ENCM 511 Final Project

Common

A set of common source and header files (common.c/common.h) are used to define commonly used helper macros and other global variables used across other source files in the program.

```
17  #define LED0    LATBbits.LATB5
18  #define LED1    LATBbits.LATB6
19  #define LED2    LATBbits.LATB7
20
21  #define PB0     PORTAbits.RA4
22  #define PB1     PORTBbits.RB8
23  #define PB2     PORTBbits.RB9
24
25  #define MESSAGE_HOME "\033[1;1H"
26  #define TIMER_HOME "\033[2;1H"
27  #define ADC_HOME   "\033[3;1H"
28  #define DR_HOME    "\033[4;1H"
29  #define CLEAR_SCREEN "\033[2J"
30  #define HIDE_CURSOR "\033[?25l"
31
32  #define ADC_MESSAGE "ADC VALUE: "
33  #define DUTY_MESSAGE "DUTY VALUE: "
34  #define SET_MESSAGE "Please enter a time to countdown"
35  #define COUNTDOWN_MESSAGE "The time remaining on the counter is:"
36  #define WELCOME_MESSAGE "Please press PB0 to start"
37  #define FINISH_MESSAGE "The timer has finished counting down"
38
39  #define ISR_PRIORITY 3
```

Shown above are various macros to improve code readability and prevent repetition of hardcoded literal values.

```
41  #define SET_BIT(flags, n)       ((flags) |= (1 << (n)))
42  #define CHECK_BIT(flags, n)     (((flags) >> (n)) & 1)
43  #define CLEAR_BIT(flags, n)     ((flags) &= ~(1 << (n)))
44  #define TOGGLE_BIT(flags, n)    ((flags) ^= (1 << (n)))
45
46  extern uint8_t pb_stat;      // Bit-field for button status flags
47  #define PB0_HELD_FLAG    0   // Flag set to indicate PB0 is being held and program has not acted on that yet
48  #define PB0_CLICKED_FLAG 1   // Flag set to indicate PB0 has been clicked
49  #define PB1_HELD_FLAG    2   // Flag set to indicate PB1 is being held and program has not acted on that yet
50  #define PB1_CLICKED_FLAG 3   // Flag set to indicate PB1 has been clicked
51  #define PB2_HELD_FLAG    4   // Flag set to indicate PB2 is being held and program has not acted on that yet
52  #define PB2_CLICKED_FLAG 5   // Flag set to indicate PB2 has been clicked
53
54  // Define the transition combination to compare with pb_stat
55  #define PB0_CLICKED         (1 << PB0_CLICKED_FLAG)
56  #define PB1_CLICKED         (1 << PB1_CLICKED_FLAG)
57  #define PB2_CLICKED         (1 << PB2_CLICKED_FLAG)
58  #define PB0_HELD            (1 << PB0_HELD_FLAG)
59  #define PB1_HELD            (1 << PB1_HELD_FLAG)
60  #define PB2_HELD            (1 << PB2_HELD_FLAG)
61  #define PB0_PB1_HELD       ((1 << PB0_HELD_FLAG) | \
62                               (1 << PB1_HELD_FLAG))
63  #define PB0_PB2_HELD       ((1 << PB0_HELD_FLAG) | \
64                               (1 << PB2_HELD_FLAG))
65  #define PB1_PB2_HELD       ((1 << PB1_HELD_FLAG) | \
66                               (1 << PB2_HELD_FLAG))
67  #define PB1_PB2_CLICKED   ((1 << PB1_CLICKED_FLAG) | \
68                               (1 << PB2_CLICKED_FLAG))
```

Shown above are helper macros for manipulating bitfields along with a variable to hold button states, which is used in conjunction with the bitfield helper macros in the code to check and modify button states. Additionally, there are some logical defines that can be compared to the pb_stat variable in order to easily check for a

particular pushbutton condition without manually using the bitfield positions. Example usages in code are shown below:

```
SET_BIT(pb_stat,PB0_CLICKED_FLAG);
```

```
if (pb_stat == PB1_PB2_CLICKED){
```

Show below are the global variables defined in common.c, explained by the comments on each line.

```
4   // Define globals here (only once)
5   SemaphoreHandle_t uart_tx_sem = NULL;       // mutex to use uart tx
6   QueueHandle_t xUartTransmitQueue = NULL;    // queue to transmit on the UART
7
8   SemaphoreHandle_t uart_rx_sem = NULL;       // mutex to use uart rx
9   QueueHandle_t xUartReceiveQueue = NULL;     // queue to transmit on the UART
10
11
12  uint16_t global_adc_value = 0;           // global ADC value
13  SemaphoreHandle_t adc_value_sem = NULL;     // and its mutex
14
15  uint16_t set_time;                   //Global variable that holds the set time by the user
16  SemaphoreHandle_t countdown_sem = NULL;     //Mutex for safely changing the countdown seconds
17
18  SemaphoreHandle_t uart_tx_queue_sem = NULL;
19
20  SemaphoreHandle_t state_sem = NULL;
21
22  extern TaskHandle_t DoUartAdcTaskHandle = NULL;
23  extern TaskHandle_t DoUartTransmitTaskHandle = NULL;
24  extern TaskHandle_t DoUartRecieveTaskHandle = NULL;
25  extern TaskHandle_t DoTimerTaskHandle = NULL;
26  extern TaskHandle_t SetTimerTaskHandle = NULL;
27  extern TaskHandle_t DoButtonTaskHandle = NULL;
28  extern TaskHandle_t DoStateTransitionHandle = NULL;
29
30  extern states current_state = waiting_state;
31  extern states next_state = waiting_state;
32  extern TaskHandle_t DoLED2TaskHandle = NULL;
33  extern TaskHandle_t DoLED01Handle = NULL;
```

Peripheral Configuration

Below is initialization for the ADC peripheral, explained in the comments line by line.

```
4   void do_adc_init(void)
5   {
6       AD1CON2bits.PVCFG = 0;      // Selects Positive voltage reference
7       AD1CON2bits.NVCFG0 = 0;     // Selects Negative voltage reference
8
9       //Setup Channel 0 sample A
10      AD1CHSbits.CH0NA = 0;       // Negative input is selected as gnd
11      AD1CHSbits.CH0SA = 5;       // Positive input selected as AN5 (PIN7)
12
13      AD1CON3bits.ADCS = 255;     // AD Conversion clock is set to 256 * Tcy
14      AD1CON1bits.SSRC = 7;       // ADC occurs based off SAMP
15      AD1CON1bits.FORM = 0;       // Data output is absolute decimal unsigned right justified
16      AD1CON5bits.ASEN = 0;       // Disable auto scan
17      AD1CON1bits.DMAEN = 0;      // Disable DMA
18      AD1CON2bits.SMPI = 0;       // Interrupts at the completion of each sample
19      AD1CON1bits.MODE12 = 0;     // 10-bit ADC mode
20      AD1CON1bits.ASAM = 0;       // Sampling starts when SAMP is set manually
21
22      AD1CON1bits.ADON = 1;       // Enable ADC
23      AD1CON1bits.SAMP = 0;       // Don't sample yet
24  }
```

Below is initialization for the GPIO peripheral, explained in the comments line by line.

```
19  void do_button_init(void)
20  {
21      TRISBbits.TRISB5 = 0;   // Set to output (LED0)
22      TRISBbits.TRISB6 = 0;   // Set to output (LED1)
23      TRISBbits.TRISB7 = 0;   // Set to output (LED2`)
24
25      TRISBbits.TRISB3 = 1;   // Set to input (ADC_input)
26
27      TRISAbits.TRISA4 = 1;   // Set to input (PB0)
28      TRISBbits.TRISB8 = 1;   // Set to input (PB1)
29      TRISBbits.TRISB9 = 1;   // Set to input (PB2)
30
31      IOCPUAbits.IOCPA4 = 1;  // Enable pull-up (PB0)
32      IOCPUBbits.IOCPB8 = 1;  // Enable pull-up (PB1)
33      IOCPUBbits.IOCPB9 = 1;  // Enable pull-up (PB2)
34
35      PADCONbits.IOCON = 1;   // Enable interrupt-on-change (IOC)
36
37      IOCNAbits.IOCNA4 = 1;   // Enable high-to-low IOC (PB0)
38      IOCPAbits.IOCPA4 = 1;   // Enable low-to-high IOC (PB0)
39      IOCNBbits.IOCNB8 = 1;   // Enable high-to-low IOC (PB1)
40      IOCPBbits.IOCPB8 = 1;   // Enable low-to-high IOC (PB1)
41      IOCNBbits.IOCNB9 = 1;   // Enable high-to-low IOC (PB2)
42      IOCPBbits.IOCPB9 = 1;   // Enable low-to-high IOC (PB2)
43
44      IFS1bits.IOCIF = 0;             // Clear system-wide IOC flag
45      IPC4bits.IOCIP = ISR_PRIORITY;  // Set IOC priority
46      IEC1bits.IOCIE = 1;             // Enable IOC
47      return;
48  }
```

Below is initialization for the ADC peripheral, explained in the comments line by line.

```
4   void do_adc_init(void)
5   {
6       AD1CON2bits.PVCFG = 0;      // Selects Positive voltage reference
7       AD1CON2bits.NVCFG0 = 0;     // Selects Negative voltage reference
8
9       //Setup Channel 0 sample A
10      AD1CHSbits.CH0NA = 0;       // Negative input is selected as gnd
11      AD1CHSbits.CH0SA = 5;       // Positive input selected as AN5 (PIN7)
12
13      AD1CON3bits.ADCS = 255;     // AD Conversion clock is set to 256 * Tcy
14      AD1CON1bits.SSRC = 7;       // ADC occurs based off SAMP
15      AD1CON1bits.FORM = 0;       // Data output is absolute decimal unsigned right justified
16      AD1CON5bits.ASEN = 0;       // Disable auto scan
17      AD1CON1bits.DMAEN = 0;      // Disable DMA
18      AD1CON2bits.SMPI = 0;       // Interrupts at the completion of each sample
19      AD1CON1bits.MODE12 = 0;     // 10-bit ADC mode
20      AD1CON1bits.ASAM = 0;       // Sampling starts when SAMP is set manually
21
22      AD1CON1bits.ADON = 1;       // Enable ADC
23      AD1CON1bits.SAMP = 0;       // Don't sample yet
24  }
```

Below is initialization for the Timer 2 peripheral, explained in the comments line by line.

```
6   void do_timer_init(void)
7   {
8       //Timer 2 setup
9
10      T2CONbits.T32 = 0;              // Operate timers 2 & 3 as separate 16-bit timers
11
12      T2CONbits.TCKPS = 2;            // set prescaler to 1:64
13      IPC1bits.T2IP = ISR_PRIORITY;   // Interrupt priority
14      IFS0bits.T2IF = 0;              // clear interrupt flag
15      IEC0bits.T2IE = 1;              // enable interrupt
16      PR2 = 62500;                    // set period for 1s
17
18      return;
19  }
```

Below is initialization for the Timer 3 peripheral, explained in the comments line by line.

```
12  void do_timer3_init(void)
13  {
14      //Timer 3 setup
15      T2CONbits.T32 = 0;              // Operate timers 2 & 3 as separate 16-bit timers
16
17      T3CONbits.TON = 0;              //Turn timer 3 off
18      T3CONbits.TCKPS = 0;            // set prescaler to 1:1
19      IPC2bits.T3IP = ISR_PRIORITY;   // Interrupt priority
20      IFS0bits.T3IF = 0;              // clear interrupt flag
21      IEC0bits.T3IE = 1;              // enable interrupt
22      PR3 = 8000;                     // set period for 2ms
23      T3CONbits.TON = 1;              //Turn timer 3 on
24      return;
25  }
```

Below is initialization for the UART 2 peripheral, explained in the comments line by line.

```
18  void do_uart_transmit_init(void)
19  {
20      xUartTransmitQueue = xQueueCreate(64, sizeof(uint8_t));
21
22      #ifndef UART2_CONFIG
23      #define UART2_CONFIG
24      RPINR19bits.U2RXR = 11;     // Assign U2RX to RP11 (pin 22)
25      RPOR5bits.RP10R = 5;        // Assign RP10 (pin 21) to U2TX
26
27      U2MODEbits.BRGH = 1;        // High baud rate is enabled
28      U2MODEbits.WAKE = 1;        // UART is awake and will trigger interrupt when Rx detected
29
30      U2BRG = 8;                  // Baud rate = 115200
31
32      U2STAbits.UTXISEL0 = 0;     // Interrupt when a character is transferred to the Transmit Shift Register
33      U2STAbits.UTXISEL1 = 0;     // (this implies there is at least one character open in the transmit buffer)
34      U2STAbits.URXEN = 1;        // Receive is enabled, U2RX pin is controlled by UART2
35      U2STAbits.UTXEN = 1;        // Transmit is enabled, U2TX pin is controlled by UART2
36      U2STAbits.URXISEL = 0b00;   // Interrupt is set when any character is received and transferred from the RSR to the receive buffer
37
38      IFS1bits.U2TXIF = 0;        // Set flag to 0
39      IPC7bits.U2TXIP = 3;        // Set priority
40      IEC1bits.U2TXIE = 1;        // Enable interrupt
41
42      IFS1bits.U2RXIF = 0;        // Set flag to 0
43      IPC7bits.U2RXIP = 4;        // Set priority
44      IEC1bits.U2RXIE = 1;        // Enable interrupt
45
46      U2MODEbits.UARTEN = 1;      // Enable UART Rx
47
48      U2STAbits.UTXEN = 1;        // Enable UART Tx
49
50      Disp2String(HIDE_CURSOR);
51      Disp2String(CLEAR_SCREEN);
52
53      return;
54      #endif
55  }
```

Implemented Functions/Tasks

For brevity of this report, full code for each function/task will not be pasted here, but the functionality of each will be explained. Full code can be found in the project submission.

void vDoAdcTask(void * pvParameters)

This function is a FreeRTOS task that runs every 10ms. It performs the ADC conversion (reading the potentiometer voltage), and if in an appropriate state will also display the value over UART. Mutexes are used to control access to the ADC reading value variable and the UART transmission queue.

void vDoButtonTask(void *pvParameters)

This function is a FreeRTOS task that runs every 10ms. It contains logic to read current and record past button input values and then update the corresponding flags (clicked, held, etc) in pb_stat for the rest of the program to reference.

void vDoLED01Task(void *pvParameters)

This function is a FreeRTOS task that runs every 250ms. If in the timer_finished state, it toggles LED0 and LED1 alternately. Otherwise it turns off LED0.

void vDoLED2Task(void *pvParameters)

This function is a FreeRTOS task that runs every 150ms. Depending on the state, LED2 will either be pulsed (breath), blink, or not blink, taking into account the intensity setting for blinking. Also depending on the state, the duty ratio is displayed on the UART terminal. Mutexes are used to control access to the ADC reading value variable and the UART transmission queue.

void vDoStateTransitionTask(void * pvParameters)

This function is a FreeRTOS task that runs every 100ms. It transitions the state of the program according to the logic defined in the diagram (see later section). It also sends a command to clear the UART terminal screen, and depending on state will also send either a welcome or timer finished message. Mutexes are used to control access to the state variable and the UART transmission queue.

void vDoTimerTask(void *pvParameters)

This function is a FreeRTOS task that runs every 200ms. When the program is in a state with the timer running, this task continuously updates the display of the remaining time.

void vDoUartRecieveTask(void * pvParameters)

This function is a FreeRTOS task that runs every 10ms. When it runs, it reads the UART receive buffer, and if the buffer holds the value of a new character it will be added to the UART receive queue (FreeRTOS queue). A mutex controls access to the UART receive buffer.

void vDoUartTransmitTask(void * pvParameters)

This function is a FreeRTOS task that runs every 10ms. If there is an item in the UART transmit queue (FreeRTOS queue), it will be displayed by putting it in the UART 2 transmit buffer. A mutex controls access to the UART transmit buffer.

void vSetTimerTask(void * pvParameters)

This function is a FreeRTOS task that runs every 100ms. When in the set_timer state, this task handles the display of the UART terminal screen where the user enters the desired time. Mutexes are used in accessing the current state, countdown value, and UART transmit and receive queues.

Program Flow Diagram

Below is a diagram showing the flow of the program, with the current state labelled within each bubble, and state transition logic shown on the arrows.