PROJECT #2 VERIFICATION

PROGRAMMING IN C

COOPER BROTHERTON

INSTRUCTOR: DR. MICHAEL JO

ECE230-03

REQUIREMENTS

- Project implemented in C
- LED2 shall be initially *off* on system start
- While LED2 is *off*, upon press and release of S1, LED2 shall begin blinking at a rate of 1Hz
    - Switch S1 shall be software-debounced on press and release
    - LED2 shall turn *on* within 10ms of the release of S1
    - LED2 shall blink at 50% duty cycle (*on* for 500ms and *off* for 500ms)
- While LED2 is blinking, upon press of S1, LED2 shall turn *off* and stop blinking
    - LED2 shall turn *off* within 1ms of S1 press
    - System shall wait until S1 has been released before returning to a state of detecting a new press and release of S1

ADVANCED REQUIREMENTS

- For all requirements, accuracy shall be within ±10µs
- Initially on system start, the red LED of LED2 shall be the *active* LED whose state is toggled by S1
- While LED2 is blinking, upon press of S2, the *active* LED shall toggle in the following cyclic pattern: red → green → blue → red → etc.
    - The newly *active* LED shall turn *on* within 1ms of the press of S2, but not prior to the former *active* LED turning *off*
    - The *active* LED shall toggle only once per press and release of S2
    - Switch S2 shall be software-debounced on press and release
    - The system shall be in a paused state between the press and release of S2

TEST PLAN

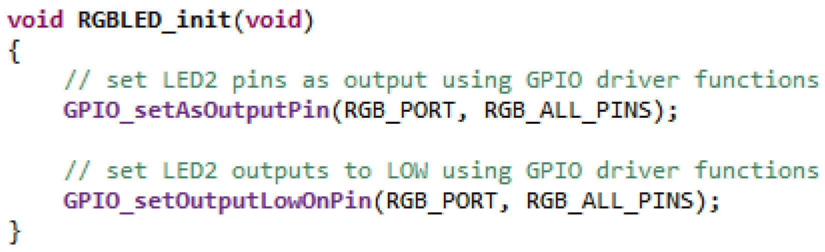
The following details a plan for testing the specifications

Test	Procedure	Pass/Fail Criteria
1	On start, verify LED2 is off	LED2 is off at t=0
2	On start, verify the active LED is red	The red LED activates on the first press and release of S1
3	Verify LED2 turns on within spec	LED2 turns on within 0<t<10.01ms after S1 is released
4	Verify S1 and S2 are debounced on both press and release	S1 and S2 have software debouncing on every press and release
5	Use an oscilloscope to verify LED2 has a frequency of 1Hz and a 50% duty cycle	LED2 toggles every 499.99<t<500.01ms
6	Verify LED2 turns off and stops blinking after S1 is pressed within spec	LED2 turns off and stops blinking within 0<t<1.01ms after the press of S1
7	Verify the system waits until S1 has been released before waiting for a new S1 press	The system waits until S1 has been released before detecting a new S1 press and release
8	Verify S2 toggles which LED is active and toggles in the correct cyclic pattern	S2 toggles the active LED and goes red → green → blue → red
9	Verify the next active LED turns on after S2 is pressed within spec	The former active LED turns off and the next active LED turns on within 0<t<1.01ms after the press of S2
10	Verify the active LED toggles only once per S2 press and release	The active LED changes only once when S2 is pressed and released
11	Verify the system is paused while S2 is being held	The LED does not blink nor react to switch inputs while S2 is being held

VERIFICATION

TEST 1

The following is from `RGBLED_init` in `rgbLED.c`, which is called upon startup:



```
void RGBLED_init(void)
{
    // set LED2 pins as output using GPIO driver functions
    GPIO_setAsOutputPin(RGB_PORT, RGB_ALL_PINS);

    // set LED2 outputs to LOW using GPIO driver functions
    GPIO_setOutputLowOnPin(RGB_PORT, RGB_ALL_PINS);
}
```

The output on the RGB LED port is set to low, so the LED is off on startup.

Meets criteria.

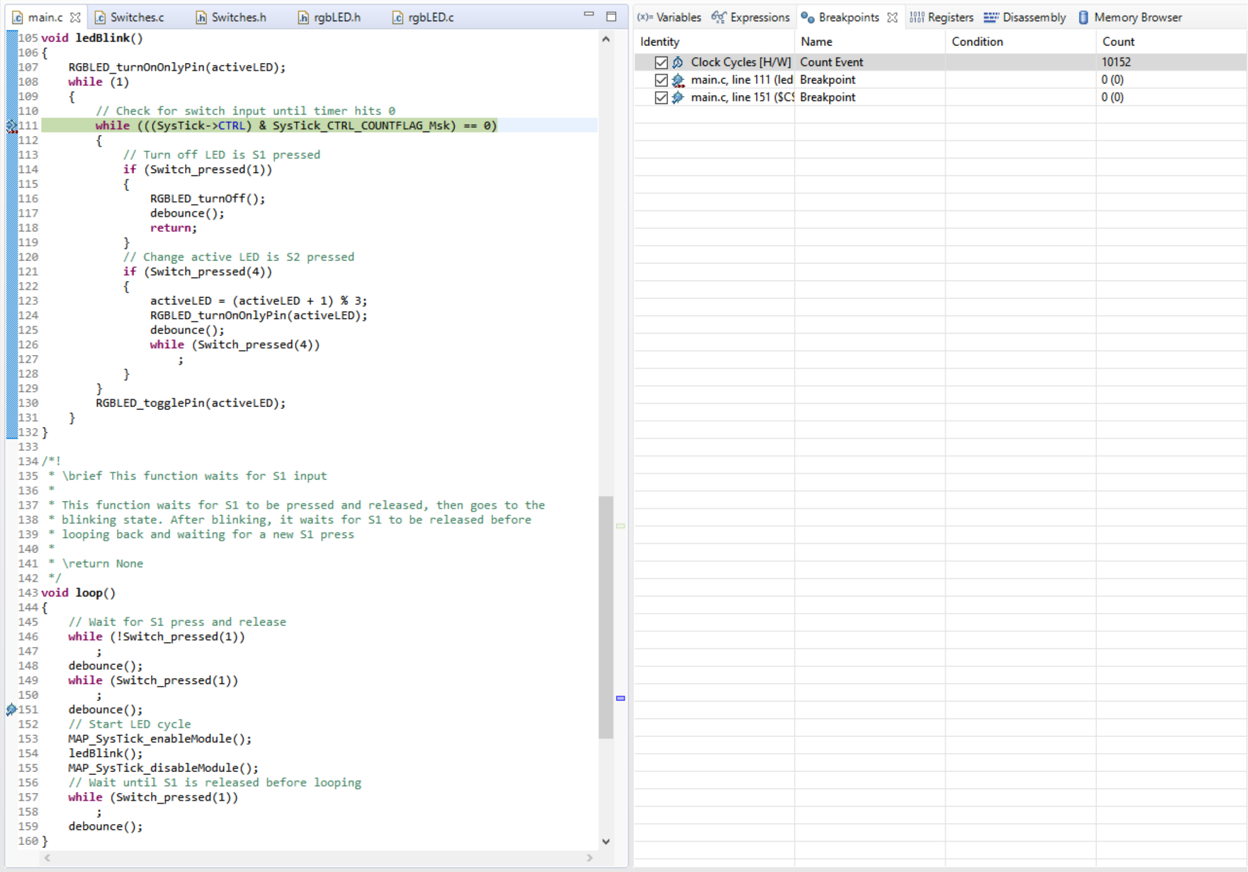
TEST 2

In the code, the variable `activeLED` is used to keep track of which LED is active. Its value correlates to the pin of the RGB LED. It is initialized with a value of `0`, which corresponds to the red LED. When S1 is first pressed and released, the red LED turns on.

Meets criteria.

TEST 3

The following code analysis depicts the code between S1 being released (breakpoint at line 151) and just after the LED turning on (breakpoint at line 111). By looking at the clock cycle counter, it can be determined how long it took.



```c
void ledBlink()
{
    RGBLED_turnOnOnlyPin(activeLED);
    while (1)
    {
        // Check for switch input until timer hits 0
        while (((SysTick->CTRL) & SysTick_CTRL_COUNTFLAG_Msk) == 0)
        {
            // Turn off LED is S1 pressed
            if (Switch_pressed(1))
            {
                RGBLED_turnOff();
                debounce();
                return;
            }
            // Change active LED is S2 pressed
            if (Switch_pressed(4))
            {
                activeLED = (activeLED + 1) % 3;
                RGBLED_turnOnOnlyPin(activeLED);
                debounce();
                while (Switch_pressed(4))
                    ;
            }
        }
        RGBLED_togglePin(activeLED);
    }
}

/*!
 * \brief This function waits for S1 input
 *
 * This function waits for S1 to be pressed and released, then goes to the
 * blinking state. After blinking, it waits for S1 to be released before
 * looping back and waiting for a new S1 press
 *
 * \return None
 */
void loop()
{
    // Wait for S1 press and release
    while (!Switch_pressed(1))
        ;
    debounce();
    while (Switch_pressed(1))
        ;
    debounce();
    // Start LED cycle
    MAP_SysTick_enableModule();
    ledBlink();
    MAP_SysTick_disableModule();
    // Wait until S1 is released before looping
    while (Switch_pressed(1))
        ;
    debounce();
}
```

Identity	Name	Condition	Count
☑ Clock Cycles [H/W]	Count Event		10152
☑ main.c, line 111 (led	Breakpoint		0 (0)
☑ main.c, line 151 ($CS	Breakpoint		0 (0)

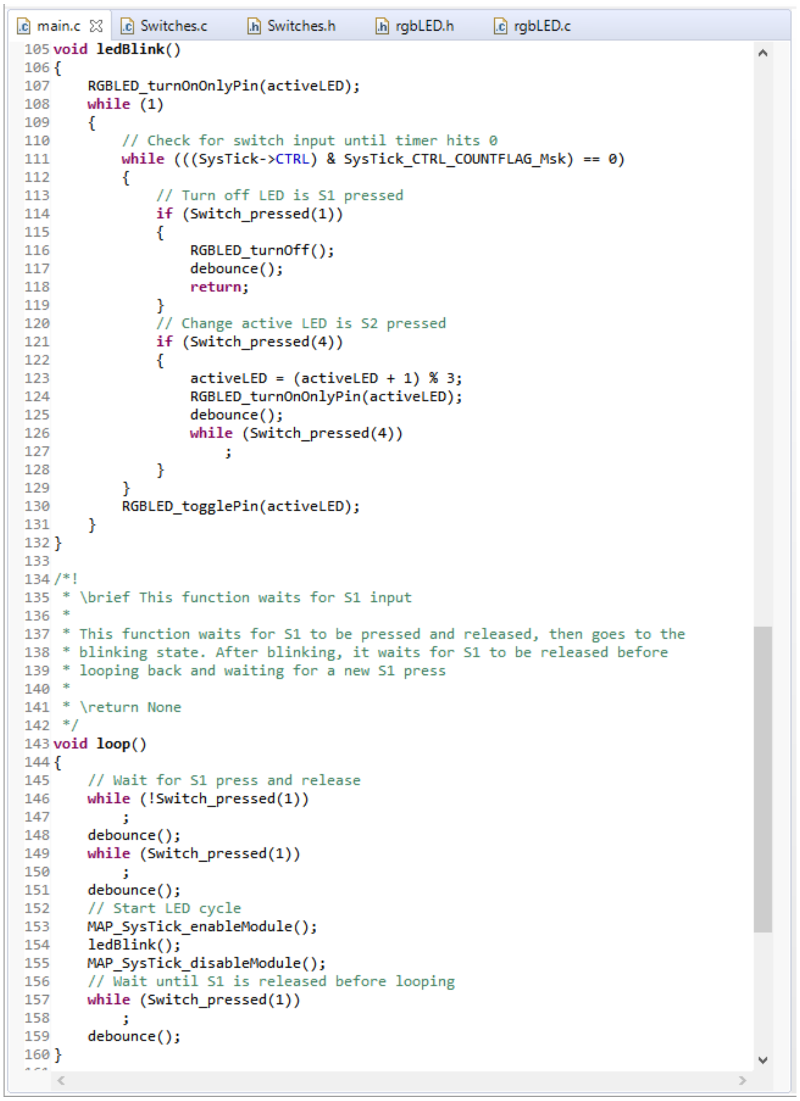
The following calculation shows how long the process took given that the MSP432P401R clock rate is 3MHz:

$$t = \frac{10152\ clock\ cycles}{3 \times 10^{6} cycles\ per\ second} = 3.384ms$$

Meets criteria.

TEST4

The following code displays the parts of the program that check for switch inputs.



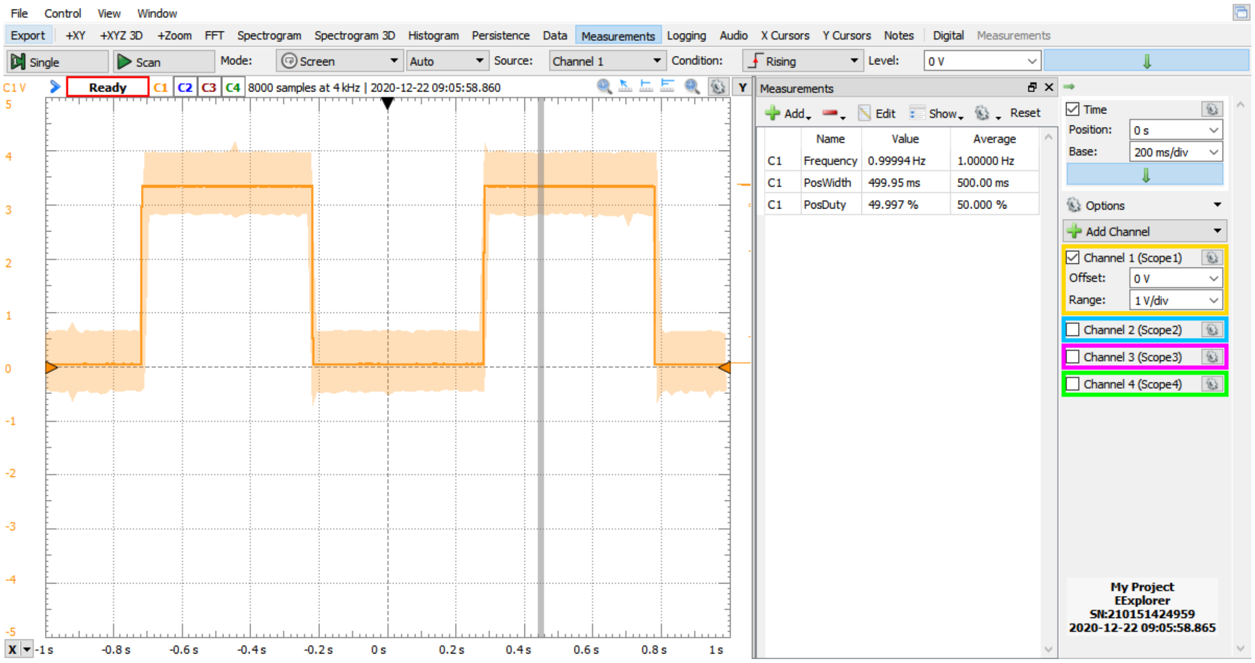
```c
105 void ledBlink()
106 {
107     RGBLED_turnOnOnlyPin(activeLED);
108     while (1)
109     {
110         // Check for switch input until timer hits 0
111         while (((SysTick->CTRL) & SysTick_CTRL_COUNTFLAG_Msk) == 0)
112         {
113             // Turn off LED is S1 pressed
114             if (Switch_pressed(1))
115             {
116                 RGBLED_turnOff();
117                 debounce();
118                 return;
119             }
120             // Change active LED is S2 pressed
121             if (Switch_pressed(4))
122             {
123                 activeLED = (activeLED + 1) % 3;
124                 RGBLED_turnOnOnlyPin(activeLED);
125                 debounce();
126                 while (Switch_pressed(4))
127                     ;
128             }
129         }
130         RGBLED_togglePin(activeLED);
131     }
132 }
133
134 /*!
135  * \brief This function waits for S1 input
136  *
137  * This function waits for S1 to be pressed and released, then goes to the
138  * blinking state. After blinking, it waits for S1 to be released before
139  * looping back and waiting for a new S1 press
140  *
141  * \return None
142  */
143 void loop()
144 {
145     // Wait for S1 press and release
146     while (!Switch_pressed(1))
147         ;
148     debounce();
149     while (Switch_pressed(1))
150         ;
151     debounce();
152     // Start LED cycle
153     MAP_SysTick_enableModule();
154     ledBlink();
155     MAP_SysTick_disableModule();
156     // Wait until S1 is released before looping
157     while (Switch_pressed(1))
158         ;
159     debounce();
160 }
```

After every single switch press or release (lines 117, 125, 148, 151, 159) the software calls the `debounce` method which causes a delay and acts as a software debounce.

Meets criteria.

TEST 5

In order to verify the LED follows the blinking pattern, an oscilloscope was used. The following is a snapshot of the oscilloscope and the LED output. Due to slight inconsistencies, how long the LED blinks (PosWidth) and the frequency varies. However, on average the LED blinks for 500.00ms and has a 50% duty cycle.



Mode: Screen, Auto | Source: Channel 1 | Condition: Rising | Level: 0 V

8000 samples at 4 kHz | 2020-12-22 09:05:58.860

Measurements

	Name	Value	Average
C1	Frequency	0.99994 Hz	1.00000 Hz
C1	PosWidth	499.95 ms	500.00 ms
C1	PosDuty	49.997 %	50.000 %

☑ Time
Position: 0 s
Base: 200 ms/div

☑ Channel 1 (Scope1)
Offset: 0 V
Range: 1 V/div

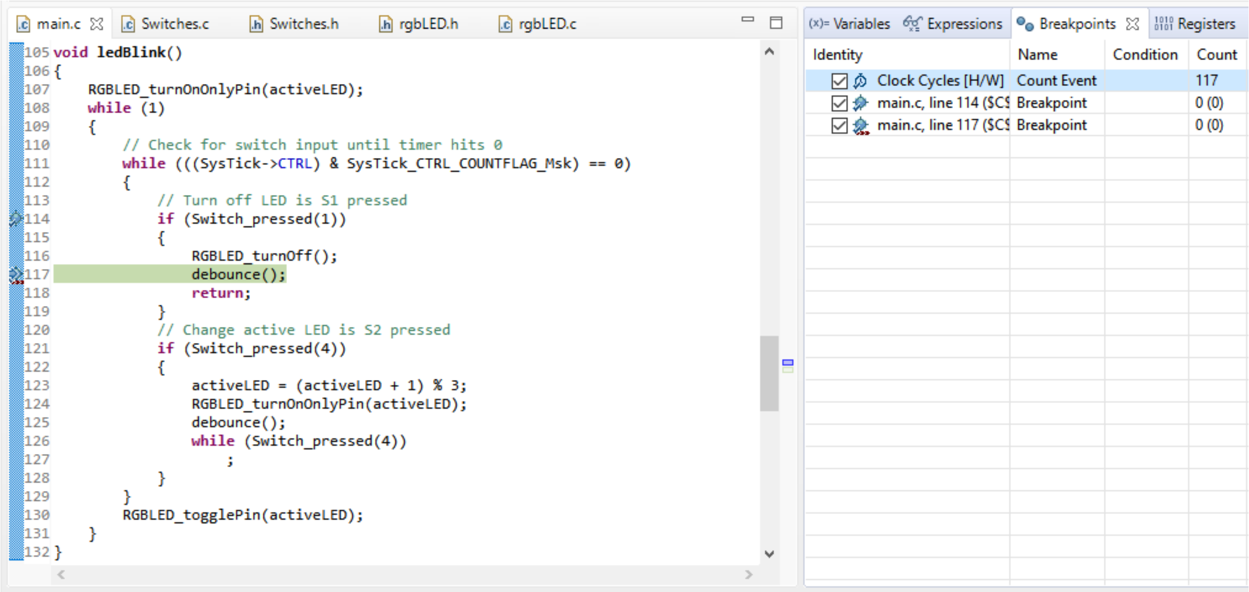
☐ Channel 2 (Scope2)
☐ Channel 3 (Scope3)
☐ Channel 4 (Scope4)

My Project
EExplorer
SN:210151424959
2020-12-22 09:05:58.865

Meets criteria.

TEST 6

The following code analysis depicts the code between S1 being pressed (breakpoint at line 114) and after the LED turning off (breakpoint at line 117). By looking at the clock cycle counter, it can be determined how long it took.



```
105 void ledBlink()
106 {
107     RGBLED_turnOnOnlyPin(activeLED);
108     while (1)
109     {
110         // Check for switch input until timer hits 0
111         while (((SysTick->CTRL) & SysTick_CTRL_COUNTFLAG_Msk) == 0)
112         {
113             // Turn off LED is S1 pressed
114             if (Switch_pressed(1))
115             {
116                 RGBLED_turnOff();
117                 debounce();
118                 return;
119             }
120             // Change active LED is S2 pressed
121             if (Switch_pressed(4))
122             {
123                 activeLED = (activeLED + 1) % 3;
124                 RGBLED_turnOnOnlyPin(activeLED);
125                 debounce();
126                 while (Switch_pressed(4))
127                     ;
128             }
129         }
130         RGBLED_togglePin(activeLED);
131     }
132 }
```

Identity	Name	Condition	Count
☑ Clock Cycles [H/W]	Count Event		117
☑ main.c, line 114 ($CS	Breakpoint		0 (0)
☑ main.c, line 117 ($CS	Breakpoint		0 (0)

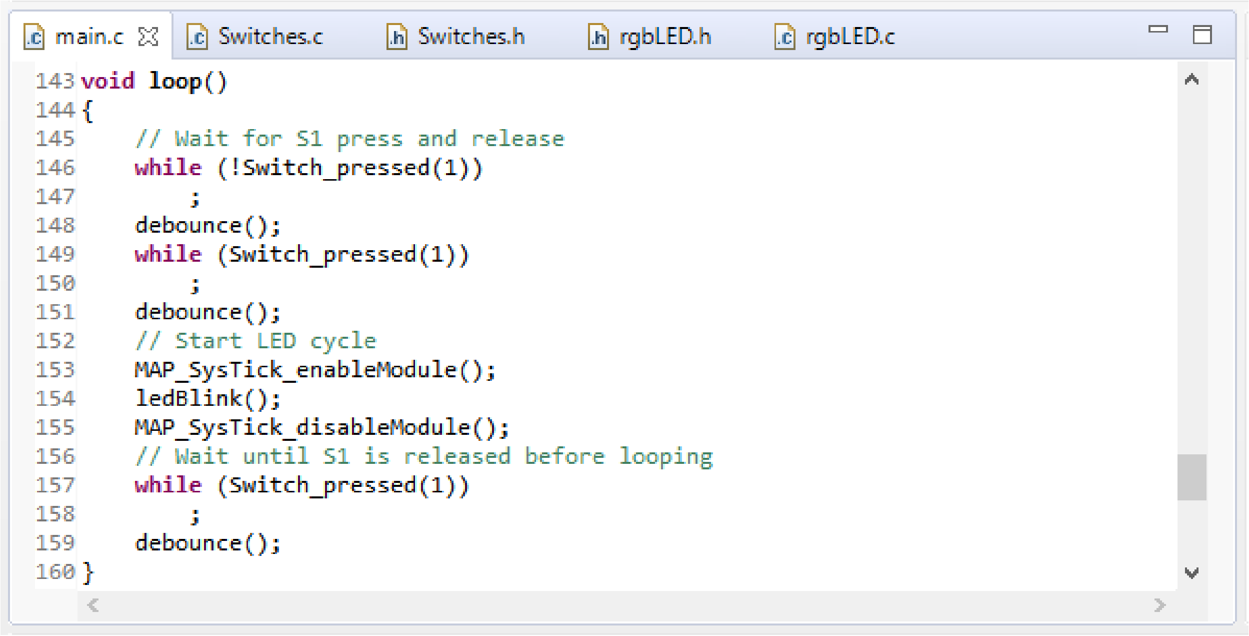
The following calculation shows how long the process took given that the MSP432P401R clock rate is 3MHz:

$$t = \frac{117 \; clock \; cycles}{3 \times 10^6 cycles \; per \; second} = 39\mu s$$

Meets criteria.

TEST 7

The following code displays the `loop` function where after returning from `ledBlink`, it will wait until S1 is released before looping again.



```
void loop()
{
    // Wait for S1 press and release
    while (!Switch_pressed(1))
        ;
    debounce();
    while (Switch_pressed(1))
        ;
    debounce();
    // Start LED cycle
    MAP_SysTick_enableModule();
    ledBlink();
    MAP_SysTick_disableModule();
    // Wait until S1 is released before looping
    while (Switch_pressed(1))
        ;
    debounce();
}
```

Meets criteria.

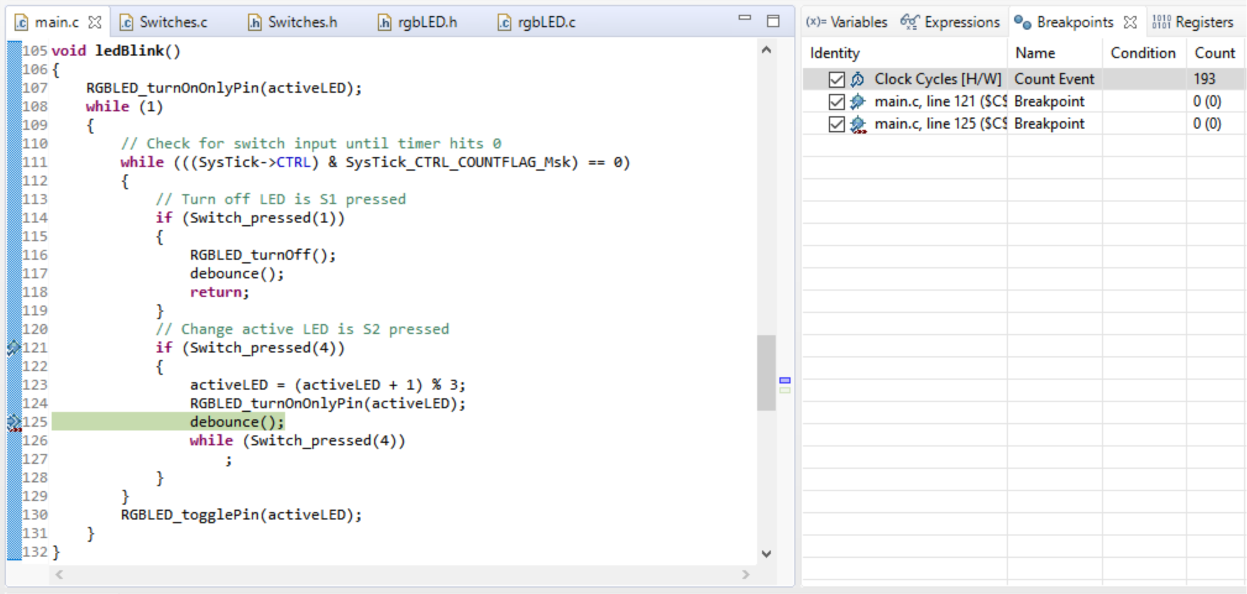
TEST 8

When S2 is pressed while LED2 is in the blinking cycle, the system cycles between the red, green, and blue LED. This is accomplished by incrementing `activeLED` by one, then using the modulo operator to ensure it stays within the range of 0 to 2.

Meets criteria.

TEST 9

The following code analysis depicts the code between S2 being pressed (breakpoint at line 121) and after the former active LED turning off and the new active current LED turning on (breakpoint line 125). By looking at the clock cycle counter, it can be determined how long it took.



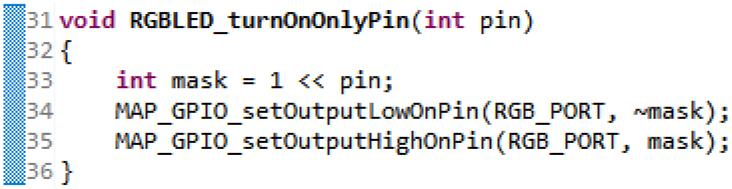
```c
105 void ledBlink()
106 {
107     RGBLED_turnOnOnlyPin(activeLED);
108     while (1)
109     {
110         // Check for switch input until timer hits 0
111         while (((SysTick->CTRL) & SysTick_CTRL_COUNTFLAG_Msk) == 0)
112         {
113             // Turn off LED is S1 pressed
114             if (Switch_pressed(1))
115             {
116                 RGBLED_turnOff();
117                 debounce();
118                 return;
119             }
120             // Change active LED is S2 pressed
121             if (Switch_pressed(4))
122             {
123                 activeLED = (activeLED + 1) % 3;
124                 RGBLED_turnOnOnlyPin(activeLED);
125                 debounce();
126                 while (Switch_pressed(4))
127                     ;
128             }
129         }
130         RGBLED_togglePin(activeLED);
131     }
132 }
```

Identity	Name	Condition	Count
☑ Clock Cycles [H/W]	Count Event		193
☑ main.c, line 121 (C	Breakpoint		0 (0)
☑ main.c, line 125 (C	Breakpoint		0 (0)

The following calculation shows how long the process took given that the MSP432P401R clock rate is 3MHz:

$$t = \frac{193\ clock\ cycles}{3 \times 10^6 cycles\ per\ second} = 64.\overline{3}\mu s$$

Additionally, the following code from `rgbLED.c` shows that all LEDs except the current active LED are first turned off, then the new active LED is turned on



```
void RGBLED_turnOnOnlyPin(int pin)
{
    int mask = 1 << pin;
    MAP_GPIO_setOutputLowOnPin(RGB_PORT, ~mask);
    MAP_GPIO_setOutputHighOnPin(RGB_PORT, mask);
}
```

Meets criteria.

TEST 10

The LED only changes once per press of S2. Regardless if S2 is pressed or held.

Meets criteria.

TEST 11

While S2 is being held, the active LED remains on and the S1 does not change the system.

Meets criteria.

CONCLUSION

All the tests met the criteria for both the basic and advanced requirements.

DEMO LINK

The following a YouTube link demonstrating the project:

https://youtu.be/zMKqCcwCVW0