CPE301 – SPRING 2019

Design Assignment 2, Part B

Student Name: James Skelly

Student #: 2000945485

Student Email: skellj1@unlv.nevada.edu

Primary Github address: https://github.com/skellj1/submission\_da

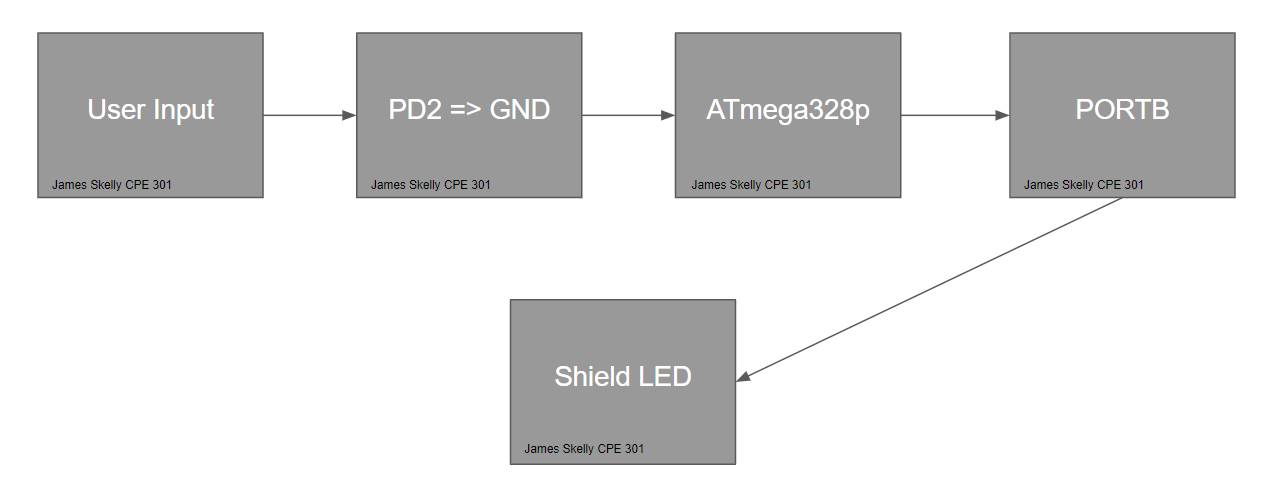
Directory: skellj1/submission\_da

Submit the following for all Labs:

1. In the document, for each task submit the modified or included code (only) with highlights and justifications of the modifications. Also, include the comments.
2. Use the previously create a Github repository with a random name (no CPE/301, Lastname, Firstname). Place all labs under the root folder ESD301/DA, sub-folder named LABXX, with one document and one video link file for each lab, place modified asm/c files named as LabXX-TYY.asm/c.
3. If multiple asm/c files or other libraries are used, create a folder LabXX-TYY and place these files inside the folder.
4. The folder should have a) Word document (see template), b) source code file(s) and other include files, c) text file with youtube video links (see template).

1. **COMPONENTS LIST AND CONNECTION BLOCK DIAGRAM w/ PINS**

* Components used include Atmel Studio 7, Multifunction Shield for Arduino, Xplained Mini, Tektronix Oscilloscope and Scope Probe, jumper wire, USB cable, iPhone for video recording.



1. **INITIAL/MODIFIED/DEVELOPED CODE OF TASK 1/A**

**Task 2 C Code**

/\* DA2A\_Task\_2\_C.c

\* Created: 3/1/2019 3:15:26 PM

\* Author : James Skelly \*/

// 2. Connect a switch to PORTC.2 to poll for an event to turn on the LED

// at PORTB.2 for 1.250 seconds after the event.

#define *F\_CPU* 16000000UL // Set frequency of CPU to 16 MHz for delay function

#include <avr/io.h> // Including AVR and Delay header libraries

#include <util/delay.h>

int main (void)

{

DDRB |= (1<<2); // Sets DDRB pin 2 to output mode by performing bitwise

// OR with a 1 shifted two places to the left, and

// and the value previously stored in DDRB. The

// result will be stored in DDRB.

PORTB |= (1<<2); // Sets PORTB pin 2 to high by performing bitwise

// OR with a 1 shifted two places to the left, and

// and the value previously stored in PORTB. The

// result will be stored in PORTB. This will initially

// turn the LED off.

DDRC &= (0 << 2); // Sets DDRC pin 2 to input mode by performing bitwise A

// AND with a 0 shifted two places to the left, and the

// value previously stored in DDRC. The result will

// be stored in DDRC.

PORTC |= (1 << 2); // Sets PORTC pin 2 to high by performing bitwise

// OR with a 1 shifted two places to the left, and

// and the value previously stored in PORTC. The

// result will be stored in PORTC. This enables the

// pull-up resistor.

while (1)

{

if (!(PINC & (1<<PINC2))) // Check PINC1 for a value of 1,

// complemented to check for a // zero instead.

{

PORTB &= ~(1<<2); // Turn on LED by setting PORTB.2 to 0.

*\_delay\_ms*(1250); // Call for a delay of 1.25 seconds

}

else

PORTB |= (1<<2); // Turn off the LED by setting PORTB.2 to 1.

}

return 0;

}

**Task 2 Assembly Code**

; DA2A\_2\_ASM.asm

; Created: 3/1/2019 3:14:30 PM

; Author : James Skelly

; 2. Connect a switch to PORTC.2 to poll for an event to turn on the LED

; at PORTB.2 for 1.250 seconds after the event.

.ORG 0

LDI R16, 0x04 ; loads 0x04 into register 16

LDI R17, 0x00 ; loads 0x04 into register 17

OUT DDRB, R16 ; sets data direction register PORTB.2 to output

OUT DDRC, R17 ; sets data direction register PORTC to all inputs

OUT PORTB,R16 ; sets PORTB.2 high

OUT PORTC, R16 ; sets port C, pin 2 high

NOP

MAIN:

IN R18, PINC ; reads input on pin C into register 20

COM R18 ; invert the bits inputted to pin c

ANDI R18, 0x04 ; bitwise AND the value in register 20 with 0x04

CPI R18, 0x04 ; compare the value in r20 with 0x04

BRNE MAIN

LIGHT\_ON:

LDI R21, 0xFB ; 11111011 INTO R21

OUT PORTB, R21 ; OUTPUT PORTB.2 LED 4 turns on

rcall delay\_1s ; call for delay of 1.25 s

rcall delay\_100ms ;

rcall delay\_100ms ;

rcall delay\_50ms ; end of 1.25 s delay

OUT PORTB, R16 ; turn LED off

RJMP MAIN ; jump back to top of code

**Developed Delay Subroutine for 1ms Delay**

delay\_1ms:

push r16 // push r16 to stack

push r18 // push r18 to stack

ldi r16, 255 // load 255 into r16

ldi r18, 6 // load 6 into r18

delay\_1b: // label for outer loop

delay\_1a: // label for inner loop

nop // no operation, takes 1 cycle

nop //

nop //

nop //

nop //

nop //

nop //

dec r16 // decrement the value in r16

brne delay\_1a // if the value in r16 is not 0,

// stay in the loop

dec r18 // decrement the value in r18

brne delay\_1b // if the value in r18 is not 0,

// stay in the loop

pop r18 // pop r18 back from the stack

pop r16 // pop r16 back from the stack

ret // return to line below rcall

**The code above was created to generate a subroutine that would last 1 ms. This subroutine was used as the building block for larger delays in assembly throughout this assignment. Calculating the delay generated by this subroutine was conducted by counting the number of instructions per loop and multiplying that number by the time taken to complete one cycle at 16 MHz, or 62.5 nanoseconds. Calculations led to a final generated delay of roughly 1.005ms.**

1. **DEVELOPED MODIFIED CODE OF TASK 2/A from TASK 1/A**

**C CODE DA2B**

/\*

\* **C Code DA2B**

\*

\* Created: 3/8/2019 4:33:30 PM

\* Author : James Skelly

\*/

#define *F\_CPU* 16000000UL // Set frequency of CPU for delay function

#include <avr/io.h> // AVR input/output header

#include <avr/interrupt.h> // AVR interrupt header

#include <util/delay.h> // delay header

int main(void)

{

DDRB |= (1<<2); // PB2 is an output

PORTB |= (1<<2); // PB2 is high

DDRD &= (0<<2); // PD2 is an input

PORTD |= (1<<2); // PD2 is high

EICRA = 0x0; // make INT0 respond to low level shift

EIMSK = (1<<INT0); // enable external interrupt 0

sei (); // globally enable interrupt

while (1); // stay here until interrupt is activated

}

ISR (INT0\_vect) // Interrupt subroutine for ext. int. 0

{

PORTB &= ~(1<<2); // Turn on LED by setting PORTB.2 to 0

*\_delay\_ms*(1250); // Call for a delay of 1.25 seconds

PORTB |= (1<<2); // Turn off LED by setting PORTB.2 to 1

}

**ASSEMBLY CODE DA2B**

; **ASSEMBLY CODE DA2B**

;

; Created: 3/8/2019 4:34:23 PM

; Author : James Skelly

;

.ORG 0 ; program memory origin is 0

JMP MAIN

.ORG 0x02 ; interrupt subroutine origin is 2

JMP EX0\_ISR

MAIN:

LDI R20, HIGH(RAMEND) ; initialize stack

OUT SPH, R20

LDI R20, LOW(RAMEND)

OUT SPL, R20

LDI R16, 0xFF ; 11111111 into R16

OUT PORTB, R16 ; turn LED off to start

LDI R20, 0x00 ; load 0x00 into reg. 20

STS EICRA, R20 ; set interrupt to activate on low level shift

SBI DDRB, 2 ; set PORTB.2 as output

SBI PORTD, 2 ; set PORTD.2 high

LDI R20, 1<<INT0 ; load interrupt 0 shifted left once into r20

OUT EIMSK, R20 ; set external interrupt mask bit zero high

SEI ; global interrupt set

HERE: JMP HERE ; wait for interrupt to occur

EX0\_ISR: ; external interrupt 0 subroutine

LDI R21, 0xFB ; 11111011 INTO R21

LDI R16, 0xFF ; 11111111 into R16

OUT PORTB, R21 ; OUTPUT PORTB.2 LED 4 turns on

rcall delay\_1s ; call for delay of 1.25 s from delay subroutines

rcall delay\_100ms ;

rcall delay\_100ms ;

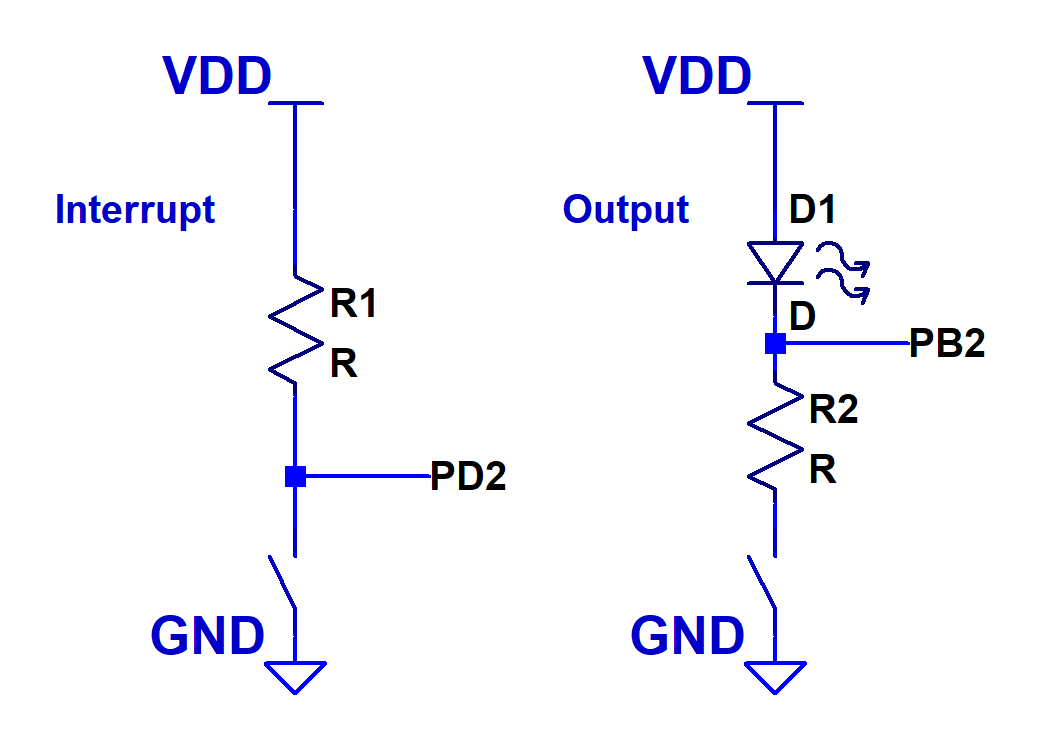
rcall delay\_50ms ; end of 1.25 s delay

OUT PORTB, R16 ; turn LED off

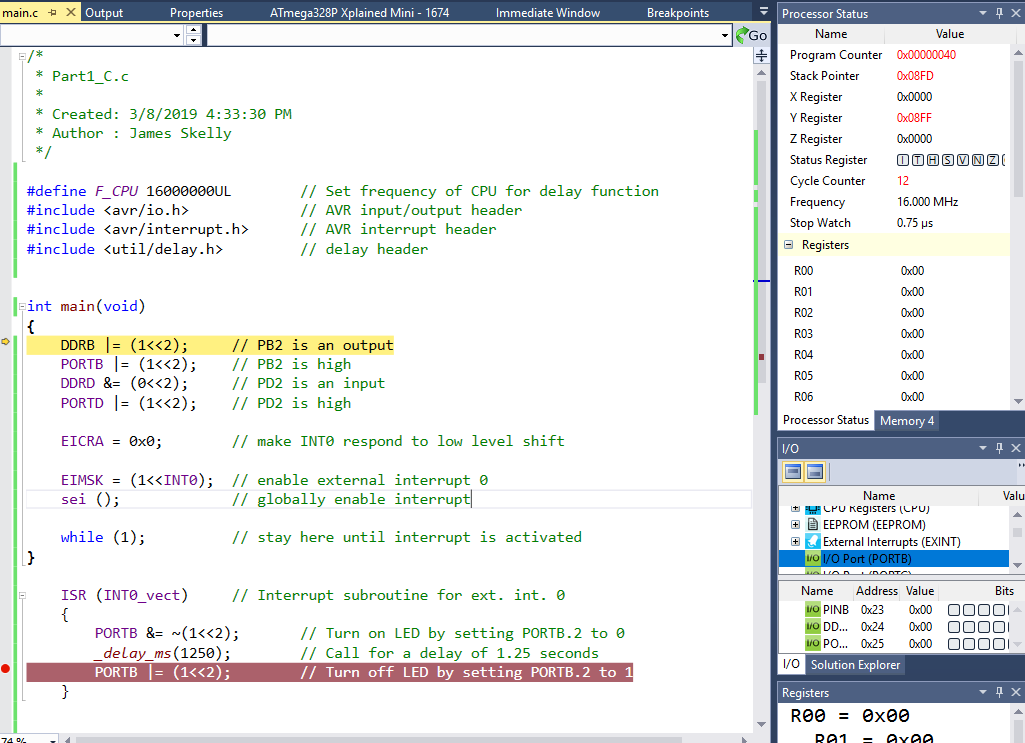
RETI ; jump back to address following interrupt

1. **SCHEMATICS**

**The schematic below was drafted in LTspice. The interrupt portion shows that when PD2 is grounded, the interrupt activates. When the interrupt is activated, the GND switch on the output closes, voltage is dropped across the diode and it is forward biased, making it light up on PB2. This is where we probe our output signal for oscilloscope analysis.**



1. **SCREENSHOTS OF EACH TASK OUTPUT (ATMEL STUDIO OUTPUT)**



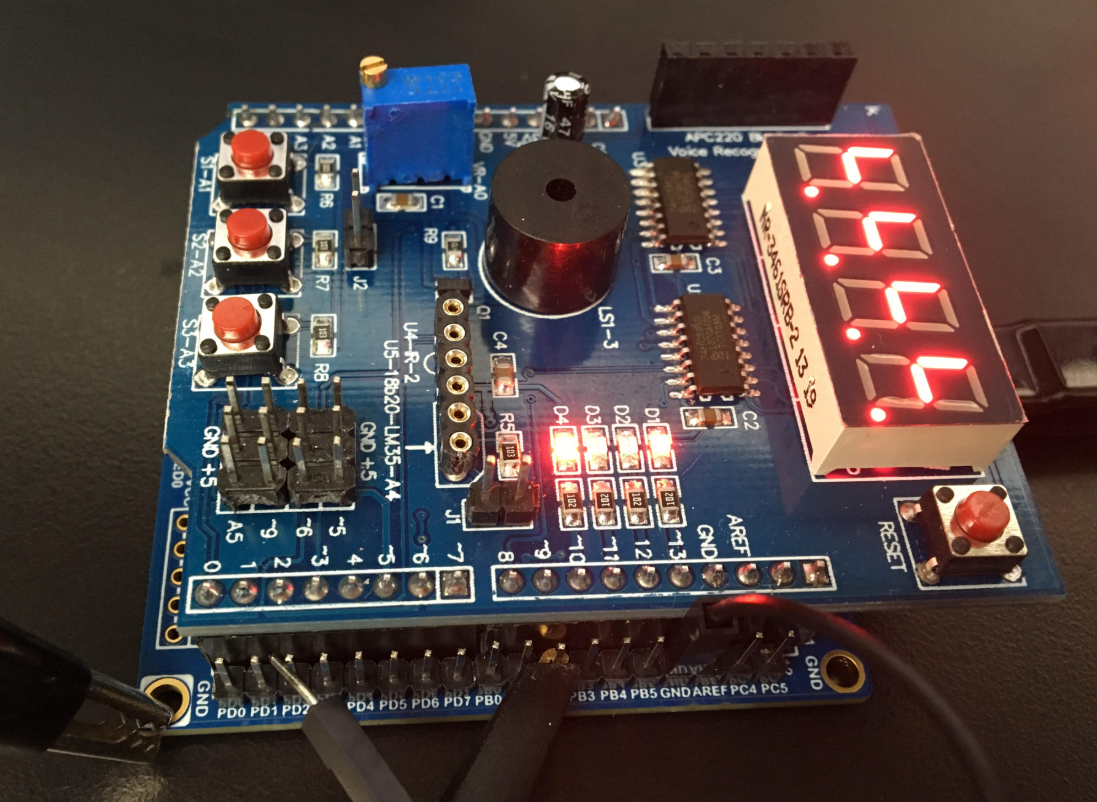
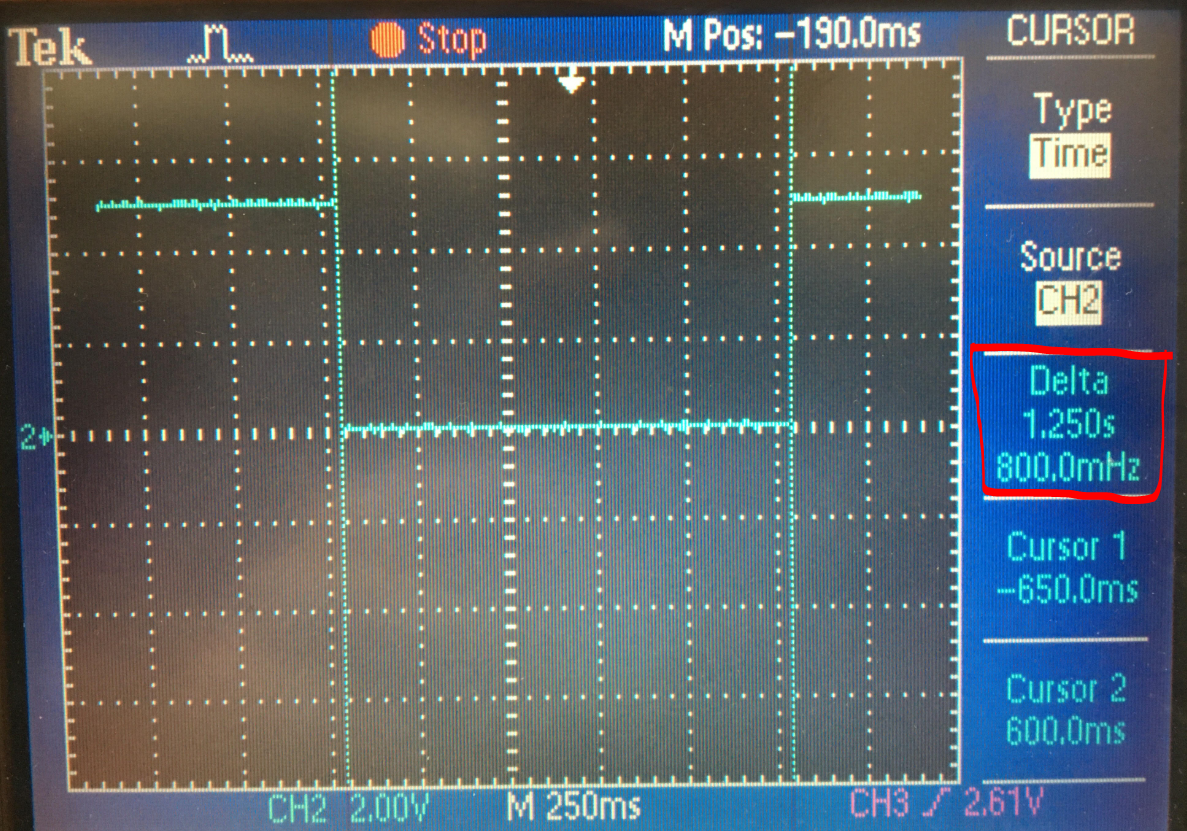


In the output snips above, we see that the assembly code took around 1.2 seconds to run up until the return from interrupt instruction. This is exactly what we expect, being that we designed the program to output our pulse for 1.25s after PD2 is grounded.

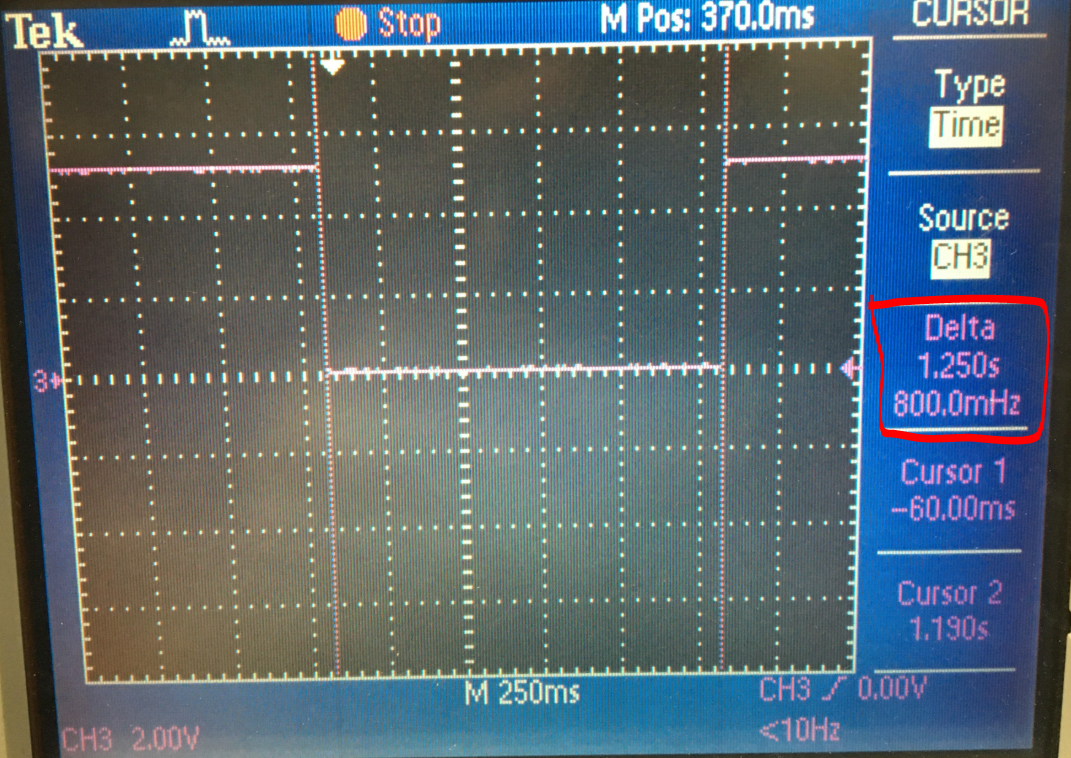
In the C code, only a few instructions are executed, and the simulator output shows that only a few microseconds were needed to execute the same function executed in Assembly.

1. **SCREENSHOT OF EACH DEMO (BOARD SETUP)**

**C Code Board Setup and Oscilloscope Output**

**Assembly Code Board Setup and Oscilloscope Output**

The snips above show the board setup with the oscilloscope probe connected to pin PB2, the ground of the scope probe connected to the Xplained Mini ground, and the female end of the jumper cable connected to ground as well. The jumper cable was used to ground pin PD2 to activate the interrupt, and the oscilloscope measured the length in time of the pulse that was generated. In both cases above, for assembly in purple and C in blue, 1.250s was achieved.

1. **VIDEO LINKS OF EACH DEMO**

**DA2B Assembly Code Video Link**

[**https://www.youtube.com/watch?v=CCTwgE5nfZA**](https://www.youtube.com/watch?v=CCTwgE5nfZA)

**DA2B C Code Video Link**

[**https://www.youtube.com/watch?v=\_S2v3Q2Lhb4**](https://www.youtube.com/watch?v=_S2v3Q2Lhb4)

1. **GITHUB LINK OF THIS DA**

<https://github.com/skellj1/submission_da>

**Student Academic Misconduct Policy**

<http://studentconduct.unlv.edu/misconduct/policy.html>

“This assignment submission is my own, original work”.

James W. Skelly