EGR 326 - 903

Lab 1
Input Interfacing with MSP432 using Interrupts

Corey Moura
Xue Hua

Date performed : 30 August 2019
Date submitted : 06 September 2019

Professor Kandalaft

Objective

The objective of this lab was to use two pushbuttons to control sequence of an RGB LED with the MSP 432. To develop a program that uses SysTick timer to generate precise time intervals for controlling duration of LEDs illumination. Furthermore, to accomodate switch bounce into the program.

Equipment

- MSP432 P401R Microcontroller
- RGB LED
- two momentary push buttons
- two 100 ohm resistor
- 220 ohm resistor

Results

Part 1: Sequencing RGB using a single momentary push button

A state machine was developed to map out the button as the input and the three different colors as the states, to include the “off” state. The state machine can be seen in the figure below.

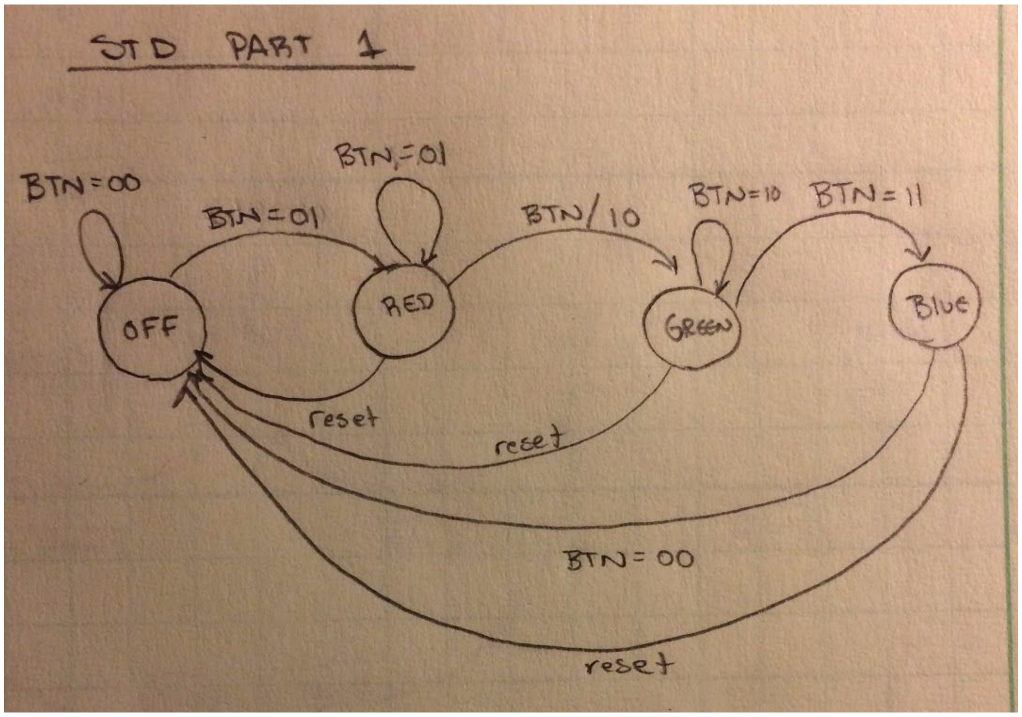


Figure 1. State Transition Diagram for Part 1

The program skeleton was created in order to accommodate for the logic of the program. The skeleton included all of the necessary programming items needed to communicate with the MSP432, things like: libraries, main(), pin declarations, while(1), etc.

Three pins were assigned as outputs, and one as an input. The input pin was assigned with a pull up resistor though the SEL bits. This means the pin will see logic 3.3V until the button is pressed and the pin is grounded. This is the safest way to prevent damage to the board.

The circuit was created next using the 2N7000 MOSFETs. A single white LED was wired first to test out the circuit and the initial program. The picture can be seen below. A small program with simple logic was used to confirm the wiring of the LED.

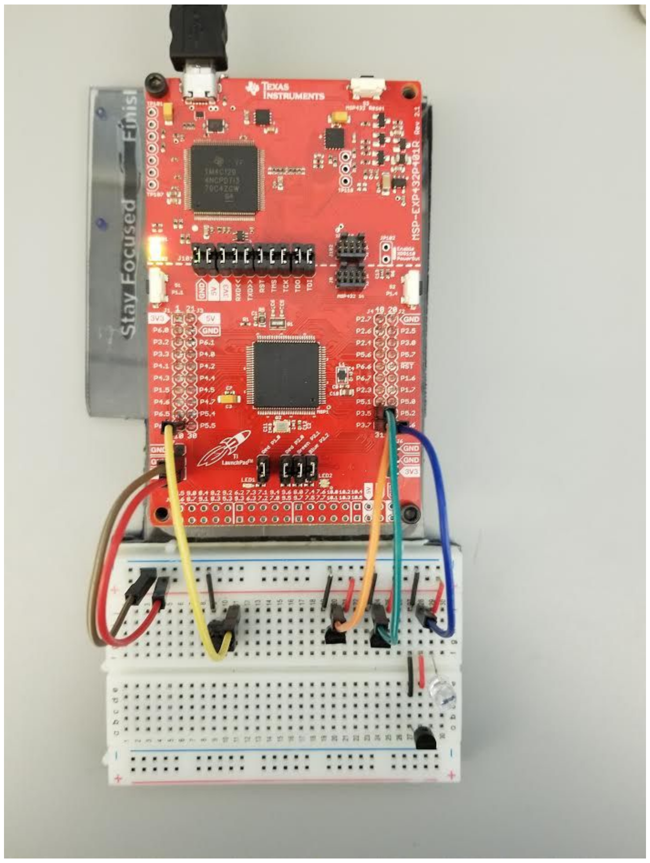


Figure 2. Establishing the circuit with white LED

The program uses a switch case statement to handle the state machine. The program is kept minimalistic via function call outs. There are many functions in this program despite the programs overall complexity. This is done for readability, debugging, and reusability.

The program was then confirmed to perform according to the guidelines specified by the lab instructions. When the button was pressed the LEDs would increment to the next color.

Part 2: Sequencing colors using two push buttons in reverse direction

The code from part 1 was used and expanded to accommodate for the new requirements of part 2. An additional input, button, and switch debounce was added to the circuit and program. Since the state machine from part 1 was robust enough, no additional logic was required to allow it to operate correctly.

New logic was written to handle the second button and the increment and decrement of the states, which depends on the button pressed. This is controlled in the “stateControl()” function. It will assign the state, and then the program will move into the switch case and assign the outputs. The difficult part was handling both buttons to be pressed at the same time.

The “buttonCheck()” function handles the checking of the buttons. Logically it was written to prevent only one of the buttons from being checked. If one is checked, then the other is checked. The response of both buttons pressed was extremely poor before this logic was re-worked. It was seen that the program would just happen to land on the second button press, skip the first button, and enter the state-machine. This poor response could also be mitigated by incorporating interrupts, which would be the next improvement to be made. Once the program was seen operating correctly with the three LED hooked up in the circuit, they were removed and the RGB was installed.

The RGB circuit required a different circuit all together. The common cathode characteristic of the RGB LED was investigated and the LED was installed using the resistor values calculated in the prelab. Using the 5V output of the MSP432, and the pin output voltages, the circuit was created and tested. It was observed that the output currents would not allow the voltage to pass the forward bias of the LEDs. Once this was discovered, the circuit was remade, to exclude the MOSFETs. The working circuit can be seen below.

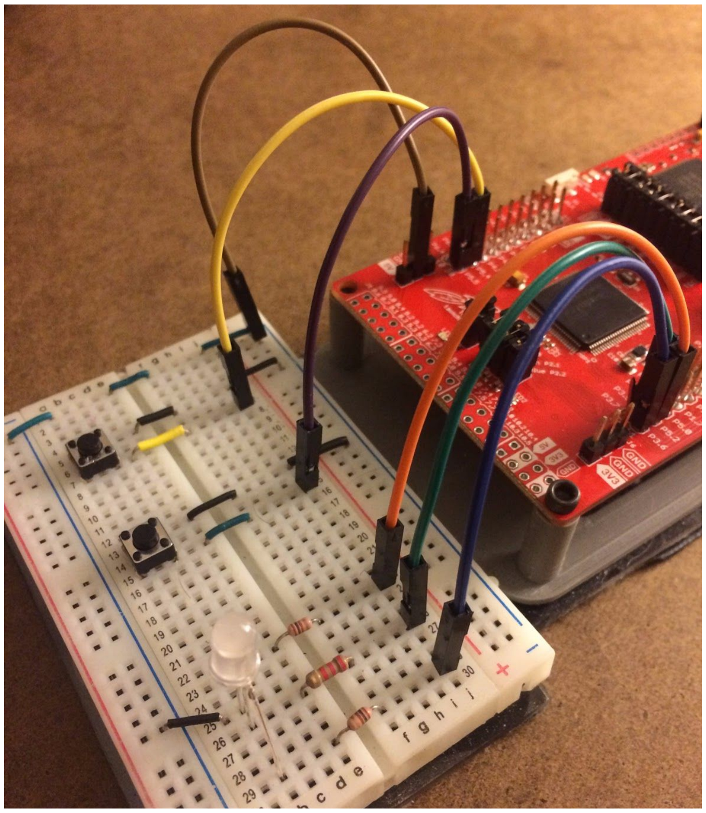


Figure 3. Establishing the circuit with white LED

Using this circuit, and the program found in the Appendix, part 2 of the lab was tested, and verified. One button will run the sequence as RGB, and the other will run the sequence BGR. Both buttons pressed results in the LED turning off.

Table 1. Pin Table

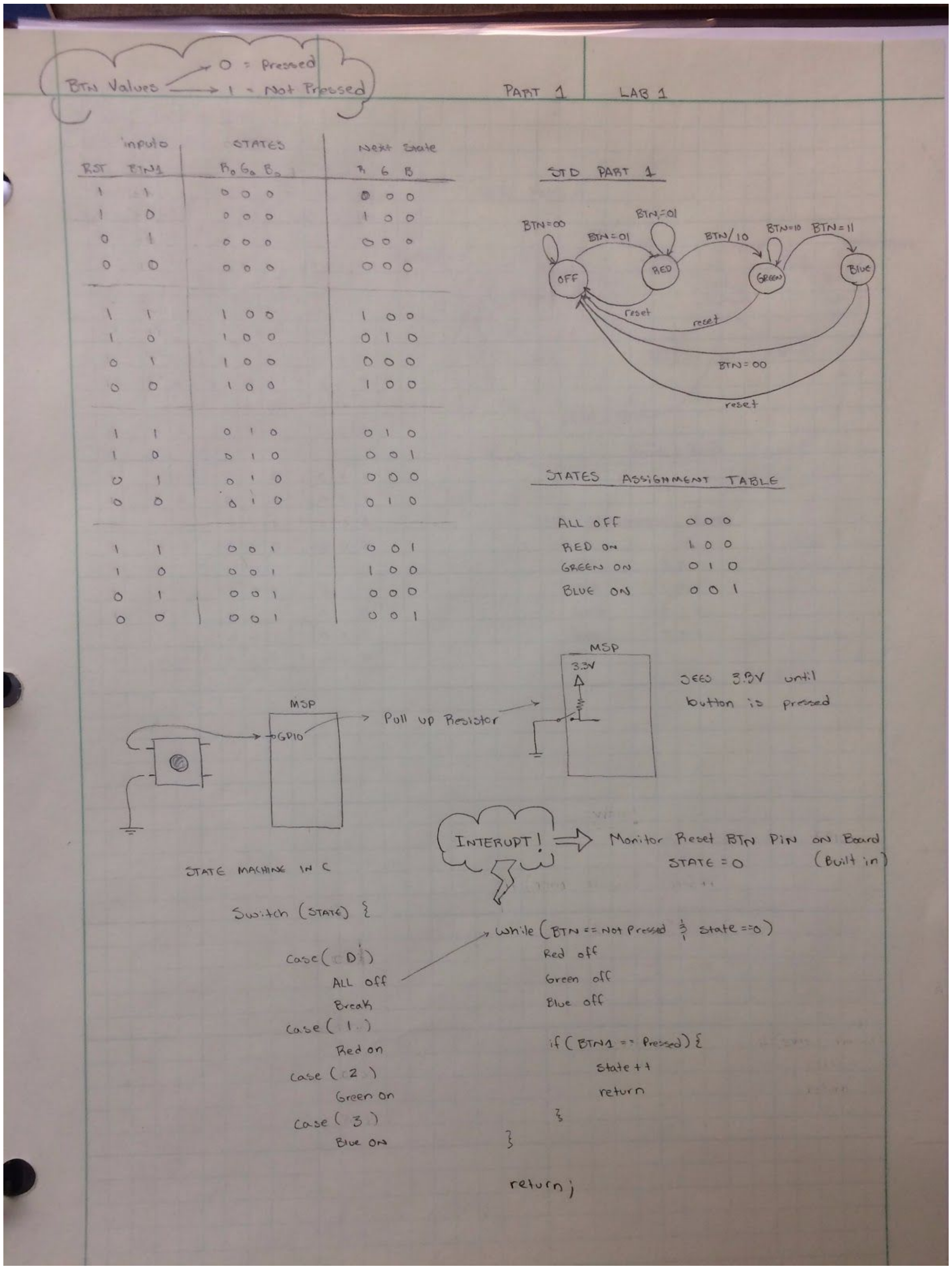
PINS	**Component**
3.4	Red
3.5	Green
3.6	Blue
6.4	Pushbutton 1
5.5	Pushbutton 2

Conclusion

The features of CCS and the MSP432 were refreshed, and the lab was completed successfully. Two push buttons were integrated with the RGB LED and MSP432 using a program with a state machine and software debouncers to handle the physical properties of metallic contact switches.Additional time was taken to get back up to speed from where EGR226 had left off. When interrupts were added to the program, it was realized that *MSP432-Ware* had not been installed during the initial installation of CCS. Without *MSP432-Ware* installed, *IT Resource Explorer* could not be used, and therefore, a driverLib project could not be opened. Since the finish of this lab, all supporting software for CCS has been installed in preparation for the next labs.

Appendix

Figure A1: Part 1 additional work



BTN Values → 0 = Pressed, 1 = Not Pressed

PART 1 LAB 1

inputs		STATES	Next State
RST	BTN1	$R_0 G_0 B_0$	R G B
1	1	0 0 0	0 0 0
1	0	0 0 0	1 0 0
0	1	0 0 0	0 0 0
0	0	0 0 0	0 0 0
1	1	1 0 0	1 0 0
1	0	1 0 0	0 1 0
0	1	1 0 0	0 0 0
0	0	1 0 0	1 0 0
1	1	0 1 0	0 1 0
1	0	0 1 0	0 0 1
0	1	0 1 0	0 0 0
0	0	0 1 0	0 1 0
1	1	0 0 1	0 0 1
1	0	0 0 1	1 0 0
0	1	0 0 1	0 0 0
0	0	0 0 1	0 0 1

STD PART 1

BTN=00
BTN=01
BTN=01
BTN/10
BTN=10
BTN=11
OFF
RED
GREEN
Blue
reset
reset
BTN=00
reset

STATES ASSIGNMENT TABLE

ALL OFF	0 0 0
RED ON	1 0 0
GREEN ON	0 1 0
BLUE ON	0 0 1

MSP
GPIO
Pull up Resistor
MSP
3.3V

Sees 3.3V until button is pressed

INTERUPT! ⟹ Monitor Reset BTN Pin on Board (Built in)
STATE = 0

STATE MACHINE IN C

```
Switch (STATE) {
    case( 0 )
        ALL off
        Break
    case( 1 )
        Red on
    case ( 2 )
        Green on
    case ( 3 )
        Blue ON
```

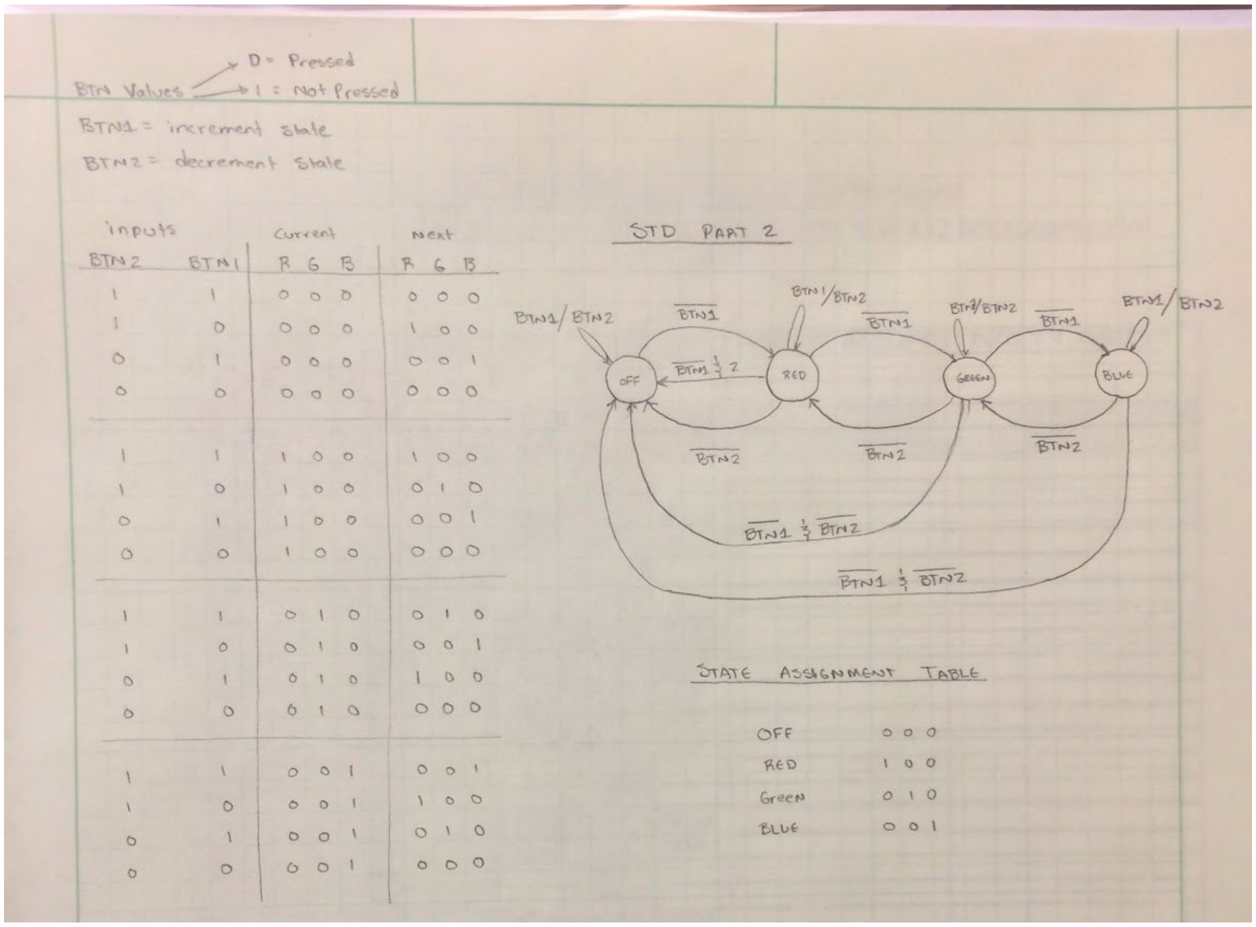
```
while (BTN == Not Pressed & state ==0)
    Red off
    Green off
    Blue off

    if (BTN1 == Pressed) {
        State ++
        return
    }
}

return;
```

Figure A1: Part 1 additional work

Figure A2: Part 2 additional work



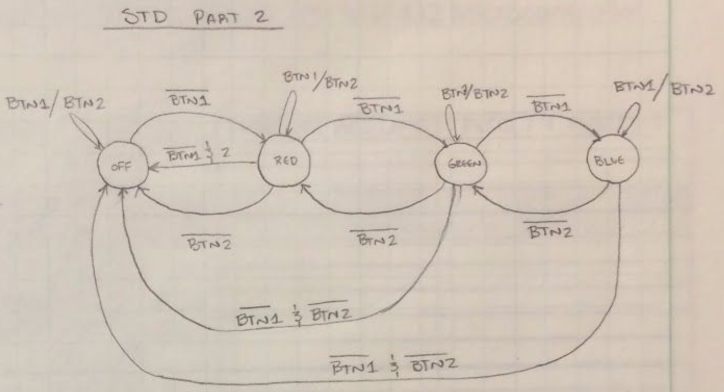
BTN Values → 0 = Pressed
→ 1 = Not Pressed

BTN1 = increment state

BTN2 = decrement state

inputs		Current			Next		
BTN2	BTN1	R	G	B	R	G	B
1	1	0	0	0	0	0	0
1	0	0	0	0	1	0	0
0	1	0	0	0	0	0	1
0	0	0	0	0	0	0	0
1	1	1	0	0	1	0	0
1	0	1	0	0	0	1	0
0	1	1	0	0	0	0	1
0	0	1	0	0	0	0	0
1	1	0	1	0	0	1	0
1	0	0	1	0	0	0	1
0	1	0	1	0	1	0	0
0	0	0	1	0	0	0	0
1	1	0	0	1	0	0	1
1	0	0	0	1	1	0	0
0	1	0	0	1	0	1	0
0	0	0	0	1	0	0	0

STD PART 2



STATE ASSIGNMENT TABLE

OFF	0 0 0
RED	1 0 0
Green	0 1 0
BLUE	0 0 1

Figure A2: Part 2 additional work

Part 2 Verified Program:

```
/**********************************************************************************************************************
 *  Author:     Corey Moura and Xue Hua
 *  Lab:        1 input interfacing with the MSP432
 *  Date:       9/3/19
 *  Instructor: Dr. Kandalaft
 *
 *  Description:    Program uses a state machine to increment the color of an RGB LED through a sequence of colors
 *                  based on the inputs of the button to the MSP432.  When the user presses button 1 the state
 *                  increments front to back in
 *                  the sequence RGB.  If button 2 is pressed then the state decrements back to front in the sequence
 *                 BGR. If both are pressed at the same time, the LED will turn off.
 *
 *  Notes:         The instance of both buttons simultaneously pressed is difficult to handle.
 **********************************************************************************************************************/

/* Libraries */
#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>          //Used for debugging
#include <stdlib.h>     //Used for debugging
#include <string.h>     //Used for debugging
#include "msp.h"



void initializeAll();                   /* Initialize the pins - includes sets their initial conditions */

void red_On();                          /* Used from the state functions below to easily track what is on and off */
void red_Off();
void green_On();
void green_Off();
void blue_On();
void blue_Off();

void state_1();                         /* States of the program */
void state_2();
void state_3();
void state_4();

void stateControl();                    /* Handles state assignments */

void buttonCheck();                     /* Handles the logic of checking the buttons */

void PORT5_IRQHandler(void);

uint8_t DebounceSwitch_1();             /* Handle button debounce */
uint8_t DebounceSwitch_2();
```

```
/* Global variables */
volatile uint32_t state = 1;
volatile uint8_t btn1Pressed = 0;
volatile uint8_t btn2Pressed = 0;
volatile uint8_t stateAssigned = 0;
volatile uint8_t both = 0;


void main(void){

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

   initializeAll();                               // Initialize the pins

           while(1){

              /* If a button is pressed and the state has not already been assigned in that button press, then enter. */
              if((btn1Pressed || btn2Pressed) && stateAssigned == 0){

                 stateControl();                  // Assign the state

                    /* 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
            }
            stateAssigned = 1;                    // The state has been set for this button press
              }

              /* If the user has one, or both of the buttons pressed this will not allow the program to enter the state
machine. It prevents the program from auto incrementing through the states by setting a flag. */
              if(!(btn1Pressed) && !(btn2Pressed)) { stateAssigned = 0;}
           }
}

/* Check the state of the buttons.  If regardless where the program is when a button is pressed, it will also check the
other.
 * This logic handles the case if the program has passed the first button assignment and has moved onto the next,
even though the user is pressing both buttons.  Debounce is handled here also. */
```

```
void buttonCheck(){
    btn1Pressed = DebounceSwitch_1();
    btn2Pressed = DebounceSwitch_2();

    if(btn1Pressed) btn2Pressed = DebounceSwitch_2();
    if(btn2Pressed) btn1Pressed = DebounceSwitch_1();

    return;
}

/* Depending on which button is pressed, the state will increment or decrement.  In effect determining which LED will
be lit. */
void stateControl(){

    if(btn1Pressed && btn2Pressed){
        state = 1;
        both = 0;
        //printf("Both state = %d\n", state);
    }

    else if(btn1Pressed){
        state++;
        if(state > 4)  state = 2;
        //printf("btn1 pressed state = %d\n", state);
    }

    else if(btn2Pressed){
        state--;
        if(state < 2)  state = 4;
        //printf("btn2 pressed state = %d\n", state);
    }

    return;
}

/* Sample code provided by Dr. Kandalaft.  This function completes bitwise OR logic and a delay to handle the
debounce */
uint8_t DebounceSwitch_1(){
    static uint16_t check1 = 0;
    check1 = (check1<<1) | ((P5IN & BIT4)>>1) | 0xfc00;
    _delay_cycles(150000);
    if(check1 == 0xfc00){
        return 1;
    }
    return 0;
}

/* Sample code provided by Dr. Kandalaft.  This function completes bitwise OR logic and a delay to handle the
debounce */
uint8_t DebounceSwitch_2(){
    static uint16_t check2 = 0;
    check2 = (check2<<1) | ((P5IN & BIT5)>>1) | 0xfc00;
    _delay_cycles(150000);
    if(check2 == 0xfc00){
        return 1;
    }
    return 0;
}

/*************************************************************************************************************/
/* LED outputs from the MSP432 */
void green_On(){    P3->OUT |= BIT5;    }
void blue_On() {    P3->OUT |= BIT6;    }
void red_On()  {    P5->OUT |= BIT2;    }

void green_Off(){   P3->OUT &=~ BIT5;   }
void blue_Off() {   P3->OUT &=~ BIT6;   }
void red_Off()  {   P5->OUT &=~ BIT2;   }

/*************************************************************************************************************/
/* States use the above functions to create easily readable logic */
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();   }


/*************************************************************************************************************/
/* Initialize all of the pins used in the program. */
void initializeAll(){

    //Green LED Pin
    P3SEL0 &=~ BIT5;                //set to GPIO
    P3SEL1 &=~ BIT5;
    P3->DIR |= BIT5;                //set as output
    P3->OUT &=~ BIT5;

    //Blue LED Pin
    P3SEL0 &=~ BIT6;                //set to GPIO
    P3SEL1 &=~ BIT6;
    P3->DIR |= BIT6;                //set as output
    P3->OUT &=~ BIT6;

    //Red LED Pin
    P5SEL0 &=~ BIT2;                //set to GPIO
    P5SEL1 &=~ BIT2;
    P5->DIR |= BIT2;                //set as output
    P5->OUT &=~ BIT2;

    // BTN 1 input
    P5SEL0 &=~ BIT4;                //set to GPIO
    P5SEL1 &=~ BIT4;                 //Clear P3.2
    P5->DIR &=~ BIT4;                //set as input
    P5REN |= BIT4;                  //Set to 1 enables internal resistor
    P5OUT |= BIT4;                  //Assigned as a Pull-Up resistor

    // BTN 2 input
    P5SEL0 &=~ BIT5;                 //set to GPIO
    P5SEL1 &=~ BIT5;                //Clear P3.2
    P5->DIR &=~ BIT5;               //set as input
    P5REN |= BIT5;                  //Set to 1 enables internal resistor
    P5OUT |= BIT5;                  //Assigned as a Pull-Up resistor
  }
  }
```