**EGR 226: Microcontroller Programming and Applications**

**Winter 2021**

Instructor: Professor Trevor Ekins

Lab 5: External Inputs and Outputs (Breadboard LED’s + Switches)

Sam Wieneke

February 24, 2021

**Contents**

1. **Objectives**
2. **Equipment**
3. **Introduction**
4. **Procedure**
5. **Results/ Discussion**
6. **Conclusion and Future Work**

**Appendices**

1. **Source Code: Part 1**

**Source Code: Part 2**

1. **Objectives**

The objectives of this lab were to develop a program for the MSP432 that interfaces with pushbutton switches to control the sequencing of lighting different colored LED’s, to develop a program for the MSP432 to generate time intervals using Systick Timer, and to build a LED circuit and pushbutton circuit.

1. **Equipment**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Part | Description | Model | Measured Value | Notes |
| Code Composer Studios | Texas Instruments programming environment | Version 9.3.0 | N/A | N/A |
| GitLab | Remote Repository for code maintenance | N/A | N/A | Makes collaboration on team projects and code very convenient. |
|  |  |  |  |  |

1. **Introduction**

**Part 1: Sequencing Colored LED’s using a pushbutton switch**

Part 1 involved using external LED’s to change when an external push button was pressed. A debouncing function was needed for the button to be functional. The program was a loop and could go on forever if the user pleased and the lights will be in a sequential order. If the button was held down, it would not advance to the next color unless the button was released.

**Part 2: Sequencing Colored LED’s using timer and pushbutton switch**

Part 2 involved controlling the external LED sequence using a time delay. The program sequenced through the colors as long as the button was pushed and when it was released, it would stop on the color it was on.

**Part 3: Reversing the Order of the Colored LED’s**

Part 3 involved the same idea as part 2, however it would reverse the order of the LED’s from whatever it was left on in part 2.

1. **Procedure**

**Part 1**

This code was created by making a large looping function that will run forever, and within that loop, another loop that runs when the button is pushed. There are if statements within this loop that will turn off a color and turn on another color, each color will increment a counter which will determine which if statement should be next. There is a debounce function that will eliminate any bouncing caused by pushing the button. While the button is held down it will not advance to the next color because of the while debounce function.

**Part 2**

Part 2 was fairly similar to part 1, but in this part of the lab the colors had to be changing by themselves when the button was pushed and would stop on the color if the button was released. A function Systick, was used to determine how long the light will stay on before moving to the next color. This function is called in the main and is shown in Figure 4.1.

Figure 4.1: Delay Cycles

SysTick\_delay(1000);

This function will keep the light on for 1 second.

This code was created by having on main looping function that ran while the button was pushed. Within this loop, there were if statements that determined what color would be displayed along with a delay, so the lights do not change instantly. There was also a counter which allowed for the changing of the lights and worked as a loop where it would go back to the first color when it reached a certain number.

**Part 3**

Part 3 is almost identical to part 2 however, it is the opposite because the function was designed to reverse the order from part 2. The counter was changed from i++ to i- - and the if statements criteria were changed so it would make since while decrementing.

1. **Results/ Discussion**

**Lab Results**

To demonstrate this lab, part 1 had to be shown to the instructor as well as part 2 and 3. For part 1, the button had to be pushed to change the color and it didn’t change on its own. If it was held down, it would not switch to the next light until the button was released. For part 2 and 3, the lights should change for as long as the button was held and be able to stop on the color if the button was released. Part 3 was the same as part 2, but it reversed the order of the lights.

**Prelab:**

The prelab involved calculating minimum resistor values for each colored LED’s. With these values (Red: 140 ohms, Yellow: 150 ohms, Green: 90 ohms) a resistor was needed to be chosen that was close to the value but not lower. Three resistors were chosen (100 ohms, 156 ohms, 156 ohms). These resistors were used to complete the circuit so the LED’s would not fry.

1. **Conclusion/ Future Work**

This lab was the fist time using external devices with the MSP 432. It was difficult making sure all the wires were hooked up correctly and the resistors were where they needed to be. The code was fairly simple, because it was almost identical to Lab 4, except for part 3. Nothing could be thought of to improve the design of the code of the lab, it was the same as Lab 4 based on design. Maybe adding a way to change the brightness of the lights would be beneficial.

**Appendix A**

Part 1 Source Code:

**#include** "msp.h"

**int** **DebounceSwitch1**(**void**);

**void** **main**(**void**)

{

WDT\_A->CTL = WDT\_A\_CTL\_PW | WDT\_A\_CTL\_HOLD; // stop watchdog timer

P3->SEL1 &= ~BIT6; // configure P3.6 as simple I/O

P3->SEL0 &= ~BIT6;

P3->DIR &= ~BIT6; // P3.6 set as output pin \*/

P3->REN |= BIT6;

P3->OUT |= BIT6;

P3->SEL1 &= ~BIT7; // configure P3.7 as simple I/O

P3->SEL0 &= ~BIT7;

P3->DIR &= ~BIT7; // P3.7 set as output pin \*/

P3->REN |= BIT7;

P3->OUT |= BIT7;

P2->SEL1 &= ~BIT5; // configure P2.5 as simple I/O

P2->SEL0 &= ~BIT5;

P2->DIR |= BIT5; // P2.5 set as output pin \*/

P3->SEL1 &= ~BIT0; // configure P3.0 as simple I/O

P3->SEL0 &= ~BIT0;

P3->DIR |= BIT0; // P3.0 set as output pin \*/

P5->SEL1 &= ~BIT7; // configure P5.7 as simple I/O

P5->SEL0 &= ~BIT7;

P5->DIR |= BIT7; // P5.7 set as output pin \*/

**int** i=0;

P2->OUT &= ~BIT5;

P3->OUT &= ~BIT0;

P5->OUT &= ~BIT7;

**while** (1) {

**if**(((P3->IN & BIT7) != BIT7))

{

**while**(DebounceSwitch1())

{

}

**if**(i==0)

{

P2->OUT &= ~BIT5;

P3->OUT &= ~BIT0;

P5->OUT ^= BIT7;

i++;

}

**else** **if**(i==1)

{

P2->OUT &= ~BIT5;

P5->OUT &= ~BIT7;

P3->OUT ^= BIT0;

i++;

}

**else** **if**(i==2)

{

P5->OUT &= ~BIT7;

P3->OUT &= ~BIT0;

P2->OUT ^= BIT5;

i=0;

}

}

}

}

**int** **DebounceSwitch1**(**void**)

{

**int** pin\_Value = 0; //initialize variable as low

**if** ((P3IN & 0x80) == 0x00) //check of button pushed

{ \_\_delay\_cycles(300000); //pause for 10 m-sec for switch bounce

**if** ((P3IN & 0x80) == 0x00) //check of button is still pushed

pin\_Value = 1;

}

**return** pin\_Value; //return 1 if pushed- 0 if not pushed

}

Part 2 Source Code:

**#include** "msp.h"

**int** **DebounceSwitch1**(**void**);

**void** **SysTick\_Init** (**void**);

**void** **SysTick\_delay** (uint16\_t delay);

**void** **main**(**void**)

{

WDT\_A->CTL = WDT\_A\_CTL\_PW | WDT\_A\_CTL\_HOLD; // stop watchdog timer

SysTick\_Init ();

P3->SEL1 &= ~BIT6; // configure P3.6 as simple I/O

P3->SEL0 &= ~BIT6;

P3->DIR &= ~BIT6; // P3.6 set as output pin \*/

P3->REN |= BIT6;

P3->OUT |= BIT6;

P3->SEL1 &= ~BIT7; // configure P3.7 as simple I/O

P3->SEL0 &= ~BIT7;

P3->DIR &= ~BIT7; // P3.7 set as output pin \*/

P3->REN |= BIT7;

P3->OUT |= BIT7;

P2->SEL1 &= ~BIT5; // configure P2.5 as simple I/O

P2->SEL0 &= ~BIT5;

P2->DIR |= BIT5; // P2.5 set as output pin \*/

P3->SEL1 &= ~BIT0; // configure P3.0 as simple I/O

P3->SEL0 &= ~BIT0;

P3->DIR |= BIT0; // P3.0 set as output pin \*/

P5->SEL1 &= ~BIT7; // configure P5.7 as simple I/O

P5->SEL0 &= ~BIT7;

P5->DIR |= BIT7; // P5.7 set as output pin \*/

**int** i=0;

P2->OUT &= ~BIT5;

P3->OUT &= ~BIT0;

P5->OUT &= ~BIT7;

**while** (1) {

**while**((P3->IN & BIT7) != BIT7)

{

**if**(i==3)

{

i=0;

}

**if**(i==0)

{

P3->OUT &= ~BIT0;

P2->OUT &= ~BIT5;

P5->OUT |= BIT7; //green

SysTick\_delay(1000);

i++;

}

**else** **if**(i==1)

{

P2->OUT &= ~BIT5;

P5->OUT &= ~BIT7;

P3->OUT |= BIT0; //yellow

SysTick\_delay(1000);

i++;

}

**else** **if**(i==2)

{

P5->OUT &= ~BIT7;

P3->OUT &= ~BIT0;

P2->OUT |= BIT5; //red

SysTick\_delay(1000);

i++;

}

}

**while**((P3->IN & BIT6) != BIT6)

{

**if**(i==0)

{

i=3;

}

**if**(i==1)

{

P3->OUT &= ~BIT0;

P2->OUT &= ~BIT5;

P5->OUT |= BIT7; //green

SysTick\_delay(1000);

i--;

}

**else** **if**(i==2)

{

P2->OUT &= ~BIT5;

P5->OUT &= ~BIT7;

P3->OUT |= BIT0; //yellow

SysTick\_delay(1000);

i--;

}

**else** **if**(i==3)

{

P5->OUT &= ~BIT7;

P3->OUT &= ~BIT0;

P2->OUT |= BIT5; //red

SysTick\_delay(1000);

i--;

}

}

}

}

**int** **DebounceSwitch1**(**void**)

{

**int** pin\_Value = 0; //initialize variable as low

**if** ((P3IN & 0x80) == 0x00) //check of button pushed

{ \_\_delay\_cycles(300000); //pause for 100 m-sec for switch bounce

**if** ((P3IN & 0x80) == 0x00) //check of button is still pushed

pin\_Value = 1;

}

**return** pin\_Value; //return 1 if pushed- 0 if not pushed

}

**void** **SysTick\_Init** (**void**)

{ //initialization of systic timer

SysTick->CTRL = 0; // disable SysTick During step

SysTick->LOAD = 0x00FFFFFF; // max reload value

SysTick->VAL = 0; // any write to current clears it

SysTick->CTRL = 0x00000005; // enable systic, 3MHz, No Interrupts

}

**void** **SysTick\_delay** (uint16\_t delay)

{ // Systick delay function

SysTick->LOAD = ((delay \* 3000) - 1); //delay for 1 msecond per delay value

SysTick->VAL = 0; // any write to CVR clears it

**while** ( (SysTick->CTRL & 0x00010000) == 0); // wait for flag to be SET

}