Jeffrey Razon

**CPE301 – SPRING 2018**

# Design Assignment 02

**DO NOT REMOVE THIS PAGE DURING SUBMISSION:**

The student understands that all required components should be submitted in complete for grading of this assignment.

| NO | SUBMISSION ITEM | COMPLETED (Y/N) | MARKS (/MAX) |
|---|---|---|---|
| 1 | COMPONENTS LIST AND CONNECTION BLOCK DIAGRAM w/ PINS | | |
| 2. | INITIAL CODE OF TASK 1/A | | |
| 3. | INCREMENTAL / DIFFERENTIAL CODE OF TASK 2/B | | |
| 3. | INCREMENTAL / DIFFERENTIAL CODE OF TASK 3/C | | |
| 3. | INCREMENTAL / DIFFERENTIAL CODE OF TASK 4/D | | |
| 3. | INCREMENTAL / DIFFERENTIAL CODE OF TASK 5/E | | |
| 4. | SCHEMATICS | | |
| 5. | SCREENSHOTS OF EACH TASK OUTPUT | | |
| 5. | SCREENSHOT OF EACH DEMO | | |
| 6. | VIDEO LINKS OF EACH DEMO | | |
| 7. | GOOGLECODE LINK OF THE DA | | |
| | | | |
| | | | |

**Task 1/A**: Design a delay subroutine to generate a waveform on PORTB.2 with 50% DC and 0.5 sec period. (I was told that just having the LED light turn on/off every 0.25s would suffice.)

a) C code

```c
/*
    1) toggle LED light on
    2) delay 0.25s
    3) toggle LED light off
    4) delay 0.25s
    5) repeat
*/

#define F_CPU 8000000UL
#include <avr/io.h>
#include <util/delay.h>

#define LEDPORT PORTB
#define LED_PIN PORTB2
#define LED_DDR DDRB

int main(void)
{
    LED_DDR |= (1 << LED_PIN);  // Set LED to output
    LEDPORT &= ~(1 << LED_PIN); // Initial state of OFF

    while (1)   // Loops forever
    {
        LEDPORT |= (1 << LED_PIN);  // turn LED on
        _delay_ms(250); // 0.25 sec delay
        LEDPORT &= ~(1 << LED_PIN); // turn LED off
        _delay_ms(250); // 0.25 sec delay
    }
}
```

b) Assembly code

```
.org 0

    LDI R16, 4 ;needed to toggle LED
    SBI DDRB, 2 ;PB2 as output
    LDI R17,0  ;needed to toggle led

    ;OUT PORTB,R17
    LDI R20,5 ;to set prescaler
    STS TCCR1B,R20 ;Prescaler: 1024

begin:
    RCALL delay ;calling timer to wait for 0.25 sec
    EOR R17,R16 ;XOR to toogle led
    OUT PORTB,R17 ;display LED
    RJMP begin ;repeating i.e, while(1)
delay:
    LDS R29, TCNT1H ;loading upper bit of counter to R29
    LDS R28, TCNT1L ;loading lower bit of counter to R28
    CPI R28, 0xA1 ;comparing if lower is 0xA1
    BRSH body
    RJMP delay
body:
    CPI R29,0x07 ;comparing if higher is 0x07
    BRSH done ;if equal, branch to "done"
    RJMP delay ;if not, loop back
done:
    LDI R20,0x00
    STS TCNT1H,R20 ;resetting the counter to 0 for next round
    LDI R20,0x00
    STS TCNT1L,R20 ;resetting the counter to 0 for next round
    RET
```

c) Timing Proof
In /da02_screenshots/TimingProof folder, under the names:
Assembly code: *ASM_T1.PNG*
C code: *C_T1.PNG*

**Task 2/B**: Connect a switch to PORTD.2 (active high - turn on the pull up transistor) to poll for an event to turn on the led at PORTB.2 for 1 sec after the event. (There were no further instructions of what to do after, so I made the LED light toggle on/off 1 second after pressing the button.)

a) C code

```c
/*
    1) push button is initially off
    2) push button
    3) toggle LED light
*/

#define F_CPU 8000000UL
#include <avr/io.h>
#include <util/delay.h>

#define LEDPORT PORTB
#define LED_PIN PORTB2
#define LED_DDR DDRB

#define BUTTONPORT PORTD
#define BUTTONDDR DDRD
#define BUTTON_PIN PORTD2
#define BUTTONPIN PIND

short checkButton(); // prototype

int main(void)
{
    LED_DDR |= (1 << LED_PIN);  // Set LED to output
    LEDPORT &= ~(1 << LED_PIN); // Initial state of OFF

    BUTTONPORT |= (1 << BUTTON_PIN);    // pull-up

    while (1)   // Loops forever
    {
        short pressed = checkButton(); // branch to
        if(pressed) // if button is pressed,
        {
            LEDPORT |= (1 << LED_PIN);  // turn LED on
            _delay_ms(1000); // 1 sec delay
            LEDPORT &= ~(1 << LED_PIN); // turn LED back off
        }
    }
}

short checkButton()
{
    short count = 0; // counter

    while(!(BUTTONPIN & (1 << BUTTON_PIN)) && // check if button is stayed press
            count < 10) //for 0.1 sec
    {
        // counter to make sure button is being pressed for 0.1 sec
        count++;
        _delay_ms(5);
    }

    return (count == 10);
    // return true if button has been held for 0.1 sec
}
```

b) Assembly code

```
.INCLUDE "M328pDEF.INC"

.org 0

    LDI R16, 4 ;needed to toggle LED
    SBI DDRB, 2 ;PB2 as output
    CBI DDRD, 2 ;PD2 as input
    LDI R17, 0  ;needed to toggle led

    ;OUT PORTB,R17
    LDI R20,5 ;to set prescaler
    OUT TCCR0B,R20 ;Prescaler: 1024

button:
    SBIS PIND, 2 ;prompts for button
    JMP begin ;jumps to LED toggle function
    RJMP button ;loop back to prompt for button

begin:
    EOR R17,R16 ;XOR to toogle led
    OUT PORTB,R17 ;display LED
    RCALL delay ;calling timer to wait for 1 sec
    EOR R17,R16
    OUT PORTB,R17
    RJMP button ;jump to prompt for button again
delay:
    IN R28, TCNT0 ;loading lower bit of counter to R28
    CPI R28, 0xFF ;check for overflow
    BREQ HIGHINC ;jump to increment overflow count

cont:
    CPI R28, 0xA0 ;comparing if timer is 0xA0
    BREQ body ;if equal, check for OVF_count
    RJMP delay ;if not, loop back

HIGHINC:
    INC R21 ;increment overflow count
    LDI R20, 0
    OUT TCNT0, R20 ;reset TCNT
    RJMP cont ;loop back to check timer bits

body:
    CPI R21,0x1E ;comparing if OVF_count is 0x1E
    BRSH done ;if equal, branch to "done"
    RJMP delay ;if not, loop back
done:
    LDI R21, 0 ;reset overflow count
    LDI R20, 0
    OUT TCNT0,R20 ;resetting the counter to 0 for next round
    RET
```

c) Timing Proof

In /da02_screenshots/TimingProof folder, under the names:

Assembly code: *ASM_T2.PNG*

C code: *C_T2.PNG*

**Task 3/C**: Implement Task 1 using Timer 0. Count OVF occurrence if needed. Do not use interrupts.

a) C code

```c
/*
    1) USE TIMER0 (TCNT0) TO CHECK FOR OVF
    2) ONCE OVF == 1, TOGGLE LED LIGHT
    3) RESTART TIMER0
    4) REPEAT
*/
#include <avr/io.h>

#define LEDPORT PORTB
#define LED_PIN PORTB2
#define LED_DDR DDRB

int main(void)
{
    volatile int count = 0; // initialize overflow count
    LED_DDR |= (1 << LED_PIN);  // Set LED to output
    LEDPORT &= ~(1 << LED_PIN); // Initial state of OFF

    TCCR0A = 0; // normal mode
    TCCR0B = (1 << CS02) | (1 << CS00); // Clock divided by 1024

    while (1)   // Loops forever
    {
        if(TCNT0 >= 255)
        {
            // using this check and the two for loops inside
            //with cause the LED to toggle for 0.25 sec
            if(count == 6)
            {
                for(volatile unsigned int i=0; i<38; i++)
                for(volatile unsigned int j=0; j<255; j++);
                LEDPORT ^= (1 << LED_PIN); //toggle LED light
                count = 0; //reset count to continuously toggle LED
            }
            else
                count++; // overflow occurred
        TCNT0 = 0; // reset timer
        }
    }
}
```

b) Assembly code

```
.org 0

    LDI R16, 4 ;needed to toggle LED
    LDI R21, 0 ;OVF count
    SBI DDRB, 2 ;PB2 as output
    LDI R17,0  ;needed to toggle led

    OUT PORTB,R17
    LDI R20,5 ;to set prescaler
    OUT TCCR0B,R20 ;Prescaler: 1024

begin:
    RCALL delay ;calling timer to wait for 0.25 sec
    EOR R17,R16 ;XOR to toogle led
    OUT PORTB,R17 ;display LED
    RJMP begin ;repeating i.e, while(1)
delay:
    IN R28, TCNT0 ;loading lower bit of counter to R28
    CPI R28, 0xFF ;check for overflow
    BREQ HIGHINC ;jump to increment overflow count
```

(assembly code cont.)

```
cont:
    CPI R28, 0xA0 ;comparing if timer is 0xA0
    BREQ body ;if equal, check for OVF_count
    RJMP delay ;if not, loop back

HIGHINC:
    INC R21 ;increment overflow count
    LDI R20, 0
    OUT TCNT0, R20 ;reset TCNT
    RJMP cont ;loop back to check timer bits

body:
    CPI R21,0x07 ;comparing if higher is 0x07
    BRSH done ;if equal, branch to "done"
    RJMP delay ;if not, loop back
done:
    LDI R21, 0 ;reset overflow count
    LDI R20, 0
    OUT TCNT0,R20 ;resetting the counter to 0 for next round
    RET
```

c) Timing Proof
In /da02_screenshots/TimingProof folder, under the names:
Assembly code: *ASM_T3.PNG*
C code: *C_T3.PNG*

**Task 4/D**: Implement Task 1 using TIMER0_OVF_vect interrupt mechanism.

a) C code

```
/*
    1) HAVE A 0.5 PERIOD, TOGGLE LED EVERY 0.25s
    2) ONCE OVF == 1, TURN LED ON FOR 1 SEC
    3) AFTER 1 SEC, CONTINUE 0.5 PERIOD LIKE NORMAL
*/
#include <avr/io.h>
#include <avr/interrupt.h>

#define LEDPORT PORTB
#define LED_PIN PORTB2
#define LED_DDR DDRB

volatile int count;

int main(void)
{
    volatile int count = 0;
    LED_DDR |= (1 << LED_PIN);  // Set LED to output

    TCCR0A = 0;
    TCCR0B |= (1 << CS02) | (1 << CS00);    // clock divided by 1024

    TCNT0 = 0;
    TIMSK0 |= (1 << TOIE0); // enable interrupt on overflow

    sei(); // turn on interrupts

    while (1);
}
```

(c code cont.)

```
// called after overflow of timer.
// toggle LED on/off
ISR(TIMER0_OVF_vect)
{
    if(count == 6){
        for(volatile unsigned int i=0; i<36; i++)
            for(volatile unsigned int j=0; j<255; j++);
                LEDPORT ^= (1 << LED_PIN);  //turn LED on
    count = 0;
    }
    else
        count++;
}
```

b) Assembly code

```
.org 0
    jmp  main
.org 0x20
    jmp  TIM0_OVF ;Timer0 overflow interrupt vector

main:
    LDI R16, 4 ;needed to toggle LED
    LDI R21, 0 ;OVF count
    SBI DDRB, 2 ;PB2 as output
    LDI R17,0  ;needed to toggle led

    ;OUT PORTB,R17
    LDI R20,5 ;to set prescaler
    OUT TCCR0B,R20 ;Prescaler: 1024

    LDI R20, 0x01 ;can also use (1<<TOIE0)
    STS TIMSK0,R20 ;enable overflow interrupt
    SEI ;enable global interrupts
Loop:
    RJMP Loop ;infinite loop

TIM0_OVF:

delay:
    IN R28, TCNT0 ;loading timer0 to R28
    CPI R28, 0xFF ;check for overflow
    BREQ HIGHINC ;jump to increment overflow count

cont:
    CPI R28, 0xB6 ;comparing if timer is 0xB6
    BREQ body ;if equal, check for OVF_count
    RJMP delay ;if not, loop back

HIGHINC:
    INC R21 ;increment overflow count
    LDI R20, 0
    OUT TCNT0, R20 ;reset TCNT
    RJMP cont ;loop back to check timer bits

body:
    CPI R21,0x06 ;comparing if OVF_count is 0x06
    BRSH done ;if equal, branch to "done"
    RJMP delay ;if not, loop back
done:
    EOR R17,R16 ;XOR to toogle led
    OUT PORTB,R17 ;display LED
    LDI R21, 0 ;reset overflow count
    LDI R20, 0
    OUT TCNT0,R20 ;resetting the counter to 0 for next round
    LDI R20, 5 ;Timer0: enabled, prescaler = 1024
    OUT TCCR0B, R20 ; prescaler = 1024
    RETI ;return from interrupt, interrupts enabled
```

c) Timing Proof
In /da02_screenshots/TimingProof folder, under the names:
Assembly code: *ASM_T4.PNG*
C code: *C_T4.PNG*

**Task 5/E**: Implement Task 2 using INT0 interrupt mechanism.

a) C code

```c
#define F_CPU 8000000UL
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

#define LEDPORT PORTB
#define LED_PIN PORTB2
#define LED_DDR DDRB

#define BUTTONPORT PORTD
#define BUTTONDDR DDRD
#define BUTTON_PIN PORTD2
#define BUTTONPIN PD2

ISR(INT0_vect)
{
    unsigned char temp; // Saves current state of LED light
    EIFR |= (1 <<INTF0); // clear flag
    temp = LEDPORT; // Save the LED light state
    LEDPORT |= (1 << LED_PIN);  // Turn LED on
    _delay_ms(1000);    // Keep LED on for 1 sec
    LEDPORT = temp; // Restore state
}

int main(void)
{
    LED_DDR |= (1 << LED_PIN);  // Set LED to output
    LEDPORT &= ~(1 << LED_PIN); // Initial state of OFF

    BUTTONPORT = (1 << BUTTONPIN);  // pull-up
    BUTTONDDR = (1 << BUTTONPIN);   // set PD2 as input (using for interrupt INT0)

    EIMSK = (1 << INT0);                    // Enable INT0
    EICRA = (1 << ISC01) | (1 << ISC00);    // Trigger INT0 on rising edge

    sei(); // enables interrupt

    while (1)   // Loops forever
    {
        LEDPORT |= (1 << LED_PIN);  // turn LED on
        _delay_ms(250); // 0.25 sec delay
        LEDPORT &= ~(1 << LED_PIN); // turn LED off
        _delay_ms(250); // 0.25 sec delay
    }
}
```

b) Assembly code

```
.INCLUDE "M328pDEF.INC"

.org 0
    jmp main
.org 0x02
    jmp INT0_v ;external interrupt vector

main:
    LDI R16, 4 ;needed to toggle LED
    SBI DDRB, 2 ;PB2 as output
    CBI DDRD, 2 ;PD2 as input
    LDI R17, 0  ;needed to toggle led

    LDI R20,5 ;to set prescaler
    OUT TCCR0B,R20 ;Prescaler: 1024

    LDI R20, (1 << INT0)
    OUT EIMSK, R20 ;set bits for EIMSK
    LDI R20, (1 << ISC11) | (1 << ISC01)
    STS EICRA, R20 ;set bits for EICRA
    SEI

button:
    SBI PIND, 2 ;prompts for button
    RJMP begin ;toggle LED if != button

begin:
    RCALL delay ;delay function
    EOR R17,R16 ;XOR to toogle led
    OUT PORTB,R17 ;display LED
    RJMP button ;jump to prompt for button again
delay:
    IN R28, TCNT0 ;loading lower bit of counter to R28
    CPI R28, 0xFF ;comparing if lower is 0x41
    BREQ HIGHINC

cont:
    CPI R28, 0xA0 ;comparing if timer is 0xA0
    BREQ body ;if equal, check for OVF_count
    RJMP delay ;if not, loop back

HIGHINC:
    INC R21 ;increment overflow count
    LDI R20, 0
    OUT TCNT0, R20 ;reset TCNT
    RJMP cont ;loop back to check timer bits

body:
    CPI R21,0x07 ;comparing if higher is 0x07
    BRSH done ;if equal, branch to "done"
    RJMP delay ;if not, loop back
done:
    LDI R21, 0 ;reset overflow count
    LDI R20, 0
    OUT TCNT0,R20 ;resetting the counter to 0 for next round
    RET

INT0_v:
begin2:
    LDI R22, (1 << INTF0)
    OUT EIFR, R22 ;clear flag
    SBI PORTB, 2 ;turn LED light on
    RCALL delay2 ;delay function
    LDI R22, 5 ;Timer0: enabled, prescaler = 1024
    OUT TCCR0B, R22 ; prescaler = 1024
    RETI ;return from interrupt, interrupts enabled
delay2:
    IN R28, TCNT0 ;loading timer0 to R28
    CPI R28, 0xFF ;check for overflow
    BREQ HIGHINC2 ;jump to increment overflow count

cont2:
    CPI R28, 0x84 ;comparing if timer is 0x84
    BREQ body2 ;if equal, check for OVF_count
    RJMP delay2 ;if not, loop back

HIGHINC2:
    INC R23 ;increment overflow count
    LDI R22, 0
    OUT TCNT0, R22 ;reset TCNT
    RJMP cont2 ;loop back to check timer bits

body2:
    CPI R23,0x1E ;comparing if OVF_count is 0x1E
    BRSH done2 ;if equal, branch to "done2"
    RJMP delay2 ;if not, loop back
done2:
    LDI R23, 0 ;reset overflow count
    LDI R22, 0
    OUT TCNT0,R22 ;resetting the counter to 0 for next round
    RET
```

c) Timing Proof

In /da02_screenshots/TimingProof folder, under the names:
Assembly code: *ASM_T5a.PNG* and *ASM_T5b.PNG*
C code: *C_T5.PNG*

**Calculations:**

For the purpose of obtaining the correct values for the delays, I used the TCNT formula to get the proper numbers for each task.

a) TCNT formula

$$TCNT = \left(\frac{clock\_speed}{prescaler\_value} * desired\_time\_in\_seconds\right) - 1$$

b) Plugging in values:
1st formula: 8000000 represents clock speed (in Hz), 1024 represents prescaler value, desired time is 0.25 seconds; the output value is 1952

2nd formula: 8000000 represents clock speed (in Hz), 1024 represents prescaler value, desired time is 1 second; the output value is 7812

$$\left(\frac{8000000}{1024} \cdot 0.25\right) - 1 = 1952.125$$

$$\left(\frac{8000000}{1024} \cdot 1\right) - 1 = 7811.5$$

c) converting 1952 to hex

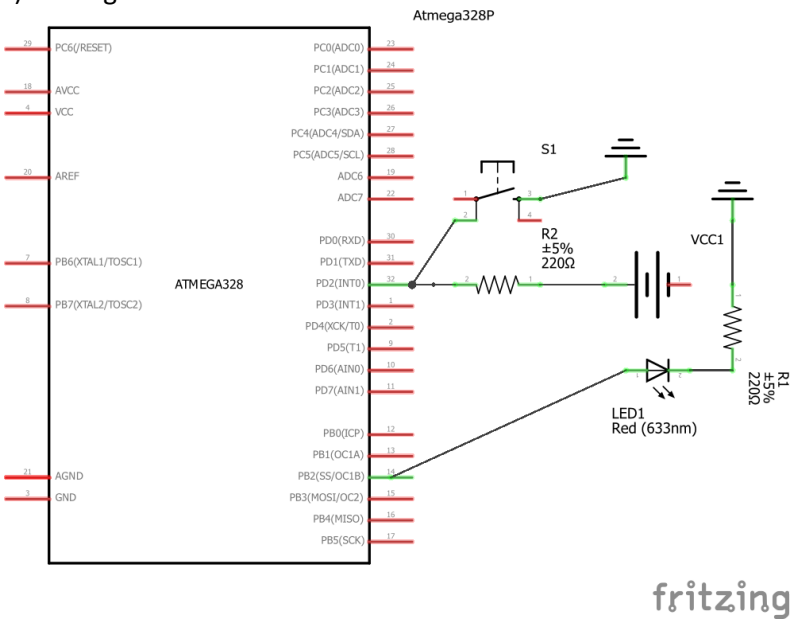
Enter decimal number:
1952 10
Convert Reset Swap
Hex number:
7A0 16
Hex signed 2's complement:
07A0 16
Binary number:
11110100000 2

d) converting 7812 to hex

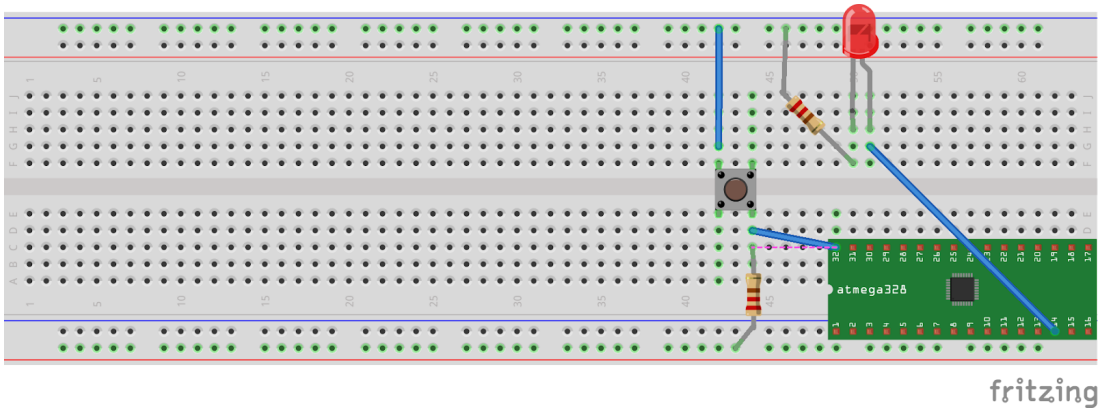
Enter decimal number:
7812 10
Convert Reset Swap
Hex number:
1E84 16
Hex signed 2's complement:
1E84 16
Binary number:
1111010000100 2

**Schematics**:
This schematic is the basic setup for this assignment. PORTB2 was used in all 5 tasks (website used to build this schematic did not let me connect the resistor and ATmega to the led, so the connection in the picture below was the best I could do). PORTD2 was used in task 2 and 5. The ATmega328p was connected to the ISR cable to my laptop and a VCC and GND source, which connected to the power supply (delivering a fixed 5V to microcontroller).

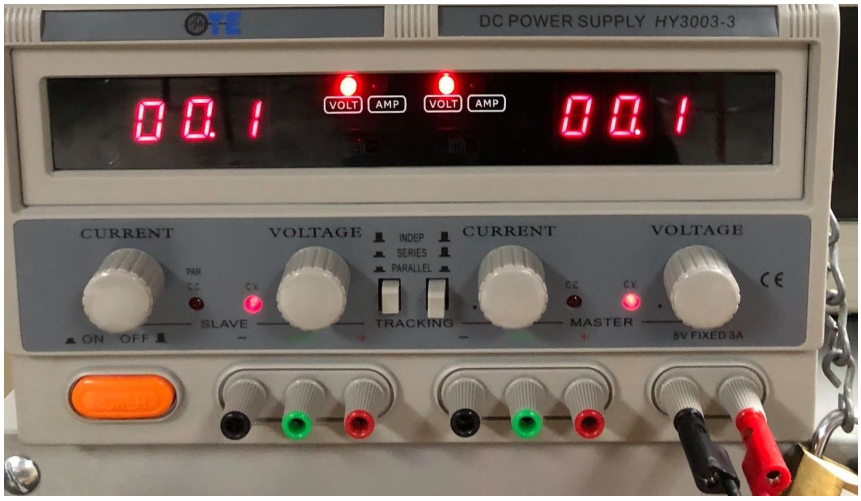
a) Fritzing schematics



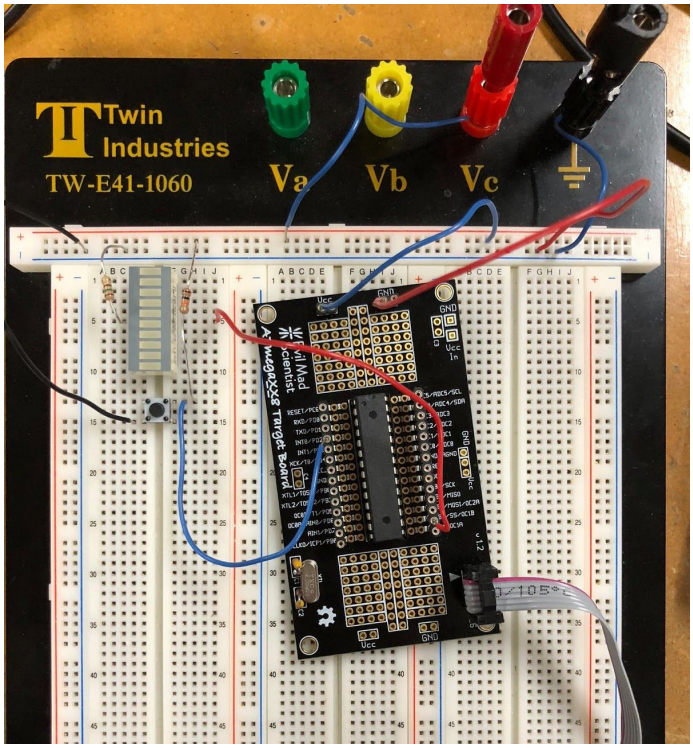

b) Fritzing breadboard




**Physical Set-Up**:
b) the power supply (using fixed 5V)




a) the breadboard set-up




Other notes:
**da02_screenshots** in the repository contains the photos of the timing proof, schematics, physical setup, and screenshots of all of my code for this assignment.
Brief explanations of the timing proof are in **TP_README.txt** in the same folder.
Outputs of each demo are shown in the Youtube videos. (It's difficult to prove with screenshots)

**GITHUB LINK:** https://github.com/JeffinVegas/EmbSys.git
**YOUTUBE LINK:** In the videos_DA02.txt file

**Student Academic Misconduct Policy**
http://studentconduct.unlv.edu/misconduct/policy.html

"*This assignment submission is my own, original work*".
Jeffrey Razon