

1 Lab2 - Polling, Button

```
1 #include <stdint.h>
2 #include <stdbool.h>
3 #include "inc/hw_memmap.h"
4 #include "inc/hw_types.h"
5 #include "driverlib/sysctl.h"
6 #include "driverlib/gpio.h"
7 #include "inc/hw_gpio.h"
8 int32_t ButtonState = 0;
9 uint32_t g_w_delay = 2000000;
10 uint32_t g_RGB_delay = 20000000;
11 uint8_t g_flash_LED = 0;
12
13 void read_Switches(void) {
14     if (GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_4)) {
15         g_flash_LED = 0;
16     } else {
17         g_flash_LED = 1;
18     }
19 }
20
21 void RGB_FSM(void) {
22     static enum {ST_RED, ST_GREEN, ST_BLUE, ST_OFF} next_state;
23     if (g_flash_LED == 0) {
24         switch (next_state) {
25             case ST_RED:
26                 //Control_RGB_LEDs(1, 0, 0);
27                 GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|
GPIO_PIN_3, GPIO_PIN_1);
28                 SysCtlDelay(g_RGB_delay);
29                 next_state = ST_BLUE;
30                 break;
31             case ST_BLUE:
32                 //Control_RGB_LEDs(0, 1, 0);
33                 GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|
GPIO_PIN_3, GPIO_PIN_2);
34                 SysCtlDelay(g_RGB_delay);
35                 next_state = ST_GREEN;
36                 break;
37             case ST_GREEN:
38                 //Control_RGB_LEDs(0, 0, 1);
39                 GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|
GPIO_PIN_3, GPIO_PIN_3);
40                 SysCtlDelay(g_RGB_delay);
41                 next_state = ST_RED;
42                 break;
43
44             default:
45                 next_state = ST_RED;
46                 break;
47         }
48     }
49 }
50
51 void flash_FSM(void) {
52     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

```
1 #include <stdint.h>
2 #include <stdbool.h>
3 #include <string.h>
4 #include "inc/tm4c123gh6pm.h"
5 #include "inc/hw_memmap.h"
6 #include "inc/hw_types.h"
7 #include "inc/hw_gpio.h"
8 #include "driverlib/sysctl.h"
9 #include "driverlib/gpio.h"
10 #include "driverlib/timer.h"
11 #include "driverlib/interrupt.h"
12
13 #define W_DELAY 150
14 #define RGB_DELAY 1500
15
16 uint8_t g_flash_LED = 0;
17 bool timer1finish = 1;
18 bool timer0finish = 1;
19
20 uint32_t ui32Period;
21
22 void Timer1IntHandler(void);
23 void Timer0IntHandler(void);
24 void PORTF_IRQHandler(void);
25
26 void initialization(void)
27 {
28     SysCtlClockSet(SYSCTL_SYSDIV_5 | SYSCTL_USE_PLL | SYSCTL_XTAL_16MHZ
29         | SYSCTL_OSC_MAIN);
30     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
31     GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 |
32         GPIO_PIN_3);
33     GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, GPIO_PIN_4 | GPIO_PIN_0);
34     GPIOPadConfigSet(GPIO_PORTF_BASE, GPIO_PIN_4, GPIO_STRENGTH_2MA,
35         GPIO_PIN_TYPE_STD_WPU);
36
37     GPIOIntTypeSet(GPIO_PORTF_BASE, GPIO_PIN_4, GPIO_BOTH_EDGES);
38     GPIOIntEnable(GPIO_PORTF_BASE, GPIO_PIN_4);
39     GPIOIntRegister(GPIO_PORTF_BASE, PORTF_IRQHandler);
40
41     // Configure Timer1
42     SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1);
43     TimerConfigure(TIMER1_BASE, TIMER_CFG_ONE_SHOT);
44     IntEnable(INT_TIMER1A);
45     TimerIntRegister(TIMER1_BASE, TIMER_A, Timer1IntHandler);
46     TimerIntEnable(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
47
48     ui32Period = SysCtlClockGet() / 1000;
49     TimerLoadSet(TIMER1_BASE, TIMER_A, RGB_DELAY * ui32Period);
50
51     // Configure Timer0
52     SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
53     TimerConfigure(TIMER0_BASE, TIMER_CFG_ONE_SHOT);
54     IntEnable(INT_TIMER0A);
55     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
61 // Read_Switches_Timer()
62 void PORTF_IRQHandler(void)
63 {
64     // Check if the interrupt was triggered by PD4
65     if (GPIOIntStatus(GPIO_PORTF_BASE, true) & GPIO_INT_PIN_4)
66     {
67         // Toggle the LED or perform your action
68         if (GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_4))
69             g_flash_LED = 0; // Button released
70         else
71             g_flash_LED = 1; // Button pressed
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 |
97                     GPIO_PIN_3, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3);
98                 timer0finish = 0;
99                 TimerEnable(TIMER0_BASE, TIMER_A);
100                 next_state = W_wait;
101                 break;
102             case W_wait:
103                 if (timer0finish)
104                     next_state = OFF;
105                 break;
106             case OFF:
107                 GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 |
108                     GPIO_PIN_3, 0);
109                 timer0finish = 0;
```

```
108         TimerEnable(TIMERO_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

```

1 #include <stdint.h>
2 #include <stdbool.h>
3 #include <string.h>
4 #include "inc/tm4c123gh6pm.h"
5 #include "inc/hw_memmap.h"
6 #include "inc/hw_types.h"
7 #include "driverlib/sysctl.h"
8 #include "driverlib/interrupt.h"
9 #include "driverlib/gpio.h"
10 #include "driverlib/timer.h"
11 #define RS_PIN GPIO_PIN_5 // select pin 5 for RS
12 #define RW_PIN GPIO_PIN_6 // select pin 6 for RS
13 #define EN_PIN GPIO_PIN_7 // select pin 7 for EN
14 #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
15 #define ROW GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4
    // select pins 1~4 for ROW
16 #define COL GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6
    // select pins 4~6 for COL
17
18 void flushInput(uint32_t ui32Port, uint8_t ui8Pins);
19 void delayUs(int n);
20 void LCD_command(bool rs, bool rw, unsigned char data);
21 char *message_str1 = "Please Enter:";
22 char *message_str2 = " ";
23 int flag;
24 int n;
25 int cursor_pos = 0xC0;
26 int inputEnable = 1;
27 int main(void)
28 {
29     begin:
30     SysCtlClockSet(SYSCTL_SYSDIV_5 | SYSCTL_USE_PLL | SYSCTL_XTAL_16MHZ
        | SYSCTL_OSC_MAIN);
31     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
32     GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, DB_PIN);
33     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
34     GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, RS_PIN | RW_PIN | EN_PIN);
35     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
36     GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE, ROW);
37     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
38     GPIOPinTypeGPIOInput(GPIO_PORTC_BASE, COL);
39     GPIOPadConfigSet(GPIO_PORTC_BASE, COL, GPIO_STRENGTH_4MA,
        GPIO_PIN_TYPE_STD_WPU);
40     /* LCD configuration */
41     LCD_command(0, 0, 0x38); // SET FUNCTION: specify 8-bit interface,
        2 line display, and 5x7 dots (font)
42     LCD_command(0, 0, 0x08); // DISPLAY OFF: set display off
43     LCD_command(0, 0, 0x01); // DISPLAY CLEAR: set display clear
44     LCD_command(0, 0, 0x06); // ENTRY MODE SET: set cursor move
        direction as decreasing & display is not shifted
45     LCD_command(0, 0, 0x0F); // DISPLAY ON/OFF: set display on, cursor
        on, & cursor blinking on
46     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_PORTC_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_PORTC_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     }
```



```

97
98     GPIOWrite(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     GPIOWrite(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:
162         Data
163         GPIOPinWrite(GPIO_PORTA_BASE, RS_PIN, 0x00); // set RS as L
164     else
165         GPIOPinWrite(GPIO_PORTA_BASE, RS_PIN, 0x20); // set RS as H
166     if (rw == 0) // L: Write mode;
167         H: Read mode
168         GPIOPinWrite(GPIO_PORTA_BASE, RW_PIN, 0x00); // set RW as L
169     else
170         GPIOPinWrite(GPIO_PORTA_BASE, RW_PIN, 0x40); // set RW as H
171     delayUs(1);
172     GPIOPinWrite(GPIO_PORTA_BASE, EN_PIN, 0x80); // set H to enable
173     signal EN
174     GPIOPinWrite(GPIO_PORTB_BASE, DB_PIN, data); // assign DB0~DB7 with
175     "data"
176     delayUs(1);
177     GPIOPinWrite(GPIO_PORTA_BASE, EN_PIN, 0x00); // set H->L to enable
178     signal EN
179     delayUs(1);
180     if (rs == 0) // L: Command
181     {
182         if ((data == 0x01) | (data == 0x02) | (data == 0x03))
183             delayUs(1640); // Clear Display & Display/Cursor Home take
184             1.64ms
185         else
186             delayUs(40); // all the others commands require only 40us
187         to execute
188     }
189     else
190         delayUs(40); // Data Write takes 40us to execute
191 }
```

4 Lab5 - UART, Bluetooth, Interrupts

main_at.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
16         correctly enabled if the LED blinks slowly (around 2-second period)
17     1.2 Use the following AT commands: AT+UART? / AT+ROLE? / AT+ADDR?
18         to check the current configuration
19     1.3 Set the baud rate (AT+UART) of both the master and the slave
20         38400
21     1.4 Set the role (AT+ROLE) of the master 1, the slave 0
22     1.5 Enable the fixed-address mode (AT+CMODE) of the master
23     1.6 Bind (AT+BIND) the destination of the master to the slave
24         address (AT+ADDR?)
25 2. Disconnect bluetooth-EN and Tiva-VCC to disable AT mode
26 3. Complete, compile and run the code
27 */
28
29 #include <stdint.h>
30 #include <stdbool.h>
31 #include "inc/hw_memmap.h"
32 #include "inc/hw_types.h"
33 #include "driverlib/gpio.h"
34 #include "driverlib/pin_map.h"
35 #include "driverlib/sysctl.h"
36 #include "driverlib/uart.h"
37 #include "utils/uartstdio.h"
38 #include "inc/hw_ints.h"
39 #include "driverlib/interrupt.h"
40
41 void UART0IntHandler(void);
42 void UART5IntHandler(void);
43
44 int main(void) {
45     // set clock
46     SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
47                     SYSCTL_XTAL_16MHZ);
48
49     // enable UART0 and GPIOA to communicate with PC
50     SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
51     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
52     // configure PA0 for RX, PA1 for TX
53     GPIOPinConfigure(GPIO_PA0_U0RX);
54     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, 'O');
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
15         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?
17         to check the current configuration
18     1.3 Set the baud rate (AT+UART) of both the master and the slave
19         38400
20     1.4 Set the role (AT+ROLE) of the master 1, the slave 0
21     1.5 Enable the fixed-address mode (AT+CMODE) of the master
22     1.6 Bind (AT+BIND) the destination of the master to the slave
23         address (AT+ADDR?)
24 2. Disconnect bluetooth-EN and Tiva-VCC to disable AT mode
25 3. Complete, compile and run the code
26 */
27
28 #include <stdint.h>
```

```
25 #include <stdbool.h>
26 #include "inc/hw_memmap.h"
27 #include "inc/hw_types.h"
28 #include "driverlib/gpio.h"
29 #include "driverlib/pin_map.h"
30 #include "driverlib/sysctl.h"
31 #include "driverlib/uart.h"
32 #include "utils/uartstdio.h"
33 #include "inc/hw_ints.h"
34 #include "driverlib/interrupt.h"
35
36
37 void PORTF_IRQHandler(void);
38
39 int main(void) {
40     // set clock
41     SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
42         SYSCTL_XTAL_16MHZ);
43
44     // TODO: YOUR UART INITIALIZATION PROCEDURE
45     // enable UART5 and GPIOE to communicate with BLUETOOTH
46     SysCtlPeripheralEnable(SYSCTL_PERIPH_UART5);
47     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
48     // configure PE4 for RX, PE5 for TX
49     GPIOPinConfigure(GPIO_PE4_U5RX);
50     GPIOPinConfigure(GPIO_PE5_U5TX);
51     // set PORTE pin4 and pin5 as type UART
52     GPIOPinTypeUART(GPIO_PORTE_BASE, GPIO_PIN_4 | GPIO_PIN_5);
53     // set UART5 base address, clock and baud rate
54     UARTConfigSetExpClk(UART5_BASE, SysCtlClockGet(), 38400,
55         (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
56         UART_CONFIG_PAR_NONE));
57
58     // enable SW1 of GPIOF for button control
59     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
60     GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, GPIO_PIN_4);
61     GPIOPadConfigSet(GPIO_PORTF_BASE, GPIO_PIN_4, GPIO_STRENGTH_2MA,
62         GPIO_PIN_TYPE_STD_WPU);
63
64     // TODO: YOUR BUTTON CONTROL PROCEDURE
65     // you can do
66     // polling (i.e., checking button status and send data in a while
67     // loop), or
68     // interrupt (i.e., writing an interrupt function to check button
69     // status and send data)
70     GPIOIntTypeSet(GPIO_PORTF_BASE, GPIO_PIN_4, GPIO_BOTH_EDGES);
71     GPIOIntEnable(GPIO_PORTF_BASE, GPIO_PIN_4);
72     GPIOIntRegister(GPIO_PORTF_BASE, PORTF_IRQHandler);
73     IntMasterEnable();
74 }
75
76 void PORTF_IRQHandler(void)
77 {
78     // Check if the interrupt was triggered by PD4
79     if (GPIOIntStatus(GPIO_PORTF_BASE, true) & GPIO_INT_PIN_4)
80     {
81         char a;
82         // Toggle the LED or perform your action
83     }
84 }
```

```
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
16         correctly enabled if the LED blinks slowly (around 2-second period)
17     1.2 Use the following AT commands: AT+UART? / AT+ROLE? / AT+ADDR?
18         to check the current configuration
19     1.3 Set the baud rate (AT+UART) of both the master and the slave
20         38400
21     1.4 Set the role (AT+ROLE) of the master 1, the slave 0
22     1.5 Enable the fixed-address mode (AT+CMODE) of the master
23     1.6 Bind (AT+BIND) the destination of the master to the slave
24         address (AT+ADDR?)
25 2. Disconnect bluetooth-EN and Tiva-VCC to disable AT mode
26 3. Complete, compile and run the code
27 */
28
29 #include <stdint.h>
30 #include <stdbool.h>
31 #include "inc/hw_memmap.h"
32 #include "inc/hw_types.h"
33 #include "driverlib/gpio.h"
34 #include "driverlib/pin_map.h"
35 #include "driverlib/sysctl.h"
36 #include "driverlib/uart.h"
37 #include "utils/uartstdio.h"
38 #include "inc/hw_ints.h"
39 #include "driverlib/interrupt.h"
40
41 bool led_on = false;
42
43 void UART5IntHandler(void);
44
45 int main(void) {
```

```
42 // set clock
43 SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
44 SYSCTL_XTAL_16MHZ);
45
46 // TODO: YOUR UART INITIALIZATION PROCEDURE
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
61 // enable LED of GPIOF for display
62 SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
63 GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 |
64 GPIO_PIN_3);
65
66 // TODO: YOUR UART INTERRUPT INITIALIZATION PROCEDURE
67 UARTIntRegister(UART5_BASE, UART5IntHandler); // Register handler
68 UARTIntEnable(UART5_BASE, UART_INT_RX | UART_INT_RT); // Enable
69 interrupt
70 IntEnable(INT_UART5); // Enable the processor to handle UART5
71 interrupts
72 IntMasterEnable(); // Enable global interrupts
73 // set interrupt
74
75 while (true)
76 {
77     if (led_on) {
78         GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 |
79 GPIO_PIN_3, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3);
80     }
81     else {
82         GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 |
83 GPIO_PIN_3, 0);
84     }
85 }
86
87 // TODO: YOUR UART INTERRUPT HANDLER
88
89 // handler when Tiva receives data from UART5
90 void UART5IntHandler(void)
91 {
92     // get interrupt status
93     uint32_t ui32Status = UARTIntStatus(UART5_BASE, true);
94     // clear the interrupt signal
95     UARTIntClear(UART5_BASE, ui32Status);
96     // receive data from UART5
97     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

```
1 #include <stdint.h>
2 #include <stdbool.h>
3 #include "inc/hw_memmap.h"
4 #include "inc/hw_types.h"
5 #include "inc/hw_ints.h"
6 #include "inc/tm4c123gh6pm.h"
7 #include "inc/hw_gpio.h"
8 #include "driverlib/debug.h"
9 #include "driverlib/sysctl.h"
10 #include "driverlib/adc.h"
11 #include "driverlib/interrupt.h"
12 #include "driverlib/gpio.h"
13 #include "driverlib/timer.h"
14
15 uint32_t ui32ADC0Value[4];
16 volatile uint32_t ui32VolAvg;
17 volatile uint32_t ui32TempValueC;
18 volatile uint32_t ui32TempValueF;
19
20 void ADC0IntHandler(void) {
21     //
22     // Clear the ADC interrupt flag
23     //
24     ADCIntClear(ADC0_BASE, 1);
25
26     //
27     // Get the ADC data
28     //
29     ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0Value);
30
31     //
32     // Calculate average voltage value
33     //
34     ui32VolAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] + ui32ADC0Value[3] + 2) / 4;
35
36     //
37     // TODO: Convert the voltage to Celsius and Fahrenheit
38     //
39     ui32TempValueC = 147.5 - ((75 * ui32VolAvg * 3.3) / 4096);
40     ui32TempValueF = (ui32TempValueC * 1.8) + 32;
41 }
42
43 int main(void) {
44     //
45     // Set up the system clock
46     //
47     SysCtlClockSet(SYSCTL_SYSDIV_5 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
48                     SYSCTL_XTAL_16MHZ);
49
50     //
51     // The ADC0 peripheral must be enabled for use.
52     //
53     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_ADCOSS1);
93 ADCIntEnable(ADC0_BASE, 1);
94
95 TimerControlTrigger(TIMER0_BASE, TIMER_A, true);
96 TimerEnable(TIMER0_BASE, TIMER_A);
97 while(1) {
98 }
99 }
```