // you can do
    // polling (i.e., checking button status and send data in a while loop), or
    // interrupt (i.e., writing an interrupt function to check button status and send data)
    GPIOIntTypeSet(GPIO_PORTF_BASE, GPIO_PIN_4, GPIO_BOTH_EDGES);
    GPIOIntEnable(GPIO_PORTF_BASE, GPIO_PIN_4);
    GPIOIntRegister(GPIO_PORTF_BASE, PORTF_IRQHandler);
    IntMasterEnable();
}

void PORTF_IRQHandler(void)
{
    // Check if the interrupt was triggered by PD4
    if (GPIOIntStatus(GPIO_PORTF_BASE, true) & GPIO_INT_PIN_4)
    {
        char a;
        // Toggle the LED or perform your action
```

```
78          if (GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_4))
79              a = 'Y';
80          else
81              a = 'X'; // Button pressed
82
83          UARTCharPut(UART5_BASE, a);
84      }
85      // Clear the interrupt flag
86      GPIOIntClear(GPIO_PORTF_BASE, GPIO_INT_PIN_4);
87  }
```

main_slave.c

```
1  /*
2  ### UART communication with bluetooth HC-05 and PC ###
3  UART0 used to communicate with PC
4  UART5 used to communicate with BLUETOOTH HC-05
5  UART5IntHandler to handle interrupt
6
7  Hardware connection:
8  RXD -> PE5
9  TXD -> PE4
10 GND -> GND
11 VCC -> VBUS
12
13 Steps:
14 1. Enable AT mode and configure the bluetooth module
15     1.1 Connect bluetooth-EN to Tiva-VCC to enable AT mode. AT mode is
       correctly enabled if the LED blinks slowly (around 2-second period)
16     1.2 Use the following AT commands: AT+UART? / AT+ROLE? / AT+ADDR?
       to check the current configuration
17     1.3 Set the baud rate (AT+UART) of both the master and the slave
       38400
18     1.4 Set the role (AT+ROLE) of the master 1, the slave 0
19     1.5 Enable the fixed-address mode (AT+CMODE) of the master
20     1.6 Bind (AT+BIND) the destination of the master to the slave
       address (AT+ADDR?)
21 2. Disconnect bluetooth-EN and Tiva-VCC to disable AT mode
22 3. Complete, compile and run the code
23 */
24
25 #include <stdint.h>
26 #include <stdbool.h>
27 #include "inc/hw_memmap.h"
28 #include "inc/hw_types.h"
29 #include "driverlib/gpio.h"
30 #include "driverlib/pin_map.h"
31 #include "driverlib/sysctl.h"
32 #include "driverlib/uart.h"
33 #include "utils/uartstdio.h"
34 #include "inc/hw_ints.h"
35 #include "driverlib/interrupt.h"
36
37 bool led_on = false;
38
39 void UART5IntHandler(void);
40
41 int main(void) {
```

```
42      // set clock
43      SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
        SYSCTL_XTAL_16MHZ);
44
45      // TODO: YOUR UART INITIALIZATION PROCEDURE
46
47
48      // enable UART5 and GPIOE to communicate with BLUETOOTH
49      SysCtlPeripheralEnable(SYSCTL_PERIPH_UART5);
50      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
51      // configure PE4 for RX, PE5 for TX
52      GPIOPinConfigure(GPIO_PE4_U5RX);
53      GPIOPinConfigure(GPIO_PE5_U5TX);
54      // set PORTE pin4 and pin5 as type UART
55      GPIOPinTypeUART(GPIO_PORTE_BASE, GPIO_PIN_4 | GPIO_PIN_5);
56      // set UART5 base address, clock and baud rate
57      UARTConfigSetExpClk(UART5_BASE, SysCtlClockGet(), 38400,
58      (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE))
        ;
59
60      // enable LED of GPIOF for display
61      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
62      GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 |
        GPIO_PIN_3);
63
64      // TODO: YOUR UART INTERRUPT INITIALIZATION PROCEDURE
65      UARTIntRegister(UART5_BASE, UART5IntHandler); // Register handler
66      UARTIntEnable(UART5_BASE, UART_INT_RX | UART_INT_RT); // Enable
        interrupt
67      IntEnable(INT_UART5); // Enable the processor to handle UART5
        interrupts
68      IntMasterEnable(); // Enable global interrupts
69      // set interrupt
70
71      while (true)
72      {
73          if (led_on) {
74              GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 |
        GPIO_PIN_3, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3);
75          }
76          else {
77              GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 |
        GPIO_PIN_3, 0);
78          }
79      }
80  }
81
82  // TODO: YOUR UART INTERRUPT HANDLER
83
84  // handler when Tiva receives data from UART5
85  void UART5IntHandler(void)
86  {
87      // get interrupt status
88          uint32_t ui32Status = UARTIntStatus(UART5_BASE, true);
89          // clear the interrupt signal
90          UARTIntClear(UART5_BASE, ui32Status);
91          // receive data from UART5
92          char b;
```

```
93          while (UARTCharsAvail(UART5_BASE))
94          {
95              // forward the characters from UART5 to UART0
96              b = UARTCharGet(UART5_BASE);
97  //            UARTCharPut(UART0_BASE, b);
98          }
99      // set `led_on` according to the received data
100         if (b == 'X') {
101             led_on = true;
102         } else if (b == 'Y') {
103            led_on = false;
104        }
105
106 }
```

5 Lab6 - ADC, Interrupts, Timer Periodic

```c
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_ints.h"
#include "inc/tm4c123gh6pm.h"
#include "inc/hw_gpio.h"
#include "driverlib/debug.h"
#include "driverlib/sysctl.h"
#include "driverlib/adc.h"
#include "driverlib/interrupt.h"
#include "driverlib/gpio.h"
#include "driverlib/timer.h"

uint32_t ui32ADC0Value[4];
volatile uint32_t ui32VolAvg;
volatile uint32_t ui32TempValueC;
volatile uint32_t ui32TempValueF;

void ADC0IntHandler(void) {
    //
    // Clear the ADC interrupt flag
    //
    ADCIntClear(ADC0_BASE, 1);

    //
    // Get the ADC data
    //
    ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0Value);

    //
    // Calculate average voltage value
    //
    ui32VolAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value
    [2] + ui32ADC0Value[3] + 2) / 4;

    //
    // TODO: Convert the voltage to Celsius and Fahrenheit
    //
    ui32TempValueC = 147.5 - ((75 * ui32VolAvg * 3.3) / 4096);
    ui32TempValueF = (ui32TempValueC * 1.8) + 32;
}

int main(void) {
    //
    // Set up the system clock
    //
    SysCtlClockSet(SYSCTL_SYSDIV_5 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
    SYSCTL_XTAL_16MHZ);

    //
    // The ADC0 peripheral must be enabled for use.
    //
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);

```

```
54      //
55      // Enable sample sequence 2 with a timer trigger. Sequence 2
56      // will do 4 sampleS when the processor sends a singal to start the
57      // conversion. Each ADC module has 4 programmable sequences,
        sequence 0
58      // to sequence 3. This example is arbitrarily using sequence 2.
59      //
60      ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_TIMER, 0);
61
62      //
63      // Configure step 0-3 on sequence 2. Sample the temperature sensor
64      // (ADC_CTL_TS) and configure the interrupt flag (ADC_CTL_IE) to be
        set
65      // when the sample is done. Tell the ADC logic that this is the
        last
66      // conversion on sequence 2 (ADC_CTL_END).  Sequence 2 and Sequence
        1 have 4
67      // programmable stepS. Sequence 3 has 1 programmable step, and
        sequence 0 has
68      // 8 programmable steps. For more information on the
69      // ADC sequences and steps, reference the datasheet.
70      //
71      ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_TS);
72      ADCSequenceStepConfigure(ADC0_BASE, 1, 1, ADC_CTL_TS);
73      ADCSequenceStepConfigure(ADC0_BASE, 1, 2, ADC_CTL_TS);
74      ADCSequenceStepConfigure(ADC0_BASE, 1, 3, ADC_CTL_TS | ADC_CTL_IE |
        ADC_CTL_END);
75
76      //
77      // Since sample sequence 2 is now configured, it must be enabled.
78      //
79      ADCSequenceEnable(ADC0_BASE, 1);
80
81      //
82      // TODO: Configure Timer to trigger the ADC
83      //
84      SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
85      TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);
86      TimerLoadSet(TIMER0_BASE, TIMER_A, SysCtlClockGet() / 10);
87
88      //
89      // Register the ADC interrupt handler and enable ADC interrupt
90      //
91      ADCIntRegister(ADC0_BASE, 1, ADC0IntHandler);
92      IntEnable(INT_ADC0SS1);
93      ADCIntEnable(ADC0_BASE, 1);
94
95      TimerControlTrigger(TIMER0_BASE,TIMER_A,true);
96      TimerEnable(TIMER0_BASE, TIMER_A);
97      while(1) {
98      }
99  }
```

6 Project - PWM, Motion sensing

mpu.c

```
#include <stdbool.h>
#include <stdint.h>
#include <stdbool.h>
#include "sensorlib/i2cm_drv.h"
#include "sensorlib/hw_mpu6050.h"
#include "sensorlib/mpu6050.h"
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_sysctl.h"
#include "inc/hw_types.h"
#include "inc/hw_i2c.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/interrupt.h"
#include "driverlib/i2c.h"
#include "driverlib/sysctl.h"

volatile bool g_bMPU6050Done;
tMPU6050 sMPU6050;
tI2CMInstance g_sI2CMSimpleInst;

//
// The function that is provided by this example as a callback when
    MPU6050
// transactions have completed.
//
void MPU6050Callback(void *pvCallbackData, uint_fast8_t ui8Status)
{
    // See if an error occurred.
    if (ui8Status != I2CM_STATUS_SUCCESS)
    {
    }

    // Indicate that the MPU6050 transaction has completed.
    g_bMPU6050Done = true;
}

//
// The interrupt handler for the I2C module.
//
void I2CMSimpleIntHandler(void)
{
    //
    // Call the I2C master driver interrupt handler.
    //
    I2CMIntHandler(&g_sI2CMSimpleInst);
}

void Initialization(void)
{
    //enable I2C module 0
    SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C0);
```

```
54
55      //reset module
56      SysCtlPeripheralReset(SYSCTL_PERIPH_I2C0);
57
58      //enable GPIO peripheral that contains I2C 0
59      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
60
61      // Configure the pin muxing for I2C0 functions on port B2 and B3.
62      GPIOPinConfigure(GPIO_PB2_I2C0SCL);
63      GPIOPinConfigure(GPIO_PB3_I2C0SDA);
64
65      // Select the I2C function for these pins.
66      GPIOPinTypeI2CSCL(GPIO_PORTB_BASE, GPIO_PIN_2);
67      GPIOPinTypeI2C(GPIO_PORTB_BASE, GPIO_PIN_3);
68
69      // Enable and initialize the I2C0 master module.
70      // Use the system clock for the I2C0 module.
71      I2CMasterInitExpClk(I2C0_BASE, SysCtlClockGet(), true);
72
73      //clear I2C FIFOs
74      HWREG(I2C0_BASE + I2C_O_FIFOCTL) = 80008000;
75
76      // Initialize the I2C master driver.
77      I2CMInit(&g_sI2CMSimpleInst, I2C0_BASE, INT_I2C0, 0xff, 0xff,
    SysCtlClockGet());
78      // Register the interrupt handler for I2C interrupts
79      I2CIntRegister(I2C0_BASE, I2CMSimpleIntHandler);
80
81      // Configure the MPU6050
82      g_bMPU6050Done = false;
83      MPU6050Init(&sMPU6050, &g_sI2CMSimpleInst, 0x68, MPU6050Callback, &
    sMPU6050);
84      while (!g_bMPU6050Done)
85      {
86      }
87
88  }
89
90  int main()
91  {
92      // Set the system clock to use the PLL with a 16 MHz crystal
    oscillator.
93      // The clock is divided by 1 (SYSCTL_SYSDIV_1) and uses an internal
     oscillator (SYSCTL_OSC_INT).
94      SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_PLL | SYSCTL_OSC_INT |
    SYSCTL_XTAL_16MHZ);
95
96      // Initialize the system (e.g., peripherals, hardware components)
97      Initialization();
98
99      // Declare arrays to store accelerometer and gyroscope data
100     float fAccel[3], fGyro[3];
101
102     // Reset the MPU6050 sensor by writing to the power management
    register (PWR_MGMT_1)
103     g_bMPU6050Done = false;
104     MPU6050ReadModifyWrite(&sMPU6050, MPU6050_O_PWR_MGMT_1, 0x00, 0
    b00000010 & MPU6050_PWR_MGMT_1_DEVICE_RESET, MPU6050Callback, &
```

```
    sMPU6050);
105     while (!g_bMPU6050Done)
106     {
107     }
108 
109     // Configure the MPU6050 to not be low power mode by writing to the
        power management register (PWR_MGMT_2)
110     g_bMPU6050Done = false;
111     MPU6050ReadModifyWrite(&sMPU6050, MPU6050_O_PWR_MGMT_2, 0x00, 0x00,
        MPU6050Callback, &sMPU6050);
112     while (!g_bMPU6050Done)
113     {
114     }
115 
116     // Main infinite loop to repeatedly read data from the MPU6050
117     while (1)
118     {
119         //
120         // Request another reading from the MPU6050 sensor
121         //
122         g_bMPU6050Done = false;
123         MPU6050DataRead(&sMPU6050, MPU6050Callback, &sMPU6050);
124         while (!g_bMPU6050Done)
125         {
126         }
127 
128         // Extract the accelerometer data (in floating-point format)
129         MPU6050DataAccelGetFloat(&sMPU6050, &fAccel[0], &fAccel[1], &
    fAccel[2]);
130 
131         // Extract the gyroscope data (in floating-point format)
132         MPU6050DataGyroGetFloat(&sMPU6050, &fGyro[0], &fGyro[1], &fGyro
    [2]);
133     }
134 }
```

servo.h

```
1  // hardware connection:
2  // servo red wire -> V Bus
3  // servo brown wire -> GND
4  // servo (pitch) orange wire -> PD0
5  // servo (yaw) orange wire -> PD1
6  // pitch: up-down, yaw: left-right
7 
8  #include <stdint.h>
9  #include <stdbool.h>
10 #include "inc/hw_memmap.h"
11 #include "inc/hw_types.h"
12 #include "inc/hw_gpio.h"
13 #include "driverlib/sysctl.h"
14 #include "driverlib/gpio.h"
15 #include "driverlib/debug.h"
16 #include "driverlib/pwm.h"
17 #include "driverlib/pin_map.h"
18 #include "driverlib/rom.h"
19 
20 float servo_pwm_freq = 50;
```

```
21
22 // determine the duty cycle according to the desired angle
23 float angleToPWMDutyCycle(float angle)
24 {
25     // angle (duty cycle): 0 (0.5ms/20ms), 90 (1.5ms/20ms), 180 (2.5ms
    /20ms)
26     // angle to pulse width: pulse_width = angle / 90 + 0.5
27     // pulse width to duty cycle: duty_cycle = pulse_width / period
28     // valid angle range: 0-180
29     return (angle / 90 + 0.5) / (1000 / servo_pwm_freq);
30 }
31
32 int main()
33 {
34     // set the system clock and the PWM clock
35     // system clock frequency : PWM clock frequency = 64 : 1
36     SysCtlClockSet(SYSCTL_SYSDIV_5 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
     SYSCTL_XTAL_16MHZ);
37     SysCtlPWMClockSet(SYSCTL_PWMDIV_64);
38     // enable module PWM1
39     SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1);
40     SysCtlDelay(SysCtlClockGet() / 30); // avoid program overheat &
    logic issues
41     // configure generator 0 of PWM1
42     PWMGenEnable(PWM1_BASE, PWM_GEN_0);
43     PWMGenConfigure(PWM1_BASE, PWM_GEN_0, PWM_GEN_MODE_DOWN);
44     // calculate the number of PWM instruction cycles in each PWM
    period
45     uint32_t pwm_period = (SysCtlClockGet() / 64 / servo_pwm_freq);
46     PWMGenPeriodSet(PWM1_BASE, PWM_GEN_0, pwm_period);
47     // enable the 0th and 1st outputs of PWM1
48     PWMOutputState(PWM1_BASE, PWM_OUT_0_BIT, true);
49     PWMOutputState(PWM1_BASE, PWM_OUT_1_BIT, true);
50     // PD0 and PD1 to send the signals to the servos
51     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
52     GPIOPinTypePWM(GPIO_PORTD_BASE, GPIO_PIN_0);
53     GPIOPinTypePWM(GPIO_PORTD_BASE, GPIO_PIN_1);
54     GPIOPinConfigure(GPIO_PD0_M1PWM0);
55     GPIOPinConfigure(GPIO_PD1_M1PWM1);
56
57     float pitch_angle, yaw_angle, pitch_duty_cycle, yaw_duty_cycle;
58
59     while (true)
60     {
61         yaw_angle = 0;
62         yaw_duty_cycle = angleToPWMDutyCycle(yaw_angle);
63         PWMPulseWidthSet(PWM1_BASE, PWM_OUT_1, PWMGenPeriodGet(
    PWM1_BASE, PWM_GEN_0) * yaw_duty_cycle);
64         SysCtlDelay(SysCtlClockGet() / 3);
65         yaw_angle = 90;
66         yaw_duty_cycle = angleToPWMDutyCycle(yaw_angle);
67         PWMPulseWidthSet(PWM1_BASE, PWM_OUT_1, PWMGenPeriodGet(
    PWM1_BASE, PWM_GEN_0) * yaw_duty_cycle);
68         SysCtlDelay(SysCtlClockGet() / 3);
69         yaw_angle = 180;
70         yaw_duty_cycle = angleToPWMDutyCycle(yaw_angle);
71         PWMPulseWidthSet(PWM1_BASE, PWM_OUT_1, PWMGenPeriodGet(
    PWM1_BASE, PWM_GEN_0) * yaw_duty_cycle);
```

```
72          SysCtlDelay(SysCtlClockGet() / 3);
73          yaw_angle = 90;
74          yaw_duty_cycle = angleToPWMDutyCycle(yaw_angle);
75          PWMPulseWidthSet(PWM1_BASE, PWM_OUT_1, PWMGenPeriodGet(
    PWM1_BASE, PWM_GEN_0) * yaw_duty_cycle);
76          SysCtlDelay(SysCtlClockGet() / 3);
77          pitch_angle = 60;
78          pitch_duty_cycle = angleToPWMDutyCycle(pitch_angle);
79          PWMPulseWidthSet(PWM1_BASE, PWM_OUT_0, PWMGenPeriodGet(
    PWM1_BASE, PWM_GEN_0) * pitch_duty_cycle);
80          SysCtlDelay(SysCtlClockGet() / 3);
81          pitch_angle = 90;
82          pitch_duty_cycle = angleToPWMDutyCycle(pitch_angle);
83          PWMPulseWidthSet(PWM1_BASE, PWM_OUT_0, PWMGenPeriodGet(
    PWM1_BASE, PWM_GEN_0) * pitch_duty_cycle);
84          SysCtlDelay(SysCtlClockGet() / 3);
85      }
86  }
```