EGR 326 - 903

Lab 2
Input Interfacing with the MSP432 Using Interrupts

Corey Moura
Xue Hua

Date performed : 7 September 2019
Date submitted : 14 September 2019

Professor Kandalaft

Objectives

This weeks lab objectives were to interface external and internal LEDs of the MSP432 with external momentary push buttons. The control of the LED outputs was to be handled using the MSP432 internal systick timer interrupt along with the internal general purpose input/output interrupts (GPIOs).

Equipment

- MSP432 P401R Microcontroller
- RGB LED
- PNP Transistor
- 100 Ohm Resistor
- Prototyping board

Results

Part 1: Controlling the blink rate of an LED using pushbutton switches

The RGB LED was used for this portion of the lab along with two push button switches. The red and green pins where disconnected, and only the blue LED of the RGB was used. The resistor value in the circuit is 100 ohms. The circuit from Lab 1 was used with the modifications described above. The diagram of this can be seen below.

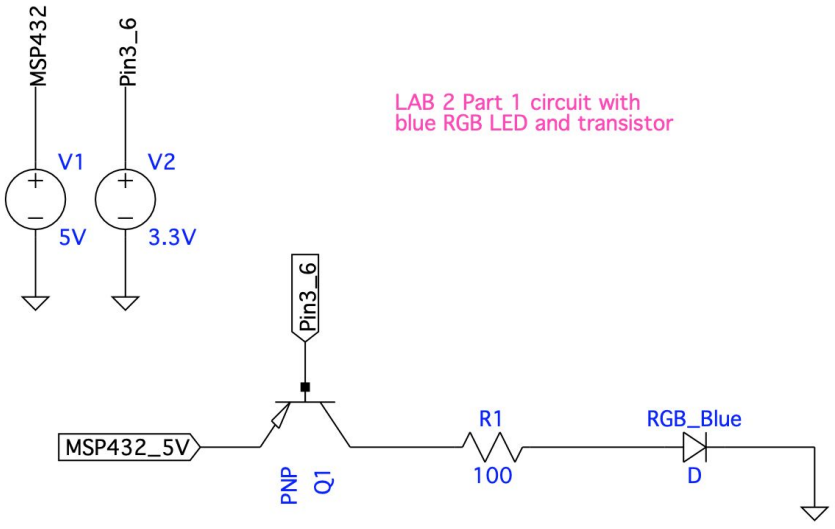
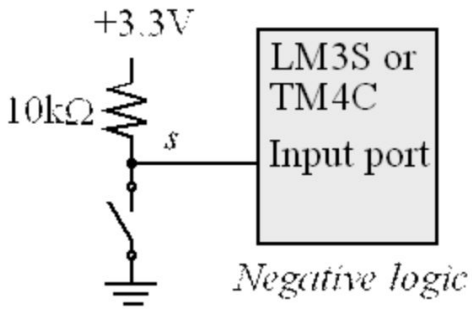


Figure 1. Blue RGB LED circuit layout

The two switches were wired into a seperate circuit. This circuits design involved the MSP432 internal pullup resistor. The pull resistor logic can be seen in Figure 2 while the button circuit can be seen in Figure 3.



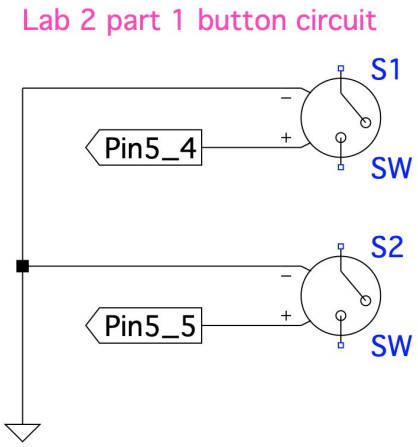


Figure 2. Pull up resistor internal logic buttons

Figure 3. Switch logic for the push buttons

The full MSP432 pin layout for this circuit can be seen below in table 1.

Table 1. Part 1 Pin Table

On-Board LED Color	Pin
Blue (output)	*3.6*
Button 1 (input)	*5.4*
Button 2 (input)	*5.5*

The program was written using the driverlib functions to initialize the interrupts and used to create the skeleton of the handlers, this can bee seen in the code provided in the appendix under “Part 1 Code”. The frequency control was determined using the equations for frequency and period as seen in equations (1) and (2).

$$Period\ (T) = \frac{1}{frequency\ (F)} \qquad (1)$$

$$Total\ number\ of\ \ cycles\ (n) = total\ times\ (s) \cdot \frac{1\ cycle\ (n)}{1\ period\ (T)} \qquad (2)$$

To generate number of cycles required to generate a 0.5 Hz frequency was calculated using equation (1) and (2).

Period of one MSP432 clock cycle was found using equation (1),

$$Period,\ T = \frac{1}{48\ MHz}$$

$$T = 2.083\ E-6\ \ s\ [\text{seconds}]$$

Total number of cycles was found using equation (2),

$$Total\ number\ of\ \ cycles = 0.500\ s \cdot \frac{1\ cycle}{2.083\ \mu s}$$

$$n = 24\ 0038\ cycles$$

The number of cycles above was used to fill a conditional statement which checks for this frequency. If the frequency is lower than this, then the LED will be turned off. Additionally, the LED is turned completely on when the period becomes to low and approaches zero.

The button debounce was handled as described in lab 1. The code used in lab 1 for button debounce was used unmodified throughout in this lab.

Part 2: Sequencing colors of RGB LED using timer interrupt and button

The RGB input pins for red and green were reconnected to the circuit and one of the push-buttons was disconnected from the MSP432. The circuit for this portion of the lab can be seen below in figure 4. Note that this portion of the lab used the on-board RGB with different ports than the ones seen below. The circuit below was used to confirm the programs functionality before switching the pins.

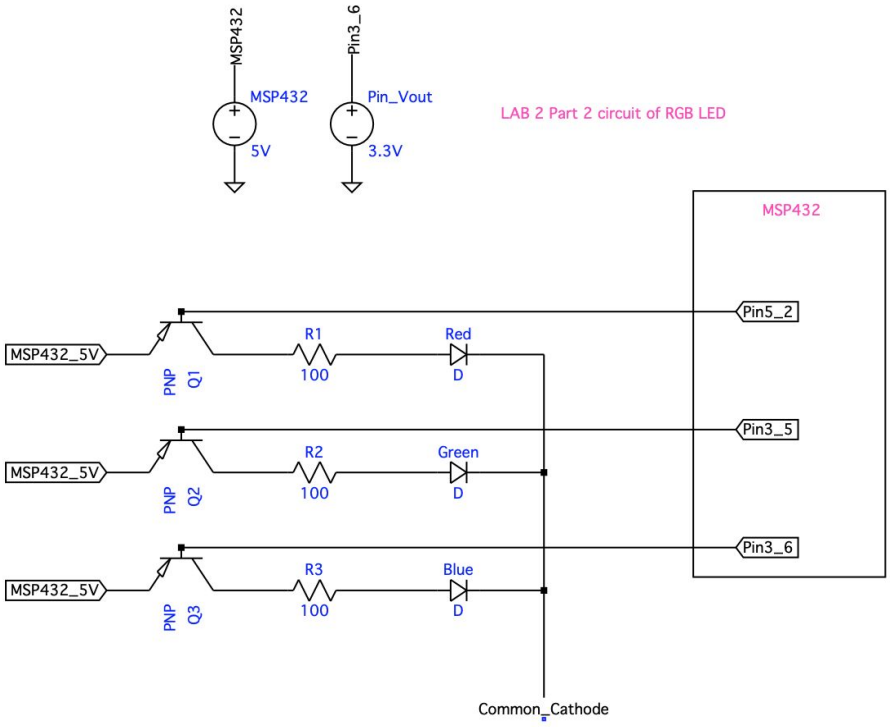


Figure 4. Common Cathode Circuit for RGB LED

The button circuit remained the same, with the exception that one push button was removed. The button circuit from part 1 can be referenced for this circuit.

The on-board LED was controlled by the pins seen in the table 2 below:

Table 2. On-Board LED Pin Table

On-Board LED Color	Pin
Red (output)	*2.0*
Green (output)	*2.1*
Blue (output)	*2.2*
Button 1 (input)	*5.4*

The systick timer interrupt was explicitly assigned to be used for this portion. The systick timer is a countdown timer internal to the MSP432. It can be used as an interrupt or without the interrupt enabled. The systick timer in this part of the lab was used as an interrupt. Being as an interrupt, the timer will countdown from the load value. When it reaches the value zero, a flag is

set and the interrupt handler is called. Additionally, the timer will automatically reset to its load value and begin counting down once again.

The “timeout” variable flag gets set when the systick interrupt handler is called. The flag is used as a parameter along with the button variable flag to access the state-machine. If the button is held, the state machine will increment through the color sequence in the state machine. If the button is released, then the program will not enter the state machine and the last LED to be illuminated will remain on.

Part 3: Controlling LED sequence with a timer and GPIO button interrupts

Three separate interrupts were used in this part, one systick interrupt, and two GPIO interrupts for the buttons. Pin layout for this section of the lab is shown in table 3.

Table 3. Part 3 Pin Table

On-Board LED Color	Pin
Red (output)	*2.0*
Green (output)	*2.1*
Blue (output)	*2.2*
Button 1 (input)	*5.4*
Button 2 (input)	*5.5*

The button circuit is identical to the one annotated in part 2 of this report. Additionally, since the on-board LEDs were used again, there is no LED circuit schematic for this portion of the lab.

The driver lib was used to initiate the GPIO interrupt ports and pins. The drivelib handlers and functions where also used in the stub functions before filling in the specific logic for this program.

Two buttons were used for different functions of the program and therefore it was critical to determine which button was pressed. To do this, the same handler was called when a button of port5 is pressed, but we distinguish the difference between buttons via the pin values and the logic in the program. The program can be seen in the appendix under the header “Part 3 Code”.

Conclusion

The systick timer, and two GPIO interrupts where used in this lab. State machines were created to interface the interrupt logic from the buttons with the systick timer. The onboard RGB LED was used in parts 2 and 3 to assign outputs based on the state machines current state, while part 1 used an external LED. The program was written in a modular way where each function has its own specific duty. When many things need to be controlled simultaneously, compartmentalizing the code allows for a more readable, debuggable, and robust program.

Appendix

PART 1 CODE

```
/*****************************************************************************************************************************************
 *  Author:          Corey Moura & Xue Hue
 *  Lab:             2.1 Controlling the blink rate of an LED using push button switches.
 *  Date:            9/4/19
 *  Instructor:      Dr. Kandalaft
 *
 *  Description:     In this part, you will drive an external LED (your choice of color) interfaced to a port pin on the MSP432
 *                   (similar to lab #1). You will also use two external pushbuttons to interface with other port pins on the
 *                   MSP432.  These two pushbuttons will control the blink rate of the LED using interrupts.
 *
 *                   The first button will start the LED to blink at 2 sec intervals (on one second, off one second).  Each
 *                   press of the button will double the blink rate. The second button will stop the LED from blinking (if the
 *                   blink rate is .5 Hz), or reduce the blink rate by a factor of 2 if the blink rate is faster than .5 Hz.
 *                   Pressed buttons should be implemented using INTERRUPTS.
 *
 *                   Holding down a button will not change the blink rate more than pressing and releasing a button.
 *
 *  Notes:           Using driverlib to initialize the GPIO and Systic interrupts.
 *
 *
 *****************************************************************************************************************************************/


#include "driverlib.h"                  // Driverlib library
#include <stdint.h>                     // A set of typedefs that specify exact-width integer types
#include <stdbool.h>                    // Allow boolean variable to be used
#include <stdio.h>                      // Standard input output for printf etc.
#include <stdlib.h>                     // Standard C-programming library
#include <string.h>                     // Allows the use of strings
#include "msp.h"                        // Another TI library of some sort



void initializeAll();                   // Called to initialize all of the MSP features used in the program.
void SysTick_Init_interupt();           // Called from initialize all, initialization of the systic timer
void gpioHandler_Init();                // Called from initialize all, initialization of the GPIO buttons as interrupts
void LED_init();                        // Called from initialize all, used to initialize the LED pins
void button_init();                     // Called from initialize all, used to initialize the button pins

void toggleControl();                   // Used to toggle the LED
void blue_On();                          // Used to turn on the specified LED
void blue_Off();                        // Used to turn off the specified LED

void speedControl();                    // Function used to increase the period or decrease
void decreaseSpeed();                       // Does math to increase the systic period load value
void increaseSpeed();                   // Does math to decrease the systic period load value

void PORT5_IRQHandler(void);            // Interrupt handler used to set the button pressed flags

void buttonCheck();                     // Uses the debounce to confirm a button is pressed
uint8_t DebounceSwitch_1();             // Debounce switch 1
uint8_t DebounceSwitch_2();             // Debounce switch 2


volatile uint8_t buttonFlag_1 = 0;                      // Flag set in GPIO handler when the button is pressed
volatile uint8_t buttonFlag_2 = 0;                      // Flag set in GPIO handler when the button is pressed

volatile uint8_t btn1Pressed = 0;                      // Assigned 1 when the button has been confirmed to be pressed
volatile uint8_t btn2Pressed = 0;                       // Assigned 1 when the button has been confirmed to be pressed

volatile uint8_t speedAssigned = 0;                     // Prevents continual increment if the period and state

volatile uint32_t timeout = 0;                          // Flag set from the syystic interrupt
volatile uint32_t sys_loadVal = 375000;                 // Initial value assigned to the systic timer
```

```
void main(void){

          WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;                 // stop watchdog timer

          initializeAll();                                            // Initialize the pins

          while(1){

              buttonCheck();                                          // Check the buttons
              toggleControl();                                        // Toggle the LED


              if((btn1Pressed || btn2Pressed) && speedAssigned == 0){

                    speedControl();                             // Increment or decrement the period of the Systick Load value
                    speedAssigned = 1;                                // The speed has been set for this button press

              }

              if(!(btn1Pressed) && !(btn2Pressed))    speedAssigned = 0; // Reset the speedAssigned if the button is released
          }
}


/*****************************************************************************************************************/
/* Check the debounce of the buttons.  Assigns the variables seen below and resets the interrupt flags.
/*****************************************************************************************************************/
void buttonCheck(){
          if (buttonFlag_1 && DebounceSwitch_1()){
          buttonFlag_1 = 0;                           // Reset the interrupt flag
          btn1Pressed = 1;                         // Set the button variable to the pressed condition
          }
          else btn1Pressed = 0;

          if (buttonFlag_2 && DebounceSwitch_2()){
          buttonFlag_2 = 0;                           // Reset the interrupt flag
          btn2Pressed = 1;                         // Set the button variable to the pressed condition
          }
          else btn2Pressed = 0;                              // If the above conditions are not met, then the button is not
pressed

          return;
}


/*****************************************************************************************************************/
/* DEBOUNCER 1 - Sample code provided by Dr. Kandalaft  */
/*****************************************************************************************************************/
uint8_t DebounceSwitch_1(){

          static uint16_t check1 = 0;                                 // Variable reset every time function is called

          check1 = (check1<<1) | ((P5IN & BIT4)>>1) | 0xfc00;         // Bitwise OR with shifts to complete in 1 cycle

          _delay_cycles(15000);                                       // Delay the second check to confirm debounce

          if(check1 == 0xfc00){
          return 1;                                          // If it is the correct value then button is pressed
          }
          return 0;                                                        // If it is not the correct value then return 0
}


/*****************************************************************************************************************/
/* DEBOUNCER 2 - Sample code provided by Dr. Kandalaft  */
/*****************************************************************************************************************/
uint8_t DebounceSwitch_2(){

          static uint16_t check2 = 0;                                 // Variable reset every time function is called

          check2 = (check2<<1) | ((P5IN & BIT5)>>1) | 0xfc00;         // Bitwise OR with shifts to complete in 1 cycle

    _delay_cycles(15000);                                             // Delay the second check to confirm debounce

          if(check2 == 0xfc00){
          return 1;                                                   // If it is the correct value then button is pressed
          }
          return 0;                                                   // If it is not the correct value then return 0
```

```
}


/*****************************************************************************************************************/
/* Controls the toggle of the LED based on the flag set by the systic interrupt.
/*****************************************************************************************************************/
void toggleControl(){
          if(timeout){
          P3->OUT ^= BIT6;                                        // Toggle pin 4 every second
          timeout = 0;                                            // Reset systic timer flag
          }
}


/*****************************************************************************************************************/
/* Depending on which button is pressed, the functions below that control speed will be called.
/*****************************************************************************************************************/
void speedControl(){

          if(btn1Pressed) increaseSpeed();                        // Decrease the period, increase the blink rate
          if(btn2Pressed) decreaseSpeed();                         // Increase the period, decrease the blink rate

          return;
}


/*****************************************************************************************************************/
/* Increases the speed of the LED blink rate by decreasing the period of the systic interrupt.
/*****************************************************************************************************************/
void increaseSpeed(){

          if(sys_loadVal < 46872){
          SysTick->CTRL = 0x00000005;                       // Systic interrupt is disabled when cutoff value reached
          blue_On();                                        // Turn on the LED to prevent odd behavior
          }

          else{
          SysTick->CTRL = 0x00000007;                             // Ensure the interrupt is enabled
          sys_loadVal = sys_loadVal / 2;                              // Half the period via division by 2
          }

          SysTick->LOAD = sys_loadVal;                                // Assign the Load value of the timer

}


/*****************************************************************************************************************/
/* Decreases the speed of the LED blink rate by increasing the period of the systic interrupt.
/*****************************************************************************************************************/
void decreaseSpeed(){

          if(sys_loadVal > 2400000){
          SysTick->CTRL = 0x00000005;                       // Systic interrupt is disabled when cutoff value reached
          blue_Off();                                      // If the freq gets to 0.5Htz, the LED is turned off
          }

          else{
          SysTick->CTRL = 0x00000007;                             // Ensure the interrupt is enabled
          sys_loadVal = sys_loadVal * 2;                              // Double the period via multiplication by 2
          }

          SysTick->LOAD = sys_loadVal;                                // Assign the Load value of the timer

}

/*****************************************************************************************************************/
/* Systic Interrupt Handler
/*****************************************************************************************************************/
void Systick_Handler(void){
          timeout = 1 ;                                           // set flag for timeout of SysTick, rest in main
          //intervalCnt++ ;                                       // increment interval timer by 1ms, reset in main ( )
}


/*****************************************************************************************************************/
/* GPIO ISR handler for the buttons.
/*****************************************************************************************************************/
void PORT5_IRQHandler(void){
```

```
        uint32_t status;                                        // Declare variable of flag of the interrupt

        status = MAP_GPIO_getEnabledInterruptStatus(GPIO_PORT_P5);  // Assign status a value if the interrupt was called
        MAP_GPIO_clearInterruptFlag(GPIO_PORT_P5, status);          // After assignment above, clear the interrupt flag

        if(status & GPIO_PIN4){
        buttonFlag_1 = 1;                               // If the first button was pressed, set the flag
        }
        if(status & GPIO_PIN5){
        buttonFlag_2 = 1;                               // If the second button was pressed, set the flag
        }
}


/*****************************************************************************************************************/
/* LED outputs from the MSP432.
/*****************************************************************************************************************/
void blue_On(){     P3->OUT |= BIT6;    }                   // Turn on the LED
void blue_Off(){    P3->OUT &=~ BIT6;   }                   // Turn off the LED
void blue_Toggle(){ P3->OUT ^= BIT6;    }                   // Toggle the LED


/*****************************************************************************************************************/
/* Call out to the functions and initialize the pins and timers used in the program.
/*****************************************************************************************************************/
void initializeAll(){

        SysTick_Init_interupt();                        // Systic initialization
        gpioHandler_Init();                             // GPIO interrupt initialization
        LED_init();                                     // LED initialization (outputs)
        button_init();                                // Button initializations (inputs)

}

/*****************************************************************************************************************/
/* Systick timer initialization with interrupts.
/*****************************************************************************************************************/
void SysTick_Init_interupt(){
        SysTick->CTRL = 0;                              // Disable SysTick during step
        SysTick->LOAD = sys_loadVal;                    // Max reload value 0xFFFFFFFF but: for 1ms use 3000 - flag set every
millisecond
        SysTick->VAL = 0;                               // Any write to current clears it
        SysTick->CTRL = 0x00000007;                     // Enable SysTick, 3MHz, 0x00000007 interrupts & 0x00000005 no
interrupts
}


/*****************************************************************************************************************/
/* Port GPIO interrupt initialization via Driverlib
/*****************************************************************************************************************/
void gpioHandler_Init(){

        MAP_WDT_A_holdTimer();                                                          //Halting the Watchdog

        MAP_GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P5, GPIO_PIN4 | GPIO_PIN5);  // 5.4 & 5.5 pull up inputs
        MAP_GPIO_clearInterruptFlag(GPIO_PORT_P5, GPIO_PIN4 | GPIO_PIN5);               // Clear interrupt flag
        MAP_GPIO_enableInterrupt(GPIO_PORT_P5, GPIO_PIN4 | GPIO_PIN5);                  // Enable the interrupt on the pins
        MAP_Interrupt_enableInterrupt(INT_PORT5);                                       // Additional interrupt enable

        MAP_SysCtl_enableSRAMBankRetention(SYSCTL_SRAM_BANK1);                          // Enabling SRAM Bank Retention
        MAP_Interrupt_enableMaster();                                                   // Enabling MASTER interrupts

}


/*****************************************************************************************************************/
/* LED initialization function (outputs).
/*****************************************************************************************************************/
void LED_init(){

        //Blue LED Pin
        P3->SEL0 &=~ BIT6;                      // Select the pin as a GPIO
        P3->SEL1 &=~ BIT6;                      // Select the pin as a GPIO
        P3->DIR |= BIT6;                         // Set pin as output
        P3->OUT &=~ BIT6;                       // Assign the initial condition as "off"
}
```

```
/*****************************************************************************************************************/
/* Button initialization function (inputs).
/*****************************************************************************************************************/
void button_init(){

          // BTN 1 input
          P5->SEL0 &=~ BIT4;                    // Set to GPIO
          P5->SEL1 &=~ BIT4;                    // Clear pin
          P5->DIR &=~ BIT4;                     // set as input
          P5->REN |= BIT4;                        // Set to 1 enables internal resistor
          P5->OUT |= BIT4;                      // 1 Assigned as a Pull-Up resistor (the pull-Up resistor)

          // BTN 2 input
          P5->SEL0 &=~ BIT5;                        // set to GPIO
          P5->SEL1 &=~ BIT5;                    // Clear pin
          P5->DIR &=~ BIT5;                     // set as input
          P5->REN |= BIT5;                      // Set to 1 enables internal resistor (the pull-Up resistor)
          P5->OUT |= BIT5;                           // 1 Assigned as a Pull-Up resistor
}


/*****************************************************************************************************************/
/*                           --------------------END OF PROGRAM--------------------
/*****************************************************************************************************************/
```

PART 2 CODE

```
/*****************************************************************************************************************************************
 *  Author:          Corey Moura
 *  Lab:             2.2  -  "Sequencing colors of a RGB LED using a timer interrupt and pushbutton"
 *  Date:            9/4/19
 *  Instructor:      Dr. Kandalaft
 *
 *  Description:     Use the three on-board RGB LEDs for this exercise (with one external pushbutton)The pushbutton should be
 *                   triggered by interrupts. On RESET, the LEDs should be OFF. When you press and hold down the pushbutton the
 *                   red LED (and only the red LED) lights for one second followed by only the green LED ON for one second and
then
 *                   the blue LED is ON for one second.   The timing should be controlled by the systick timer interrupt set at 1
 *                   second intervals.This sequence should repeat as long as the pushbutton is depressed. When the pushbutton is
 *                   released the sequencing pauses with the current LED remaining ON. Upon the next press of the pushbutton, the
 *                   LED sequence will start up again in reverse order and continue for as long as the pushbutton is held down.
 *
 *  Notes:           DriverLib used to initialize the interrupts
 *
 *
 *****************************************************************************************************************************************/
#include "driverlib.h"                          // Driverlib library
#include <stdint.h>                             // A set of typedefs that specify exact-width integer types
#include <stdbool.h>                            // Allow boolean variable to be used
#include <stdio.h>                              // Standard input output for printf etc.
#include <stdlib.h>                             // Standard C-programming library
#include <string.h>                             // Allows the use of strings
#include "msp.h"                                // Another TI library of some sort


void initializeAll();                           // Called to initialize all of the MSP features used in the program.
void SysTick_Init_interupt();                   // Called from initialize all, initialization of the systic timer
void gpioHandler_Init();                        // Called from initialize all, initialization of the GPIO buttons as interrupts
void LED_init();                                // Called from initialize all, used to initialize the LED pins
void button_init();                           // Called from initialize all, used to initialize the button pins


void red_On();                                  // Used to turn on the specified LED
void red_Off();                                 // Used to turn off the specified LED

void green_On();                                // Used to turn on the specified LED
void green_Off();                               // Used to turn off the specified LED

void blue_On();                                 // Used to turn on the specified LED
void blue_Off();                                // Used to turn off the specified LED

void state_1();                                 // Used in the state machine to complete processes of this state
void state_2();                                 // Used in the state machine to complete processes of this state
void state_3();                                 // Used in the state machine to complete processes of this state
void state_4();                                 // Used in the state machine to complete processes of this state

void stateControl();                            // Handles state assignments

void buttonCheck();                               // Uses the debounce to confirm a button is pressed
uint8_t DebounceSwitch_1();                     // Debounce switch 1

void PORT5_IRQHandler(void);


volatile uint8_t buttonFlag_1 = 0;                      // Flag set in GPIO handler when the button is pressed
volatile uint8_t btn1Pressed = 0;                       // Assigned 1 when the button has been confirmed to be pressed

volatile uint32_t timeout = 0;                          // Flag set from the syystic interrupt
volatile uint32_t sys_loadVal = 6000000;                // Initial value assigned to the systic timer

volatile uint8_t stateAssigned = 0;                     // Prevents the continuous incrementation of the state
volatile uint32_t state = 1;                           // Assigned the current state value
volatile uint32_t numPressed = 0;                       // Tracks the number of presses of the user

void main(void){

        WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;             // stop watchdog timer
```

```
    initializeAll();                                    // Initialize the pins

    while(1){

        if(buttonFlag_1) buttonCheck();                 // If the interrupt is triggered then check button

        if(btn1Pressed && timeout){                    // Waits here until the systic flag is set

            stateControl();                             // Assigns the state before entering state-machine

            switch (state){
                case(1):
                        state_1();                          // All off
                        break;
                case(2):
                        state_2();                  // Red on
                        break;
                case(3):
                        state_3();                    // Green on
                        break;
                case(4):
                        state_4();                  // Blue on
                        break;
                default: state_1();                 // All off
              }
            timeout = 0;
        }

        if(!buttonFlag_1 && !(DebounceSwitch_1())) btn1Pressed = 0;       // Resets the btnpressed variable if true
    }
}


/*****************************************************************************************************************/
/* Check the debounce of the buttons.  Assigns the variables seen below and resets the interrupt flags.
/*****************************************************************************************************************/
void buttonCheck(){

    if(DebounceSwitch_1()){
    buttonFlag_1 = 0;                               // Reset the interrupt flag
    btn1Pressed = 1;                                  // Set the button variable to the pressed condition
    numPressed++;
    }
    else btn1Pressed = 0;                             // If conditions are not met, then the button is not pressed

    if(numPressed > 2) numPressed = 1;                // Monitor the number of button presses

    return;

}

/*****************************************************************************************************************/
/* DEBOUNCERS - Sample code provided by Dr. Kandalaft
/*****************************************************************************************************************/
uint8_t DebounceSwitch_1(){

    static uint16_t check1 = 0;                                 // Variable reset every time function is called

    check1 = (check1<<1) | ((P5IN & BIT4)>>1) | 0xfc00;         // Bitwise OR with shifts to complete in 1 cycle

    _delay_cycles(15000);                                       // Delay the second check to confirm debounce

    if(check1 == 0xfc00){
    return 1;                                                   // If it is the correct value then button is pressed
    }
    return 0;                                                   // If it is not the correct value then return 0

}


/*****************************************************************************************************************/
/* Depending on the number of times the button is pressed will determine the state.
/*****************************************************************************************************************/
void stateControl(){

    if(numPressed == 1){
    state++;                                                // Increment the state
```

```
        if(state > 4)  state = 2;                               // Cycle the state
        }

        if(numPressed == 2){
        state--;                                                // Decrement the state
        if(state < 2) state = 4;                               // Cycle the state
        }

        return;

}


/*************************************************************************************************************************/
/* Functions of the different states of the state machine
/*************************************************************************************************************************/
void state_1(){  red_Off(); green_Off();   blue_Off();  }              // All LEDs off
void state_2(){  red_On();  green_Off();   blue_Off();  }              // Red LED on
void state_3(){  red_Off(); green_On(); blue_Off();  }                 // Green LED on
void state_4(){  red_Off(); green_Off();   blue_On();   }              // Blue LED on


/*************************************************************************************************************************/
/* Systic Interrupt Handler
/*************************************************************************************************************************/
void Systick_Handler(void){
        timeout = 1 ;                                           // set flag for timeout of SysTick, rest in main
        //intervalCnt++ ;                                       // increment interval timer by 1ms, reset in main ( )

}


/*************************************************************************************************************************/
/* GPIO ISR handler for the buttons.
/*************************************************************************************************************************/
void PORT5_IRQHandler(void){
        uint32_t status;                                        // Declare variable of flag of the interrupt

        status = MAP_GPIO_getEnabledInterruptStatus(GPIO_PORT_P5);  // Assign status a value if the interrupt was called
        MAP_GPIO_clearInterruptFlag(GPIO_PORT_P5, status);          // After assignment above, clear the interrupt flag

        if(status & GPIO_PIN4){
        buttonFlag_1 = 1;                                  // If the first button was pressed, set the flag
        }

}


/*************************************************************************************************************************/
/* LED outputs from the MSP432.
/*************************************************************************************************************************/
void red_On()   {    P2->OUT |= BIT0;    }                          // Turn on the LED
void red_Off()  {    P2->OUT &=~ BIT0;   }                          // Turn off the LED

void green_On(){     P2->OUT |= BIT1;    }                          // Turn on the LED
void green_Off(){    P2->OUT &=~ BIT1;   }                          // Turn off the LED

void blue_On() {     P2->OUT |= BIT2;    }                          // Turn on the LED
void blue_Off() {    P2->OUT &=~ BIT2;   }                          // Turn off the LED


/*************************************************************************************************************************/
/* Call out to the functions and initialize the pins and timers used in the program.
/*************************************************************************************************************************/
void initializeAll(){

        SysTick_Init_interupt();                                // Systic initialization
        gpioHandler_Init();                                     // GPIO interrupt initialization
        LED_init();                                             // LED initialization (outputs)
        button_init();                                          // Button initializations (inputs)

}


/*************************************************************************************************************************/
/* Systick timer initialization with interrupts.
/*************************************************************************************************************************/
void SysTick_Init_interupt(){
```

```
        SysTick->CTRL = 0;            // Disable SysTick during step
        SysTick->LOAD = sys_loadVal;  // Max reload value 0xFFFFFFFF but: for 1ms use 3000 - flag set every millisecond
        SysTick->VAL = 0;             // Any write to current clears it
        SysTick->CTRL = 0x00000007;   // Enable SysTick, 0x00000007 interrupts & 0x00000005 no interrupts

}


/*****************************************************************************************************************/
/* Port GPIO interrupt initialization via Driverlib
/*****************************************************************************************************************/
void gpioHandler_Init(){

        MAP_WDT_A_holdTimer();                                          //Halting the Watchdog

        MAP_GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P5, GPIO_PIN4);    // 5.4 pull up inputs
        MAP_GPIO_clearInterruptFlag(GPIO_PORT_P5, GPIO_PIN4);                 // Clear interrupt flag
        MAP_GPIO_enableInterrupt(GPIO_PORT_P5, GPIO_PIN4);                    // Enable the interrupt on the pins
        MAP_Interrupt_enableInterrupt(INT_PORT5);                             // Additional interrupt enable

        MAP_SysCtl_enableSRAMBankRetention(SYSCTL_SRAM_BANK1);                // Enabling SRAM Bank Retention
        MAP_Interrupt_enableMaster();                                         // Enabling MASTER interrupts

}


/*****************************************************************************************************************/
/* LED initialization function (outputs).
/*****************************************************************************************************************/
void LED_init(){

        //Red LED Pin
        P2->SEL0 &=~ BIT0;                                  // Select the pin as a GPIO
        P2->SEL1 &=~ BIT0;                                  // Select the pin as a GPIO
        P2->DIR |= BIT0;                                      // Set pin as output
        P2->OUT &=~ BIT0;                                 // Assign the initial condition as "off"

        //Green LED Pin
        P2->SEL0 &=~ BIT1;                                  // Select the pin as a GPIO
        P2->SEL1 &=~ BIT1;                                  // Select the pin as a GPIO
        P2->DIR |= BIT1;                                  //set as output
        P2->OUT &=~ BIT1;                                 // Assign the initial condition as "off"

        //Blue LED Pin
        P2->SEL0 &=~ BIT2;                                  // Select the pin as a GPIO
        P2->SEL1 &=~ BIT2;                                  // Select the pin as a GPIO
        P2->DIR |= BIT2;                                  //set as output
        P2->OUT &=~ BIT2;                                 // Assign the initial condition as "off"

}


/*****************************************************************************************************************/
/* Button initialization function (inputs).
/*****************************************************************************************************************/
void button_init(){

        // BTN 1 input
        P5->SEL0 &=~ BIT4;                                  // Set to GPIO
        P5->SEL1 &=~ BIT4;                                  // Clear pin
        P5->DIR &=~ BIT4;                                 // set as input
        P5->REN |= BIT4;                                  // Set to 1 enables internal resistor
        P5->OUT |= BIT4;                                  // 1 Assigned as a Pull-Up resistor (the pull-Up resistor)

}



/*****************************************************************************************************************/
                                    ----------------------END OF PROGRAM----------------------
/*****************************************************************************************************************/
```

PART 3 CODE

```
/*****************************************************************************************************************************
 *  Author:         Corey Moura & Xue Hue
 *  Lab:            2.3  -  "Controlling the LED lighting sequence using a timer interrupt and two pushbutton interrupts"
 *  Date:           9/4/19
 *  Instructor:     Dr. Kandalaft
 *
 *  Description:    In this part, you will use a second pushbutton in combination with the first to flash the selected LED on and
 *                  off in 0.5 second intervals.  Both the Systick and each pushbutton should be triggered using interrupts
 *                  (3 total).  Modify your code from Part II so that the first pushbutton will select the LED color.  The second
 *                  pushbutton is then used to turn that color LED on and off. Both pushbutton functions should be triggered by
 *                  interrupts. Select a color LED by sequencing through the colors with the first pushbutton. When the second
 *                  pushbutton is pressed and held, the selected color LED flashes on and off every 0.5 s. When released, the LED
 *                  stays lit with the last selected color.
 *
 *  Notes:          DriverLib used to initialize the interrupts
 *
 *
 *****************************************************************************************************************************/
#include "driverlib.h"                          // Driverlib library
#include <stdint.h>                               // A set of typedefs that specify exact-width integer types
#include <stdbool.h>                              // Allow boolean variable to be used
#include <stdio.h>                              // Standard input output for printf etc.
#include <stdlib.h>                               // Standard C-programming library
#include <string.h>                               // Allows the use of strings
#include "msp.h"                                // Another TI library of some sort


void initializeAll();                           // Called to initialize all of the MSP features used in the program.
void SysTick_Init_interupt();                   // Called from initialize all, initialization of the systic timer
void gpioHandler_Init();                        // Called from initialize all, initialization of the GPIO buttons as interrupts
void LED_init();                                      // Called from initialize all, used to initialize the LED pins
void button_init();                             // Called from initialize all, used to initialize the button pins

void red_On();                                  // Used to turn on the specified LED
void red_Off();                                 // Used to turn off the specified LED
void red_Toggle();                              // Used to toggle the specified LED

void green_On();                                // Used to turn on the specified LED
void green_Off();                               // Used to turn off the specified LED
void green_Toggle();                            // Used to toggle the specified LED

void blue_On();                                 // Used to turn on the specified LED
void blue_Off();                                // Used to turn off the specified LED
void blue_Toggle();                             // Used to toggle the specified LED

void state_1();                                 // Used in the state machine to complete processes of this state
void state_2();                                 // Used in the state machine to complete processes of this state
void state_3();                                 // Used in the state machine to complete processes of this state
void state_4();                                 // Used in the state machine to complete processes of this state

void stateControl();                                    // Handles state assignments

void toggleControl();                           // Toggles an LED based on the current state

void checkButton1();                            // Handles the logic of checking the buttons
void checkButton2();                            // Handles the logic of checking the buttons

uint8_t DebounceSwitch_1();                     // Debounce switch 1
uint8_t DebounceSwitch_2();                     // Debounce switch 2

void PORT5_IRQHandler(void);                    // ISR handling the button press interrupt


volatile uint8_t buttonFlag_1 = 0;                      // Flag set in GPIO handler when the button is pressed
volatile uint8_t buttonFlag_2 = 0;                    // Flag set in GPIO handler when the button is pressed

volatile uint8_t btn1Pressed = 0;                       // Assigned 1 when the button has been confirmed to be pressed
volatile uint8_t btn2Pressed = 0;                     // Assigned 1 when the button has been confirmed to be pressed
```

```
volatile uint32_t timeout = 0;                          // Flag set from the syystic interrupt
volatile uint32_t sys_loadVal = 1200000;                // Initial value assigned to the systic timer

volatile uint32_t state = 1;                            // Assigned the current state value


void main(void){

        WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;             // stop watchdog timer

        initializeAll();                                        // Initialize the pins

        while(1){

        if(buttonFlag_1) checkButton1();                        // If the interrupt is triggered then check button
        if(buttonFlag_2) checkButton2();                        // If the interrupt is triggered then check button

        if(btn2Pressed){
        if(timeout)  toggleControl();                   // If btn2 is pressed and timeout toggle the Current LED
        else         checkButton2();                    // Reassign the variable btn2Pressed if applicable
        }


        if(btn1Pressed){
        btn1Pressed = 0;                                // Reset the butnPressed variable
        stateControl();                                 // Determine the state before entering state-machine

        _delay_cycles(15000);                             // Delay to aid in button flopping

        switch (state){
                case(1):
                        state_1();                              // All off
                        break;
                case(2):
                        state_2();                              // Red on
                        break;
                case(3):
                        state_3();                              // Green on
                        break;
                case(4):
                        state_4();                              // Blue on
                        break;
                default: state_1();                             // All off
                }

        }
        }
}


/*****************************************************************************************************************/
/* Check the debounce of the buttons.  Assigns the variables seen below and resets the interrupt flags.
/*****************************************************************************************************************/
void checkButton1(){

        if(DebounceSwitch_1()){
        buttonFlag_1 = 0;                                       // Reset the interrupt flag
        btn1Pressed = 1;                                        // Set the button variable to the pressed condition
        }

        return;

}

/*****************************************************************************************************************/
/* Check the debounce of the buttons.  Assigns the variables seen below and resets the interrupt flags.
/*****************************************************************************************************************/
void checkButton2(){

        if(DebounceSwitch_2()){
        buttonFlag_2 = 0;                                       // Reset the interrupt flag
        btn2Pressed = 1;                                        // Set the button variable to the pressed condition
        }
        else btn2Pressed = 0;                                   // If cond not met reset the button pressed var

        return;
```

```
}


/************************************************************************************************************************/
/* DEBOUNCERS - Sample code provided by Dr. Kandalaft
/************************************************************************************************************************/
uint8_t DebounceSwitch_1(){

          static uint16_t check1 = 0;                                   // Variable reset every time function is called

          check1 = (check1<<1) | ((P5IN & BIT4)>>1) | 0xfc00;           // Bitwise OR with shifts to complete in 1 cycle

          _delay_cycles(30000);                                         // Delay the second check to confirm debounce

          if(check1 == 0xfc00){
          return 1;                                                 // If it is the correct value then button is pressed
          }
          return 0;                                                     // If it is not the correct value then return 0

}


/************************************************************************************************************************/
/* DEBOUNCERS - Sample code provided by Dr. Kandalaft
/************************************************************************************************************************/
uint8_t DebounceSwitch_2(){

          static uint16_t check1 = 0;                                   // Variable reset every time function is called

          check1 = (check1<<1) | ((P5IN & BIT5)>>1) | 0xfc00;           // Bitwise OR with shifts to complete in 1 cycle

          _delay_cycles(15000);                                         // Delay the second check to confirm debounce

          if(check1 == 0xfc00){
          return 1;                                                     // If it is the correct value then button is pressed
          }
          return 0;                                                     // If it is not the correct value then return 0

}


/************************************************************************************************************************/
/* State incremented when btn1pressed is true
/************************************************************************************************************************/

void stateControl(){
          state++;                                                      // Increment the state
          if(state > 4)  state = 2;                                     // Cycle the state

          return;
}


/************************************************************************************************************************/
/* Toggles the current LED based on the state, and if btn2 is being held down
/************************************************************************************************************************/

void toggleControl(){

          if(state == 2 && btn2Pressed){
          red_Toggle();                                                 // Toggle the LED
          timeout = 0;                                                  // Reset Systic timer flag
          }
          else if(state == 3 && btn2Pressed){
          green_Toggle();                                               // Toggle the LED
          timeout = 0;                                                  // Reset Systic timer flag
          }
          else if(state == 4 && btn2Pressed){
          blue_Toggle();                                                // Toggle the LED
          timeout = 0;                                                  // Reset Systic timer flag
          }
          else return;

}



/************************************************************************************************************************/
```

```
/* Functions of the different states of the state machine
/*****************************************************************************************************************/
void state_1(){  red_Off(); green_Off();   blue_Off();  }
void state_2(){  red_On();  green_Off();   blue_Off();  }
void state_3(){  red_Off(); green_On(); blue_Off();  }
void state_4(){  red_Off(); green_Off();   blue_On();   }


/*****************************************************************************************************************/
/* Systic Interrupt Handler
/*****************************************************************************************************************/
// Systick Handler used with the interrupt
void Systick_Handler(void){
          timeout = 1 ;                                          // set flag for timeout of SysTick, rest in main
          //intervalCnt++ ;                                     // increment interval timer by 1ms, reset in main ( )

}


/*****************************************************************************************************************/
/* GPIO ISR handler for the buttons.
/*****************************************************************************************************************/
void PORT5_IRQHandler(void){
          uint32_t status;                                       // Declare variable of flag of the interrupt

          status = MAP_GPIO_getEnabledInterruptStatus(GPIO_PORT_P5);  // Assign status a value if the interrupt was called
          MAP_GPIO_clearInterruptFlag(GPIO_PORT_P5, status);          // After assignment above, clear the interrupt flag

          if(status & GPIO_PIN4){
          buttonFlag_1 = 1;                                  // If the first button was pressed, set the flag
          }

          if(status & GPIO_PIN5){
          buttonFlag_2 = 1;                                 // If the second button was pressed, set the flag
          }

}


/*****************************************************************************************************************/
/* LED outputs from the MSP432.
/*****************************************************************************************************************/
void red_On()   {    P2->OUT |= BIT0;    }                         // Turn on the LED
void red_Off()  {    P2->OUT &=~ BIT0;   }                         // Turn off the LED
void red_Toggle(){   P2->OUT ^= BIT0;    }                         // Toggle the LED

void green_On(){     P2->OUT |= BIT1;    }                         // Turn on the LED
void green_Off(){    P2->OUT &=~ BIT1;   }                        // Turn off the LED
void green_Toggle(){ P2->OUT ^= BIT1;    }                         // Toggle the LED

void blue_On() {     P2->OUT |= BIT2;    }                         // Turn on the LED
void blue_Off() {    P2->OUT &=~ BIT2;   }                         // Turn off the LED
void blue_Toggle(){ P2->OUT ^= BIT2;     }                         // Toggle the LED//GPIO ISR handler for the buttons



/*****************************************************************************************************************/
/* Call out to the functions and initialize the pins and timers used in the program.
/*****************************************************************************************************************/
void initializeAll(){

          SysTick_Init_interupt();                               // Systic initialization
          gpioHandler_Init();                                    // GPIO interrupt initialization
          LED_init();                                            // LED initialization (outputs)
    button_init();                                               // Button initializations (inputs)

}


/*****************************************************************************************************************/
/* Systick timer initialization with interrupts.
/*****************************************************************************************************************/
void SysTick_Init_interupt(){
          SysTick->CTRL = 0;            // Disable SysTick during step
          SysTick->LOAD = sys_loadVal;  // Max reload value 0xFFFFFFFF but: for 1ms use 3000 - flag set every millisecond
          SysTick->VAL = 0;             // Any write to current clears it
          SysTick->CTRL = 0x00000007;   // Enable SysTick, 0x00000007 interrupts & 0x00000005 no interrupts
}
```

```
/*****************************************************************************************************************/
/* Port GPIO interrupt initialization via Driverlib
/*****************************************************************************************************************/
void gpioHandler_Init(){

        MAP_WDT_A_holdTimer();                                                          //Halting the Watchdog

        MAP_GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P5, GPIO_PIN4 | GPIO_PIN5);  // 5.4 & 5.5 pull up inputs
        MAP_GPIO_clearInterruptFlag(GPIO_PORT_P5, GPIO_PIN4 | GPIO_PIN5);               // Clear interrupt flag
        MAP_GPIO_enableInterrupt(GPIO_PORT_P5, GPIO_PIN4 | GPIO_PIN5);                  // Enable interrupt on the pins
        MAP_Interrupt_enableInterrupt(INT_PORT5);                                       // Additional interrupt enable

        MAP_SysCtl_enableSRAMBankRetention(SYSCTL_SRAM_BANK1);                          // Enabling SRAM Bank Retention
        MAP_Interrupt_enableMaster();                                                        // Enabling MASTER interrupts

}


/*****************************************************************************************************************/
/* LED initialization function (outputs).
/*****************************************************************************************************************/
void LED_init(){

        //Red LED Pin
        P2->SEL0 &=~ BIT0;                                  // Select the pin as a GPIO
        P2->SEL1 &=~ BIT0;                                  // Select the pin as a GPIO
        P2->DIR |= BIT0;                                   // Set pin as output
        P2->OUT &=~ BIT0;                                  // Assign the initial condition as "off"

        //Green LED Pin
        P2->SEL0 &=~ BIT1;                                  // Select the pin as a GPIO
        P2->SEL1 &=~ BIT1;                                  // Select the pin as a GPIO
        P2->DIR |= BIT1;                                    //set as output
        P2->OUT &=~ BIT1;                                  // Assign the initial condition as "off"

        //Blue LED Pin
        P2->SEL0 &=~ BIT2;                                  // Select the pin as a GPIO
        P2->SEL1 &=~ BIT2;                                  // Select the pin as a GPIO
        P2->DIR |= BIT2;                                   //set as output
        P2->OUT &=~ BIT2;                                  // Assign the initial condition as "off"

}


/*****************************************************************************************************************/
/* Button initialization function (inputs).
/*****************************************************************************************************************/
void button_init(){

        // BTN 1 input
        P5->SEL0 &=~ BIT4;                                  // Set to GPIO
        P5->SEL1 &=~ BIT4;                                  // Clear pin
        P5->DIR &=~ BIT4;                                  // set as input
        P5->REN |= BIT4;                                    // Set to 1 enables internal resistor
        P5->OUT |= BIT4;                                   // 1 Assigned as a Pull-Up resistor (the pull-Up resistor)

        // BTN 2 input
        P5->SEL0 &=~ BIT5;                                  // Set to GPIO
        P5->SEL1 &=~ BIT5;                                  // Clear pin
        P5->DIR &=~ BIT5;                                  // set as input
        P5->REN |= BIT5;                                   // Set to 1 enables internal resistor
        P5->OUT |= BIT5;                                   // 1 Assigned as a Pull-Up resistor (the pull-Up resistor)
}


/*****************************************************************************************************************/
                                        --------------------END OF PROGRAM--------------------
/*****************************************************************************************************************/
```