CPE301 – SPRING 2019

Design Assignment C

Student Name: Robert Sander

Student #: 5002102412

Student Email: sander1@unlv.nevada.edu

Primary Github address: <https://github.com/sanderUNLV/submission_DA.git>

Youtube link:

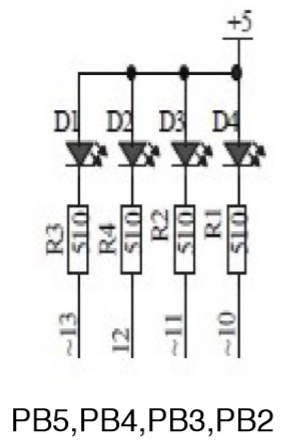
<https://youtu.be/E6ZE2szqnnI>

<https://youtu.be/teJOJcMTQrs>

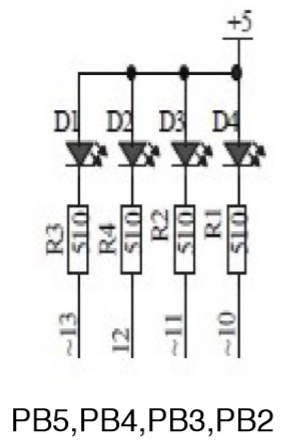
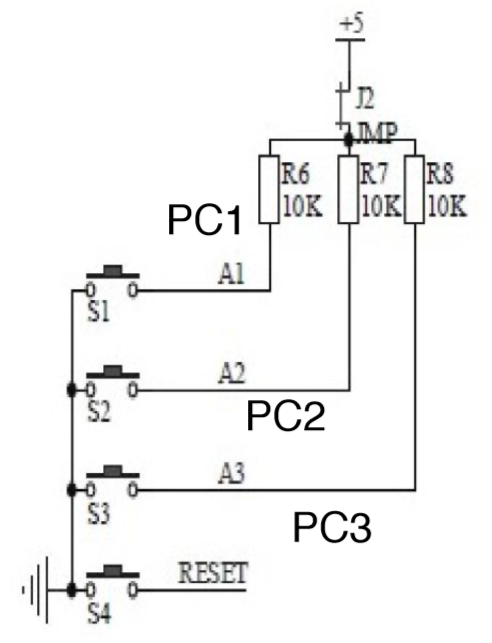
1. **COMPONENTS LIST AND CONNECTION BLOCK DIAGRAM w/ PINS**

Block diagram with pins used in the Atmega328P

T1a:



T2a:



1. **INITIAL/MODIFIED/DEVELOPED CODE OF TASK 1.1**

// DA2C\_T.1.1\_C

// Author : Robert Sander

#include <avr/io.h>

volatile int counter = 0;

int main(void)

{

DDRB = 0b00111100; //PB2-5 as output

PORTB = \_BV(PINB2)|\_BV(PINB3)|\_BV(PINB4)|\_BV(PINB5); // TURN OFF ALL LEDS AT FIRST

TCCR0A = 0; // Normal Operation

TCCR0B |= (1<<CS01)|(1<<CS00); // set prescaler to 64 and start the timer

TCNT0=0x00; // start the timer

while (1)

{

// wait for the overflow event

while ((TIFR0 & 0x01) == 0);

{

counter++; // Increment counter, TCNT0 goes back to 0 when the overflow flag is set

if (counter >= 425) // (((435ms/0.0000625ms)/64)/256)=424.8, this is how many times we have to count the overflow for 435ms

{

PORTB = ~(1 << 2); // Turn on LED

}

if (counter >= 708) // (((290ms/0.0000625ms)/64)/256)=283.2, this is how many times we have to count the overflow for 290ms

{

PORTB = 0x3C; // Turn off LED

counter = 0; // RESET COUNTER

}

TIFR0=0x01; // reset the overflow flag

TCNT0 = 0; // reset timer

}

}

}

1. **INITIAL/MODIFIED/DEVELOPED CODE OF TASK 1.2**

// DA2C\_T.1.2\_C

// Author : Robert Sander

#include <avr/io.h>

volatile int counter = 0;

int main() // what does having/not having "void" in here do?

{

DDRB = 0b00111100; //PB2-5 as output

PORTB = \_BV(PINB2)|\_BV(PINB3)|\_BV(PINB4)|\_BV(PINB5); // TURN OFF ALL LEDS AT FIRST

// SET PORTC FOR INPUT //PC.0 IS CONNECTED TO POTENTIOMETER

TCCR0A = 0; // Normal Operation

TCCR0B |= (1<<CS02)|(0<<CS01)|(0<<CS00); // set prescaler to 256 and start the timer

TCNT0 = 0x00; // start the timer

while (1)

{

if (!(PINC & (1 << PINC2))) // Polling for PC.2

{

TCNT0= 0x00; // reset timer

counter = 0;

PORTB = ~(1 << 2); // Turn on LED PB.2

}

// wait for the overflow event

while ((TIFR0 & 0x01) == 0); // If overflow flag is set, break out

{

counter++; // Increment counter, TCNT0 goes back to 0 when the overflow flag is set

if (counter >= 280) // (((1250ms/0.0000625ms)/256)/256)=305.2, this is how many times we have to count the overflow for 290ms

{

PORTB = 0x3C; // Turn off LED

counter = 0; // RESET COUNTER

}

TIFR0 = 0x01; // reset the overflow flag

}

}

}

1. **INITIAL/MODIFIED/DEVELOPED CODE OF TASK 2.1**

// DA2C\_T.2.1\_C

// Author : Robert Sander

#include <avr/io.h>

#include <avr/interrupt.h>

/\*

Needs to be global in scope so that the intterupt service routine has access to it.

It cannot be passed to the routine. It also must have the volatile modifier because

the compiler has no idea when the ISR will activate and change the variable.

This prevents the complier from making assumptions as to the value of the variable.

\*/

volatile int counter = 0;

int main()

{

DDRB = 0b00111100; // PB2-5 as output

PORTB = 0x3C; // Turn off LED PB.2-5

TIMSK0 |= (1 << TOIE0);// Enable Timer/Counter0 Overflow Interrupt Enable

TCCR0A = 0; // Normal Operatio

TCCR0B |= (1 << CS01) | (1 << CS00); // 64 prescaler

sei(); //enable interrupts

TCNT0 =0x00; // initial value

while (1)

{

//STAY HERE INDEFINITELY

}

}

ISR (TIMER0\_OVF\_vect) // timer0 overflow interrupt

{

counter++; // Increment the interrupt counter by 1

if (counter >= 425)

{

PORTB = ~(1 << 2); // Turn on LED PB.2

if (counter >= 708)

{

PORTB = 0x3C; // Turn off LED PB.2-5

counter = 0; // Reset counter

}

}

}

1. **INITIAL/MODIFIED/DEVELOPED CODE OF TASK 2.2**

// DA2C\_T.2.2\_C

// Author : Robert Sander

#include <avr/io.h>

#include <avr/interrupt.h>

/\*

Needs to be global in scope so that the intterupt service routine has access to it.

It cannot be passed to the routine. It also must have the volatile modifier because

the compiler has no idea when the ISR will activate and change the variable.

This prevents the complier from making assumptions as to the value of the variable.

\*/

volatile int counter = 0;

int main()

{

DDRB = 0b00111100; // PB2-5 as output

PORTB = 0x3C; // Turn off LED PB.2-5

TCCR0A = 0; // Normal Operation

TCCR0B |= (1<<CS02)|(0<<CS01)|(0<<CS00); // 256 prescaler

TIMSK0 |= (1 << TOIE0);// Enable Timer/Counter0 Overflow Interrupt Enable

sei(); //enable interrupts

TCNT0 = 0; // initial value

while (1)

{

if (!(PINC & (1<<PINC2))) //CONSTANTLY POLL TO SEE CONDITION FOR PINC, CHECK TO SEE IF THE PIN IS HIGH, BUTTON IS PUSHED

{

TCNT0 = 0; // Reset Timer

counter = 0; // Reset interrupt counter

PORTB = ~(1 << 2); // SET PB.2 TO 0 TO TURN THE LED ON, REVERSE LOGIC

}

}

}

ISR (TIMER0\_OVF\_vect) // timer0 overflow interrupt

{

counter++; // Increment the interrupt counter by 1

if (counter >= 280) // This should be 305 but 280 is more accurate for some reason

{

PORTB = 0x3C; // Turn off LED PB.2-5

counter = 0; // Reset interrupt counter

}

}

1. **INITIAL/MODIFIED/DEVELOPED CODE OF TASK 3.1**

// DA2C\_T.3.1\_C

// Author : Robert Sander

#include <avr/io.h>

#include <avr/interrupt.h>

/\*

Needs to be global in scope so that the intterupt service routine has access to it.

It cannot be passed to the routine. It also must have the volatile modifier because

the compiler has no idea when the ISR will activate and change the variable.

This prevents the complier from making assumptions as to the value of the variable.

\*/

volatile int counter = 0;

int main ()

{

DDRB = 0b00111100; //PB2-5 as output

PORTB = 0x3C; // Turn off LED PB.2-5

OCR0A = 255; //this value is compared with TCNT0, a match generates an output compare interrupt

TCCR0B |= (1<<CS01|1<<CS00); // 64 prescaler

TCCR0A |= (1<<WGM01); //CTC mode, internal clk, TOP=OCR0A

TIMSK0 = (1<<OCIE0A); //enable Timer0 compare match int.

sei (); //enable interrupts

while (1) //wait here

{

// Stay here for indefinitely

}

}

ISR (TIMER0\_COMPA\_vect) //ISR for Timer0 compare match

{

counter++; // Increment the interrupt counter by 1

if (counter >= 425)

{

PORTB = ~(1 << 2); // Turn on LED PB.2

if (counter >= 708)

{

PORTB = 0x3C; // Turn off LED PB.2-5

counter = 0; // Reset counter

}

}

}

1. **INITIAL/MODIFIED/DEVELOPED CODE OF TASK 3.2**

// DA2C\_T.3.2\_C

// Author : Robert Sander

#include <avr/io.h>

#include <avr/interrupt.h>

/\*

Needs to be global in scope so that the intterupt service routine has access to it.

It cannot be passed to the routine. It also must have the volatile modifier because

the compiler has no idea when the ISR will activate and change the variable.

This prevents the complier from making assumptions as to the value of the variable.

\*/

volatile int counter = 0;

int main ()

{

DDRB = 0b00111100; //PB2-5 as output

PORTB = 0x3C; // Turn off LED PB.2-5

OCR0A = 255; //this value is compared with TCNT0, a match generates an output compare interrupt

TCCR0A |= (1<<WGM01); //CTC mode, internal clk, TOP=OCR0A

TCCR0B |= (1<<CS02|0<<CS01|0<<CS00); // 256 prescaler

TIMSK0 = (1<<OCIE0A); //enable Timer0 compare match int.

sei (); //enable interrupts

TCNT0 = 0; // Initial value

while (1) //wait here

{

// Stay here for indefinitely

}

}

ISR (TIMER0\_COMPA\_vect) //ISR for Timer0 compare match

{

counter++; // Increment the interrupt counter by 1

if (!(PINC & (1<<PINC2))) //CONSTANTLY POLL TO SEE CONDITION FOR PINC

{

counter = 0;

TCNT0 = 0;

PORTB = ~(1 << 2); // Turn on LED PB.2

}

if (counter >= 280)

{

PORTB = 0x3C; // Turn off LED PB.2-5

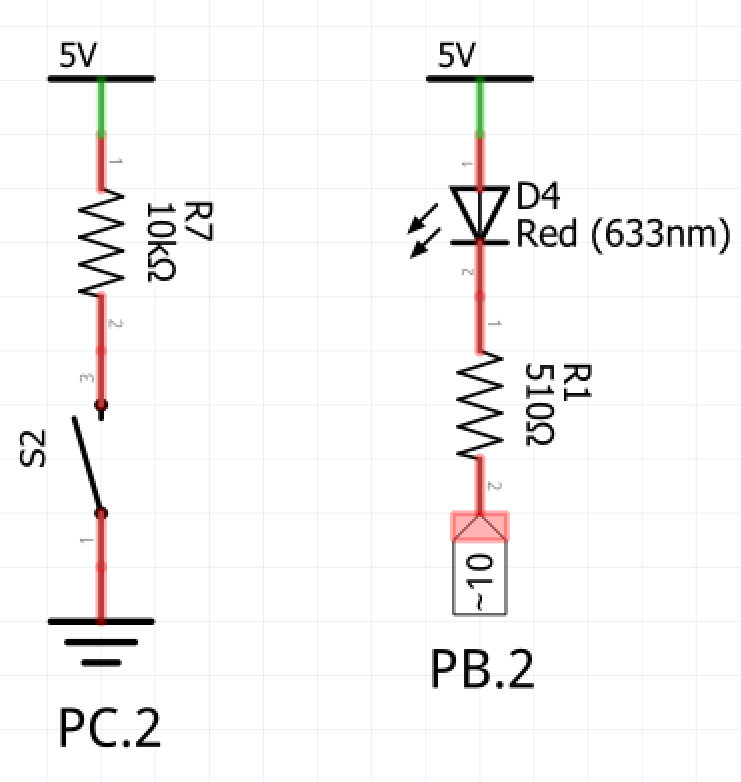
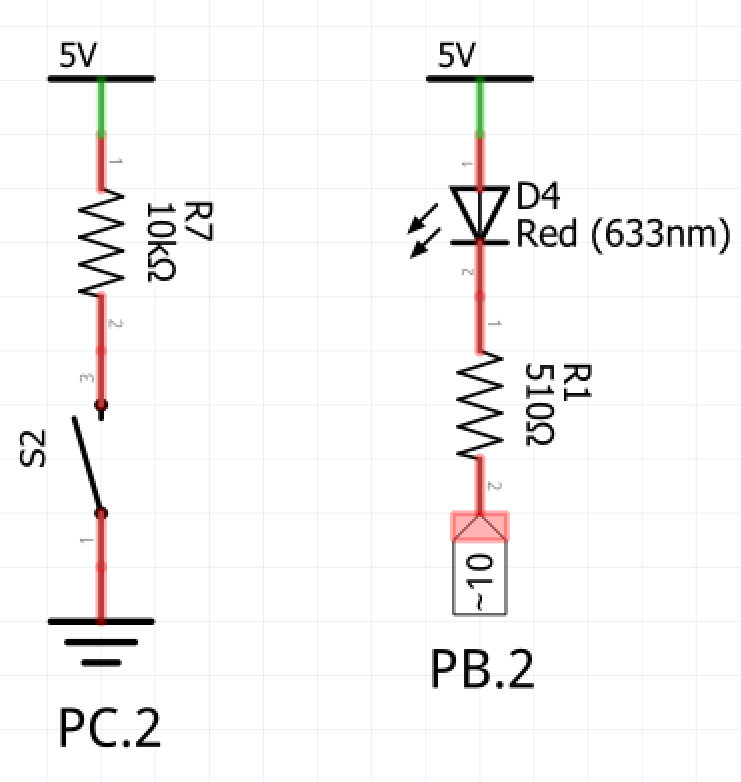
counter = 0; // Reset counter

}

}

1. **SCHEMATICS**

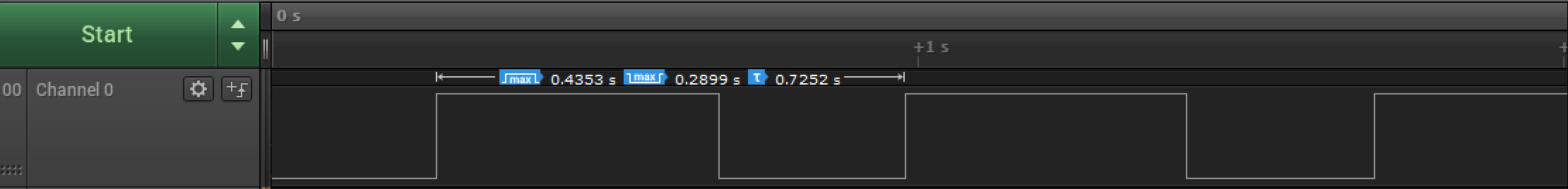
T1a T2a



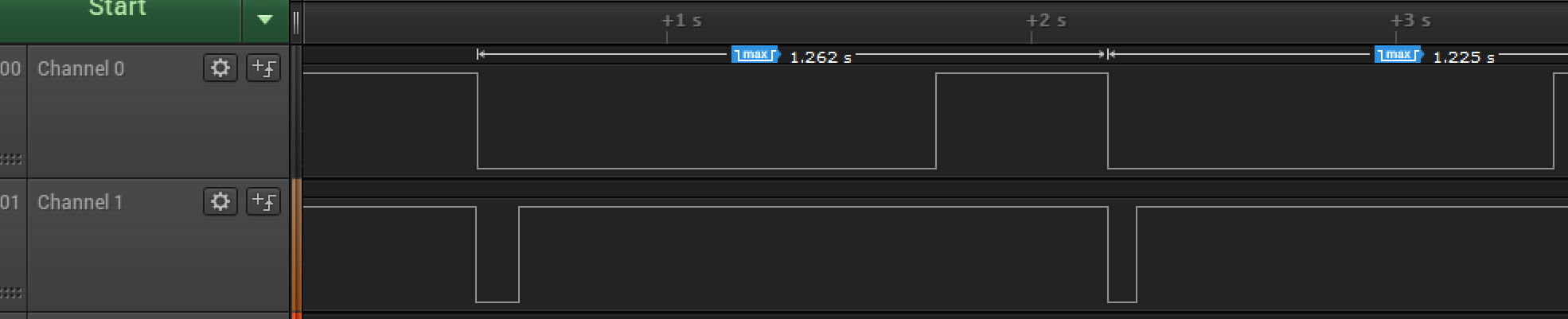
*Figure 2 – Task 2a schematic*

*Figure 1 – Task 1a schematic*

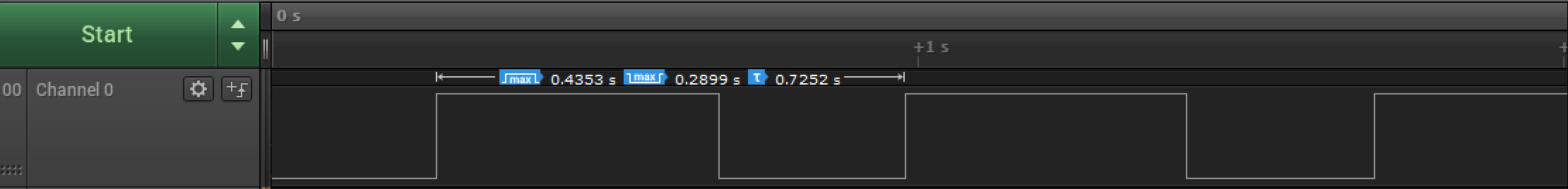
1. **SCREENSHOTS OF EACH TASK OUTPUT (ATMEL STUDIO OUTPUT)**



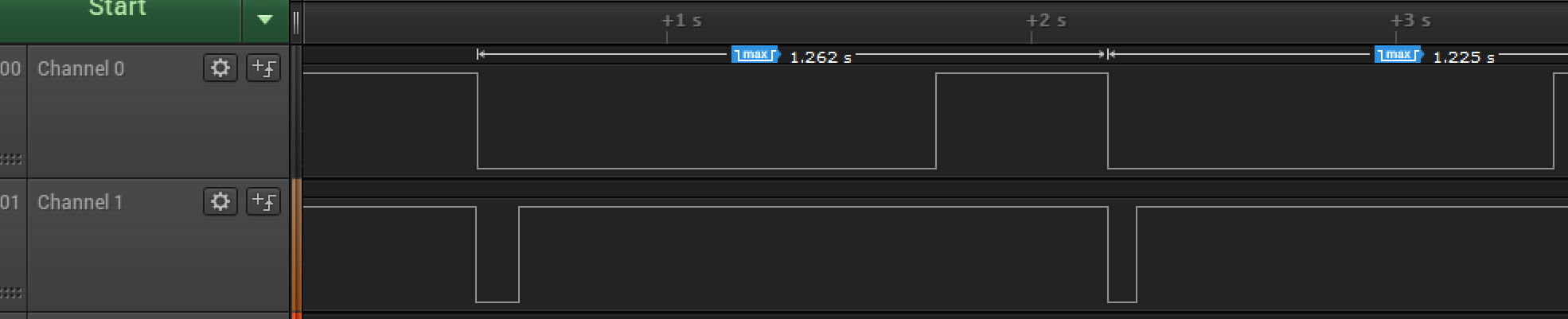
*Figure 3 – Duty Cycle and Period (C) Task 1.1*



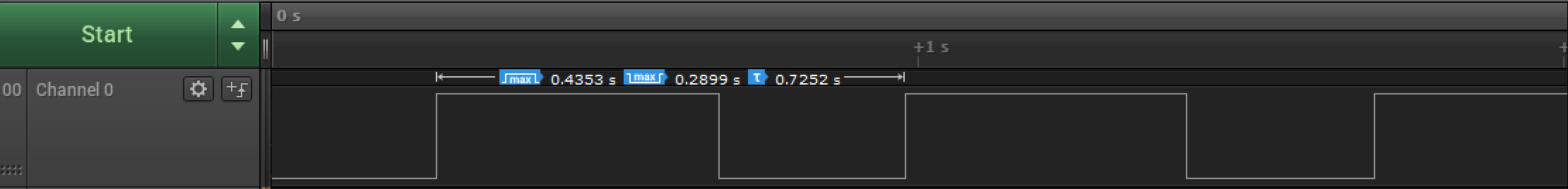
*Figure 4 – Duty Cycle and Period (C) Task 1.2*



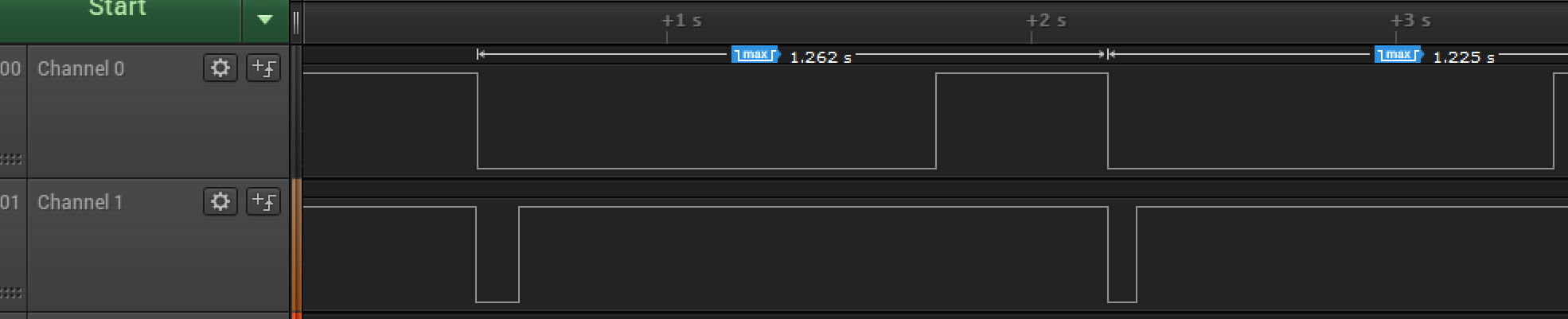
*Figure 5 – Duty Cycle and Period (C) Task 2.1*



*Figure 6 – Duty Cycle and Period (C) Task 2.2*

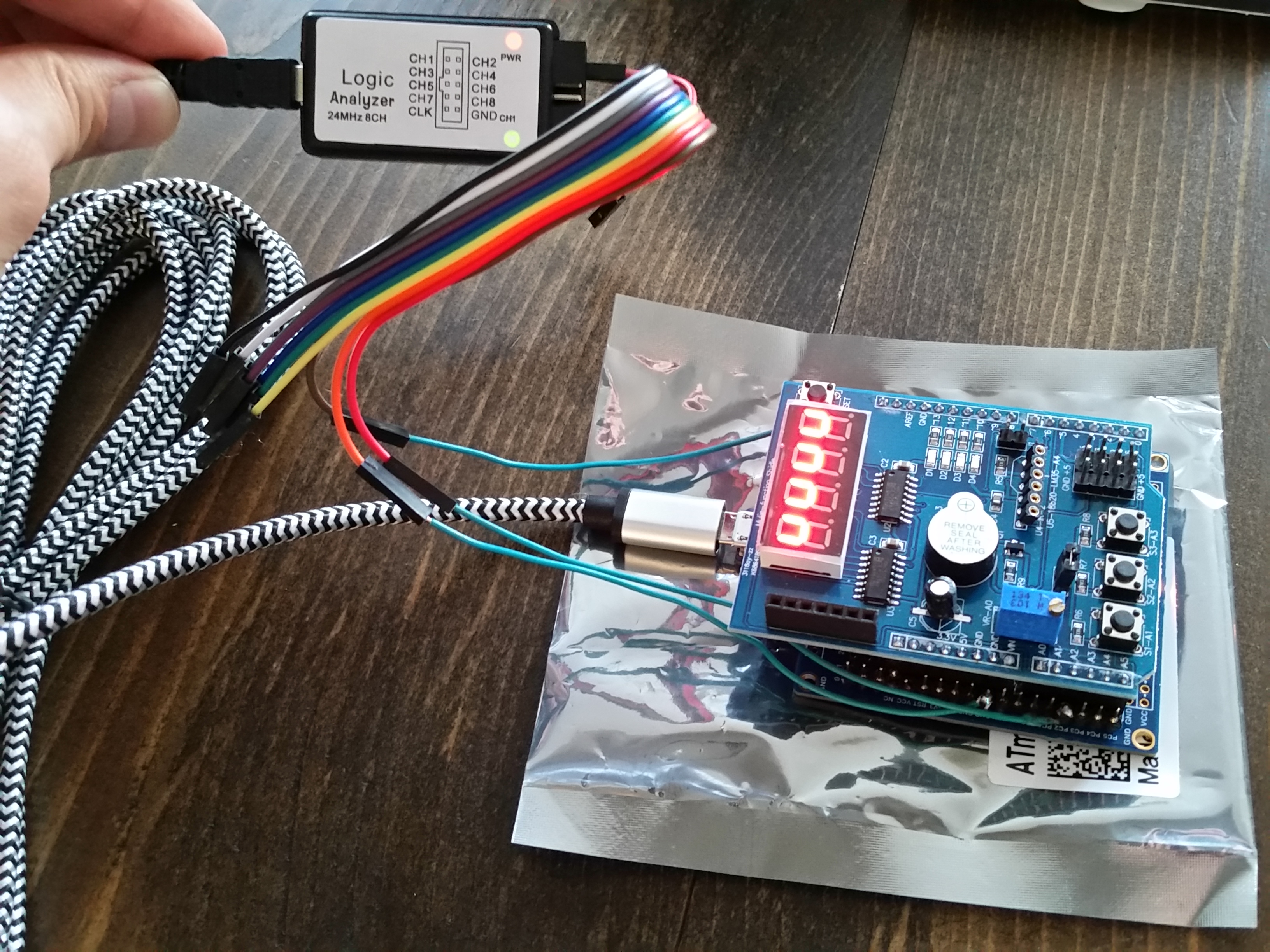


*Figure 7 – Duty Cycle and Period (C) Task 3.1*



*Figure 8 – Duty Cycle and Period (C) Task 3.2*

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

****

*Figure 9 – Xplained Mini w/ Multifunction Shield and Logic Analyzer*

1. **VIDEO LINKS OF EACH DEMO**

<https://youtu.be/E6ZE2szqnnI>

<https://youtu.be/teJOJcMTQrs>

1. **GITHUB LINK OF THIS DA**

<https://github.com/sanderUNLV/submission_DA.git>

**Student Academic Misconduct Policy**

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

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

-Robert Sander