**EGR 226: Microcontroller Programming and Applications**

**Winter 2021**

Instructor: Professor Trevor Ekins

Lab 9: Using GPIO interrupts to control PWM and 7-Segment LED

Sam Wieneke

March 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**

**Source Code: Part 3**

1. **Objectives**

The objectives of this experiment were to develop a program for the MSP432 that interfaces with a DC motor and controls the speed using PWM by GPIO interrupts and to ger familiar with the 7-Segment LED.

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: Controlling the DC motor using Timer A in PWM Mode**

Part 1 involved controlling a DC motor using TimerA to change the PWM of the DC motor. There was 3 push buttons that controlled the duty cycle, 1 to increase it, 1 to decrease it, and one to turn the motor off completely.

**Part 2: Decimal Counter Using 7 Segment LED**

Part 2 involved using the 7 Segment LED. The code involved displaying numbers 0-9 in increasing order with 1 second delays between each change.

**Part 3: Decimal Counter Using 7\_Segment LED (2)**

Part 3 was part 2 but with buttons that changed the numbers either to increase or decrease, not just a delay that would increment after a certain amount of time.

1. **Procedure**

**Part 1**

Part 1 was setup by taking the code from part 2 of lab 8 and changing it to fit the description. 3 buttons were added, 1 to increase the duty cycle by 10%, 1 to decrease the duty cycle by 10%, and one to completely turn the motor off. The code below shows how it changed the duty cycle based off the button being pushed.

**if**((P1->IN & BIT6) != BIT6) //blue button, increments by 10%

{

duty\_cycle = duty\_cycle +10;

**if** (duty\_cycle == 100)

{

duty\_cycle = 0;

}

DC\_Motor(duty\_cycle);

SysTick\_delay(1000);

}

Basically this code says that if the blue button is pushed, duty cycle will be increased by 10% and if the duty cycle reaches 100%, it will go back to 0%. The new amount for duty cycle will then be passed to the DC\_Motor function which will actually change the duty cycle for the dc motor. It will then delay for 1 second to account for debouncing.

**Part 2**

Part 2 used interrupts to increment the 7 segment LED with a timer. The global volatile variable intervalCnt was very crucial for this code. It is what determined which number will be displayed on the LED. The SysTick\_Handler incremented intervalCnt so that after the 1 second delay that was coded in the SysTick Init, it would switch to the next number. The code shown below is an example of how one of the numbers was turned on.

**if**(intervalCnt == 7)

{

//clear all lights

P4->OUT &= ~BIT3; //a

P4->OUT &= ~BIT4; //b

P4->OUT &= ~BIT5; //c

P4->OUT &= ~BIT6; //d

P4->OUT &= ~BIT7; //e

P5->OUT &= ~BIT4; //f

P5->OUT &= ~BIT5; //g

//turn on lights for 7

P4->OUT |= BIT3; //a

P4->OUT |= BIT4; //b

P4->OUT |= BIT5; //c

}

This code enters the if statement intervalCnt reaches its value. It will then turn off all the LED’s to make sure that none are accidentally left on. It will then only turn on select LED’s to create the desired number on the LED.

**Part 3**

Part 3 was designed off of part 2 but interrupts were used with the push buttons to either increment or decrement the number. The code below shows how in the handler the number was incremented.

**if**(P1->IFG & BIT6)

{

intervalCnt++;

**if**(intervalCnt ==11)

{

intervalCnt =0;

}

sevenSegment(intervalCnt);

SysTick\_delay(750);

}

P1->IFG &= ~BIT6;

In this code, when the flag is tripped for the button it enters the if statement and increments intervalCnt by 1 and checks to make sure intervalCnt isn’t above the max number. It then enters the sevenSegment function and will display the next highest number. After that it clears the flag for this button so it can be used again. To setup the interrupts for the pushbuttons other code was used than normal standard I/O pins this code is shown below.

P1->IES |= BIT6;

P1->IE |= BIT6;

P1->IFG &= ~BIT6;

These 3 statements are needed to set the pin for an interrupt. The IFG clears the flag, the IES sets the pin interrupt to trigger when it goes from high to low, and the IE enables the interrupt. Adding these to pieces of code to part 2 plus a few extra items allows for the buttons to change the number on the LED using interrupts.

1. **Results/ Discussion**

**Lab Results**

There was no lab questions for this laboratory but the demonstration went smoothly as all of the code worked as it was supposed to.

**Prelab:**

There was no prelab for this laboratory.

1. **Conclusion/ Future Work**

This lab was the first use of interrupts. Part 1 was just a build off of lab 8 but parts 2 and 3 were completely new. Getting the interrupt to work for part 2 was fairly simple but for part 3 it was quite challenging. It was believed that all the code was setup correctly for part 3, but it would only display number 0 and the buttons were not interfacing with the code. This was fixed and the code worked fine in the end but it was difficult to get the buttons to work using interrupts this was due to the fact that the Systick code was copied over from a previous lab and the interrupts were still enabled. Some future work that could be implemented into this lab could be to use this code for the final project as a timer of sorts that the user could see clearly.

**Appendix A**

Part 1 Source Code:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Author: Samuel Wieneke

\* Course: EGR 226 - 902

\* Date: 3/17/2021

\* Project: Lab 9

\* File:

\* Description: This program will change the duty cycle of the DC motor within CCS

\* \*\*Uses TimerA

\*

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**#include** "msp.h"

**#include** <stdio.h>

**void** **button\_init**(**void**);

**void** **DC\_Motor** (**int** duty\_cycle);

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

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

**int** duty\_cycle=20;

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

{

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

DC\_Motor(duty\_cycle);

button\_init();

SysTick\_Init();

**while**(1)

{

**if**((P1->IN & BIT6) != BIT6) //blue button, increments by 10%

{

duty\_cycle = duty\_cycle +10;

**if** (duty\_cycle == 100)

{

duty\_cycle = 0;

}

DC\_Motor(duty\_cycle);

SysTick\_delay(1000);

}

**if**((P1->IN & BIT7) != BIT7) //red button, decrements by 10%

{

duty\_cycle = duty\_cycle -10;

**if** (duty\_cycle == 0)

{

duty\_cycle = 100;

}

DC\_Motor(duty\_cycle);

SysTick\_delay(1000);

}

**if**((P4->IN & BIT2) != BIT2) //white button, turns motor off

{

duty\_cycle = 0;

DC\_Motor(duty\_cycle);

SysTick\_delay(1000);

}

}

}

/\*\*\*\*| DC\_Motor | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Brief: This function is to initialize Timer A and the Pin

\* connected to the MSP

\* param:

\* (void) data: N/A

\* return:

\* N/A

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** **DC\_Motor** (**int** duty\_cycle)

{

//P2.4 is connected to DC Motor

// Configure GPIO for PWM output

P2->SEL0 |= BIT4;

P2->SEL1 &= ~(BIT4);

P2->DIR |= BIT4; // P2.4 set TA0.1

TIMER\_A0->CCR[0] = 30000 - 1; // PWM Period (# cycles of clock)

TIMER\_A0->CCTL[1] = TIMER\_A\_CCTLN\_OUTMOD\_7; // CCR1 reset/set mode 7

TIMER\_A0->CCR[1] = (30000 \* (duty\_cycle/100.0)); // CCR1 PWM duty cycle in 10ths of percent

TIMER\_A0->CTL = TIMER\_A\_CTL\_SSEL\_\_SMCLK | // use SMCLK

TIMER\_A\_CTL\_MC\_\_UP | // in Up mode

TIMER\_A\_CTL\_CLR; // Clear TAR to start

}

/\*\*\*\*| button\_init | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Brief: This function is to initialize the buttons

\* param:

\* (void) data: N/A

\* return:

\* N/A

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** **button\_init**(**void**)

{

// Blue

P1->SEL1 &= ~BIT6; // configure P1.6 as simple I/O

P1->SEL0 &= ~BIT6;

P1->DIR &= ~BIT6; // P1.6 set as output pin \*/

P1->REN |= BIT6;

P1->OUT |= BIT6;

// Red

P1->SEL1 &= ~BIT7; // configure P1.7 as simple I/O

P1->SEL0 &= ~BIT7;

P1->DIR &= ~BIT7; // P1.7 set as output pin \*/

P1->REN |= BIT7;

P1->OUT |= BIT7;

// White

P4->SEL1 &= ~BIT2; // configure P4.2 as simple I/O

P4->SEL0 &= ~BIT2;

P4->DIR &= ~BIT2; // P4.2 set as output pin \*/

P4->REN |= BIT2;

P4->OUT |= BIT2;

}

/\*\*\*\*| SysTick\_Init | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Brief: This function is used for the initialization

\* of the SysTick timer

\* param:

\* (void) data: N/A

\* return:

\* N/A

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**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

}

/\*\*\*\*| SysTick\_delay | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Brief: This function sets the SysTick timer to be used as a

\* delay when called

\* param:

\* (uint16\_t) data: one integer worth of data input that

\* determines the duration of the delay

\* return:

\* N/A

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**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

}

Part 2 Source Code:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Author: Samuel Wieneke

\* Course: EGR 226 - 902

\* Date: 3/17/2021

\* Project: Lab 9

\* File:

\* Description: This program will act as a 10 second counter for the 7 segment LED

\*

\*

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**#include** "msp.h"

**#include** <stdio.h>

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

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

**void** **pin\_init** (**void**);

**void** **SysTick\_Handler** (**void**);

**volatile** uint32\_t intervalCnt;

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

{

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

SysTick\_Init();

\_\_enable\_irq();

pin\_init();

**while** (1)

{

**if**(intervalCnt == 0)

{

//clear all lights

P4->OUT &= ~BIT3; //a

P4->OUT &= ~BIT4; //b

P4->OUT &= ~BIT5; //c

P4->OUT &= ~BIT6; //d

P4->OUT &= ~BIT7; //e

P5->OUT &= ~BIT4; //f

P5->OUT &= ~BIT5; //g

//turn on lights for 0

P4->OUT |= BIT3; //a

P4->OUT |= BIT4; //b

P4->OUT |= BIT5; //c

P4->OUT |= BIT6; //d

P4->OUT |= BIT7; //e

P5->OUT |= BIT4; //f

}

**if**(intervalCnt == 1)

{

//clear all lights

P4->OUT &= ~BIT3; //a

P4->OUT &= ~BIT4; //b

P4->OUT &= ~BIT5; //c

P4->OUT &= ~BIT6; //d

P4->OUT &= ~BIT7; //e

P5->OUT &= ~BIT4; //f

P5->OUT &= ~BIT5; //g

//turn on lights for 1

P4->OUT |= BIT4; //b

P4->OUT |= BIT5; //c

}

**if**(intervalCnt == 2)

{

//clear all lights

P4->OUT &= ~BIT3; //a

P4->OUT &= ~BIT4; //b

P4->OUT &= ~BIT5; //c

P4->OUT &= ~BIT6; //d

P4->OUT &= ~BIT7; //e

P5->OUT &= ~BIT4; //f

P5->OUT &= ~BIT5; //g

//turn on lights for 2

P4->OUT |= BIT3; //a

P4->OUT |= BIT4; //b

P4->OUT |= BIT6; //d

P4->OUT |= BIT7; //e

P5->OUT |= BIT5; //g

}

**if**(intervalCnt == 3)

{

//clear all lights

P4->OUT &= ~BIT3; //a

P4->OUT &= ~BIT4; //b

P4->OUT &= ~BIT5; //c

P4->OUT &= ~BIT6; //d

P4->OUT &= ~BIT7; //e

P5->OUT &= ~BIT4; //f

P5->OUT &= ~BIT5; //g

//turn on lights for 3

P4->OUT |= BIT3; //a

P4->OUT |= BIT4; //b

P4->OUT |= BIT5; //c

P4->OUT |= BIT6; //d

P5->OUT |= BIT5; //g

}

**if**(intervalCnt == 4)

{

//clear all lights

P4->OUT &= ~BIT3; //a

P4->OUT &= ~BIT4; //b

P4->OUT &= ~BIT5; //c

P4->OUT &= ~BIT6; //d

P4->OUT &= ~BIT7; //e

P5->OUT &= ~BIT4; //f

P5->OUT &= ~BIT5; //g

//turn on lights for 4

P4->OUT |= BIT4; //b

P4->OUT |= BIT5; //c

P5->OUT |= BIT4; //f

P5->OUT |= BIT5; //g

}

**if**(intervalCnt == 5)

{

//clear all lights

P4->OUT &= ~BIT3; //a

P4->OUT &= ~BIT4; //b

P4->OUT &= ~BIT5; //c

P4->OUT &= ~BIT6; //d

P4->OUT &= ~BIT7; //e

P5->OUT &= ~BIT4; //f

P5->OUT &= ~BIT5; //g

//turn on lights for 5

P4->OUT |= BIT3; //a

P4->OUT |= BIT5; //c

P4->OUT |= BIT6; //d

P5->OUT |= BIT4; //f

P5->OUT |= BIT5; //g

}

**if**(intervalCnt == 6)

{

//clear all lights

P4->OUT &= ~BIT3; //a

P4->OUT &= ~BIT4; //b

P4->OUT &= ~BIT5; //c

P4->OUT &= ~BIT6; //d

P4->OUT &= ~BIT7; //e

P5->OUT &= ~BIT4; //f

P5->OUT &= ~BIT5; //g

//turn on lights for 6

P4->OUT |= BIT3; //a

P4->OUT |= BIT5; //c

P4->OUT |= BIT6; //d

P4->OUT |= BIT7; //e

P5->OUT |= BIT4; //f

P5->OUT |= BIT5; //g

}

**if**(intervalCnt == 7)

{

//clear all lights

P4->OUT &= ~BIT3; //a

P4->OUT &= ~BIT4; //b

P4->OUT &= ~BIT5; //c

P4->OUT &= ~BIT6; //d

P4->OUT &= ~BIT7; //e

P5->OUT &= ~BIT4; //f

P5->OUT &= ~BIT5; //g

//turn on lights for 7

P4->OUT |= BIT3; //a

P4->OUT |= BIT4; //b

P4->OUT |= BIT5; //c

}

**if**(intervalCnt == 8)

{

//clear all lights

P4->OUT &= ~BIT3; //a

P4->OUT &= ~BIT4; //b

P4->OUT &= ~BIT5; //c

P4->OUT &= ~BIT6; //d

P4->OUT &= ~BIT7; //e

P5->OUT &= ~BIT4; //f

P5->OUT &= ~BIT5; //g

//turn on lights for 8

P4->OUT |= BIT3; //a

P4->OUT |= BIT4; //b

P4->OUT |= BIT5; //c

P4->OUT |= BIT6; //d

P4->OUT |= BIT7; //e

P5->OUT |= BIT4; //f

P5->OUT |= BIT5; //g

}

**if**(intervalCnt == 9)

{

//clear all lights

P4->OUT &= ~BIT3; //a

P4->OUT &= ~BIT4; //b

P4->OUT &= ~BIT5; //c

P4->OUT &= ~BIT6; //d

P4->OUT &= ~BIT7; //e

P5->OUT &= ~BIT4; //f

P5->OUT &= ~BIT5; //g

//turn on lights for 9

P4->OUT |= BIT3; //a

P4->OUT |= BIT4; //b

P4->OUT |= BIT5; //c

P5->OUT |= BIT4; //f

P5->OUT |= BIT5; //g

}

}

}

/\*\*\*\*| pin\_init | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Brief: This function is used for the initialization

\* of the pins for the 7 segment LED

\* param:

\* (void) data: N/A

\* return:

\* N/A

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** **pin\_init** (**void**)

{

//a

P4->SEL1 &= ~BIT3; // configure P4.3 as simple I/O

P4->SEL0 &= ~BIT3;

P4->DIR |= BIT3; // P4.3 set as output pin \*/

//b

P4->SEL1 &= ~BIT4; // configure P4.4 as simple I/O

P4->SEL0 &= ~BIT4;

P4->DIR |= BIT4; // P4.4 set as output pin \*/

//c

P4->SEL1 &= ~BIT5; // configure P4.5 as simple I/O

P4->SEL0 &= ~BIT5;

P4->DIR |= BIT5; // P4.5 set as output pin \*/

//d

P4->SEL1 &= ~BIT6; // configure P4.5 as simple I/O

P4->SEL0 &= ~BIT6;

P4->DIR |= BIT6; // P4.5 set as output pin \*/

//e

P4->SEL1 &= ~BIT7; // configure P4.5 as simple I/O

P4->SEL0 &= ~BIT7;

P4->DIR |= BIT7; // P4.5 set as output pin \*/

//f

P5->SEL1 &= ~BIT4; // configure P5.4 as simple I/O

P5->SEL0 &= ~BIT4;

P5->DIR |= BIT4; // P5.4 set as output pin \*/

//g

P5->SEL1 &= ~BIT5; // configure P4.5 as simple I/O

P5->SEL0 &= ~BIT5;

P5->DIR |= BIT5; // P4.5 set as output pin \*/

}

/\*\*\*\*| SysTick\_Handler | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Brief: This function is used for the SysTick Interrupt and will increment

\* intervalCnt

\* param:

\* (void) data: N/A

\* return:

\* N/A

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** **SysTick\_Handler** (**void**)

{

intervalCnt++;

**if** (intervalCnt == 10)

{

intervalCnt = 0;

}

}

/\*\*\*\*| SysTick\_Init | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Brief: This function is used for the initialization

\* of the SysTick timer

\* param:

\* (void) data: N/A

\* return:

\* N/A

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

{ //initialization of systic timer

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

SysTick->LOAD = 3000000; // 1 sec reload value

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

SysTick->CTRL = 0x00000007; // enable systic, 3MHz, Interrupts

}

/\*\*\*\*| SysTick\_delay | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Brief: This function sets the SysTick timer to be used as a

\* delay when called

\* param:

\* (uint16\_t) data: one integer worth of data input that

\* determines the duration of the delay

\* return:

\* N/A

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**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

}

Part 3 Source Code:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Author: Samuel Wieneke

\* Course: EGR 226 - 902

\* Date: 3/17/2021

\* Project: Lab 9

\* File:

\* Description: This program will allow the user to increment or decrement the LED using

\* pushbuttons

\*

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**#include** "msp.h"

**#include** <stdio.h>

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

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

**void** **pin\_init** (**void**);

**void** **PORT1\_IRQHandler** (**void**);

**void** **setupInterrupts** (**void**);

**void** **sevenSegment** (uint32\_t intervalCnt);

**volatile** uint32\_t intervalCnt=0;

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

{

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

SysTick\_Init();

pin\_init();

NVIC->ISER[1] = 1 << ((*PORT1\_IRQn*) & 31);

\_\_enable\_interrupt();

setupInterrupts();

**while** (1)

{

}

}

/\*\*\*\*| sevenSegment | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Brief: This function is used for deciding which number will be displayed

\* on the seven segment LED

\* param: uint32\_t intervalCnt

\* return:

\* N/A

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** **sevenSegment** (uint32\_t intervalCnt)

{

**if**(intervalCnt == 1)

{

//clear all lights

P4->OUT &= ~BIT3; //a

P4->OUT &= ~BIT4; //b

P4->OUT &= ~BIT5; //c

P4->OUT &= ~BIT6; //d

P4->OUT &= ~BIT7; //e

P5->OUT &= ~BIT4; //f

P5->OUT &= ~BIT5; //g

//turn on lights for 0

P4->OUT |= BIT3; //a

P4->OUT |= BIT4; //b

P4->OUT |= BIT5; //c

P4->OUT |= BIT6; //d

P4->OUT |= BIT7; //e

P5->OUT |= BIT4; //f

}

**if**(intervalCnt == 2)

{

//clear all lights

P4->OUT &= ~BIT3; //a

P4->OUT &= ~BIT4; //b

P4->OUT &= ~BIT5; //c

P4->OUT &= ~BIT6; //d

P4->OUT &= ~BIT7; //e

P5->OUT &= ~BIT4; //f

P5->OUT &= ~BIT5; //g

//turn on lights for 1

P4->OUT |= BIT4; //b

P4->OUT |= BIT5; //c

}

**if**(intervalCnt == 3)

{

//clear all lights

P4->OUT &= ~BIT3; //a

P4->OUT &= ~BIT4; //b

P4->OUT &= ~BIT5; //c

P4->OUT &= ~BIT6; //d

P4->OUT &= ~BIT7; //e

P5->OUT &= ~BIT4; //f

P5->OUT &= ~BIT5; //g

//turn on lights for 2

P4->OUT |= BIT3; //a

P4->OUT |= BIT4; //b

P4->OUT |= BIT6; //d

P4->OUT |= BIT7; //e

P5->OUT |= BIT5; //g

}

**if**(intervalCnt == 4)

{

//clear all lights

P4->OUT &= ~BIT3; //a

P4->OUT &= ~BIT4; //b

P4->OUT &= ~BIT5; //c

P4->OUT &= ~BIT6; //d

P4->OUT &= ~BIT7; //e

P5->OUT &= ~BIT4; //f

P5->OUT &= ~BIT5; //g

//turn on lights for 3

P4->OUT |= BIT3; //a

P4->OUT |= BIT4; //b

P4->OUT |= BIT5; //c

P4->OUT |= BIT6; //d

P5->OUT |= BIT5; //g

}

**if**(intervalCnt == 5)

{

//clear all lights

P4->OUT &= ~BIT3; //a

P4->OUT &= ~BIT4; //b

P4->OUT &= ~BIT5; //c

P4->OUT &= ~BIT6; //d

P4->OUT &= ~BIT7; //e

P5->OUT &= ~BIT4; //f

P5->OUT &= ~BIT5; //g

//turn on lights for 4

P4->OUT |= BIT4; //b

P4->OUT |= BIT5; //c

P5->OUT |= BIT4; //f

P5->OUT |= BIT5; //g

}

**if**(intervalCnt == 6)

{

//clear all lights

P4->OUT &= ~BIT3; //a

P4->OUT &= ~BIT4; //b

P4->OUT &= ~BIT5; //c

P4->OUT &= ~BIT6; //d

P4->OUT &= ~BIT7; //e

P5->OUT &= ~BIT4; //f

P5->OUT &= ~BIT5; //g

//turn on lights for 5

P4->OUT |= BIT3; //a

P4->OUT |= BIT5; //c

P4->OUT |= BIT6; //d

P5->OUT |= BIT4; //f

P5->OUT |= BIT5; //g

}

**if**(intervalCnt == 7)

{

//clear all lights

P4->OUT &= ~BIT3; //a

P4->OUT &= ~BIT4; //b

P4->OUT &= ~BIT5; //c

P4->OUT &= ~BIT6; //d

P4->OUT &= ~BIT7; //e

P5->OUT &= ~BIT4; //f

P5->OUT &= ~BIT5; //g

//turn on lights for 6

P4->OUT |= BIT3; //a

P4->OUT |= BIT5; //c

P4->OUT |= BIT6; //d

P4->OUT |= BIT7; //e

P5->OUT |= BIT4; //f

P5->OUT |= BIT5; //g

}

**if**(intervalCnt == 8)

{

//clear all lights

P4->OUT &= ~BIT3; //a

P4->OUT &= ~BIT4; //b

P4->OUT &= ~BIT5; //c

P4->OUT &= ~BIT6; //d

P4->OUT &= ~BIT7; //e

P5->OUT &= ~BIT4; //f

P5->OUT &= ~BIT5; //g

//turn on lights for 7

P4->OUT |= BIT3; //a

P4->OUT |= BIT4; //b

P4->OUT |= BIT5; //c

}

**if**(intervalCnt == 9)

{

//clear all lights

P4->OUT &= ~BIT3; //a

P4->OUT &= ~BIT4; //b

P4->OUT &= ~BIT5; //c

P4->OUT &= ~BIT6; //d

P4->OUT &= ~BIT7; //e

P5->OUT &= ~BIT4; //f

P5->OUT &= ~BIT5; //g

//turn on lights for 8

P4->OUT |= BIT3; //a

P4->OUT |= BIT4; //b

P4->OUT |= BIT5; //c

P4->OUT |= BIT6; //d

P4->OUT |= BIT7; //e

P5->OUT |= BIT4; //f

P5->OUT |= BIT5; //g

}

**if**(intervalCnt == 10)

{

//clear all lights

P4->OUT &= ~BIT3; //a

P4->OUT &= ~BIT4; //b

P4->OUT &= ~BIT5; //c

P4->OUT &= ~BIT6; //d

P4->OUT &= ~BIT7; //e

P5->OUT &= ~BIT4; //f

P5->OUT &= ~BIT5; //g

//turn on lights for 9

P4->OUT |= BIT3; //a

P4->OUT |= BIT4; //b

P4->OUT |= BIT5; //c

P5->OUT |= BIT4; //f

P5->OUT |= BIT5; //g

}

}

/\*\*\*\*| pin\_init | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Brief: This function is used for the initialization

\* of the pins for the 7 segment LED

\* param:

\* (void) data: N/A

\* return:

\* N/A

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** **pin\_init** (**void**)

{

//a

P4->SEL1 &= ~BIT3; // configure P4.3 as simple I/O

P4->SEL0 &= ~BIT3;

P4->DIR |= BIT3; // P4.3 set as output pin \*/

//b

P4->SEL1 &= ~BIT4; // configure P4.4 as simple I/O

P4->SEL0 &= ~BIT4;

P4->DIR |= BIT4; // P4.4 set as output pin \*/

//c

P4->SEL1 &= ~BIT5; // configure P4.5 as simple I/O

P4->SEL0 &= ~BIT5;

P4->DIR |= BIT5; // P4.5 set as output pin \*/

//d

P4->SEL1 &= ~BIT6; // configure P4.6 as simple I/O

P4->SEL0 &= ~BIT6;

P4->DIR |= BIT6; // P4.6 set as output pin \*/

//e

P4->SEL1 &= ~BIT7; // configure P4.7 as simple I/O

P4->SEL0 &= ~BIT7;

P4->DIR |= BIT7; // P4.7 set as output pin \*/

//f

P5->SEL1 &= ~BIT4; // configure P5.4 as simple I/O

P5->SEL0 &= ~BIT4;

P5->DIR |= BIT4; // P5.4 set as output pin \*/

//g

P5->SEL1 &= ~BIT5; // configure P5.5 as simple I/O

P5->SEL0 &= ~BIT5;

P5->DIR |= BIT5; // P5.5 set as output pin \*/

}

/\*\*\*\*| setupInterrupts | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Brief: This function is used for the initialization

\* of the buttons in interrupt form

\* param:

\* (void) data: N/A

\* return:

\* N/A

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** **setupInterrupts**()

{

// Blue

P1->SEL1 &= ~BIT6; // configure P1.6 as simple I/O

P1->SEL0 &= ~BIT6;

P1->DIR &= ~BIT6; // P1.6 set as output pin \*/

P1->REN |= BIT6;

P1->OUT |= BIT6;

P1->IES |= BIT6;

P1->IE |= BIT6;

P1->IFG &= ~BIT6;

// Red

P1->SEL1 &= ~BIT7; // configure P1.7 as simple I/O

P1->SEL0 &= ~BIT7;

P1->DIR &= ~BIT7; // P1.7 set as output pin \*/

P1->REN |= BIT7;

P1->OUT |= BIT7;

P1->IES |= BIT7;

P1->IE |= BIT7;

P1->IFG &= ~BIT7;

}

/\*\*\*\*| PORT1\_IRQHandler | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Brief: the handler for the port 1 interrupts.

\*

\* param:

\* (void) data: N/A

\* return:

\* N/A

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** **PORT1\_IRQHandler** ()

{

**if**(P1->IFG & BIT6)

{

intervalCnt++;

**if**(intervalCnt ==11)

{

intervalCnt =0;

}

sevenSegment(intervalCnt);

SysTick\_delay(750);

}

P1->IFG &= ~BIT6;

**if**(P1->IFG & BIT7)

{

intervalCnt--;

**if**(intervalCnt == 0)

{

intervalCnt = 10;

}

sevenSegment(intervalCnt);

SysTick\_delay(750);

}

P1->IFG &= ~BIT7;

}

/\*\*\*\*| SysTick\_Init | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Brief: This function is used for the initialization

\* of the SysTick timer

\* param:

\* (void) data: N/A

\* return:

\* N/A

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

{ //initialization of systic timer

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

SysTick->LOAD = 3000000; // 1 sec reload value

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

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

}

/\*\*\*\*| SysTick\_delay | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Brief: This function sets the SysTick timer to be used as a

\* delay when called

\* param:

\* (uint16\_t) data: one integer worth of data input that

\* determines the duration of the delay

\* return:

\* N/A

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**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

}