Andrew Wells

CPE301 – SPRING 2016

Design Assignment 4

**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)** |
| 0. | COMPONENTS LIST AND CONNECTION BLOCK DIAGRAM w/ PINS |  |  |
| 1. | INITIAL CODE OF TASK 1/A |  |  |
| 2. | INCREMENTAL / DIFFERENTIAL CODE OF TASK 2/B |  |  |
| 3. | INCREMENTAL / DIFFERENTIAL CODE OF TASK 3/C |  |  |
| 4. | INCREMENTAL / DIFFERENTIAL CODE OF TASK 4/D |  |  |
| 5. | INCREMENTAL / DIFFERENTIAL CODE OF TASK 5/E |  |  |
| 6. | SCHEMATICS |  |  |
| 7. | SCREENSHOTS OF EACH TASK OUTPUT |  |  |
| 8. | SCREENSHOT OF EACH DEMO |  |  |
| 9. | VIDEO LINKS OF EACH DEMO |  |  |
| 10. | GOOGLECODE LINK OF THE DA |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 0. | COMPONENTS LIST AND CONNECTION BLOCK DIAGRAM w/ PINS |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 1. | INITIAL CODE OF TASK 1/A |  |  |

#define *F\_CPU* 8000000UL // Set clock speed to 8MHz

#include <avr/io.h>

#include <util/delay.h>

void DUTY\_CYCLE(); // Function to change duty cycle

void TIMER0\_INIT(); // Function to Initialize timer 0

void TIMER1\_INIT(); // Function to Initialize timer 1

void TIMER2\_INIT(); // Function to Initialize timer 2

int main(void)

{

DDRD = (1<<6);

DDRB = (1<<1)|(1<<3);

TIMER0\_INIT(); // Initialize timer 0

TIMER1\_INIT(); // Initialize timer 1

TIMER2\_INIT(); // Initialize timer 2

TCCR1B = (1<<WGM12) | // Prescaler to 1

(1<<CS10); // Timer Starts

TCCR0B = (1<<CS00); //Prescaler to 1, Timer starts

TCCR2B = (1<<CS20); // Prescaler to 1, Timer starts

while (1)

{

while (TCCR1B != 0x08)

{

while (TCCR0B != 0x00)

{

while (TCCR2B != 0x00)

{

DUTY\_CYCLE(); // Function to cycle through duty cycles

if (TCCR2B < 0x07)

TCCR2B++; // increase prescaler value

else

TCCR2B = 0x00; // Set prescaler to 0 to exit loop

}

TCCR2B = 0x01; // Reset Timer 2 prescaler to 1

if (TCCR0B < 0x05)

TCCR0B++; // increase prescaler value

else

TCCR0B = 0x00; // Set prescaler to 0 to exit loop

}

TCCR0B = 0x01; // Reset Timer 0 prescaler to 1

if (TCCR0B < 0x0D)

TCCR0B++; // Increase prescaler value

else

TCCR0B = 0x08; // Reset timer 1 to exit loop

}

TCCR1B = 0x09; // Reset Timer 1 prescaler to 1

}

return 0;

}

void DUTY\_CYCLE()

{

OCR1A = 25; // Initialize Compare value to 10% of 255

OCR0A = 25; // Initialize Compare value to 10% of 255

OCR2A = 25; // Initialize Compare value to 10% of 255

while(OCR1A < 249)

{

while(OCR0A < 249)

{

while(OCR2A < 249)

{

*\_delay\_ms*(150);

OCR2A = OCR2A + 25; // Increase compare value for timer 2 by 25 or 10%

}

OCR2A = 25; // Reset Compare value for timer 2 to 25 or 10%

OCR0A = OCR0A +25; // Increase compare value for timer 0 by 25 or 10%

}

OCR0A = 25; // Reset Compare value for timer 0 to 25 or 10%

OCR1A = OCR1A + 25; // Increase compare value for timer 1 by 25 or 10%

}

return;

}

void TIMER0\_INIT()

{

TCCR0A = (1<<COM0A1) | // Non Inverting Mode

(0<<COM0A0) | // Clear OC0A on Compare match, set OC0A at Bottom

(0<<COM0B1) | // Normal port operation

(0<<COM0B0) | // OC0B disconnected

(1<<WGM01) | // Fast PWM mode

(1<<WGM00);

return;

}

void TIMER2\_INIT()

{

TCCR2A = (1<<COM2A1) | // Non Inverting Mode

(0<<COM2A0) | // Clear OC2A on Compare match, set OC2A at Bottom

(0<<COM2B1) | // Normal port operation

(0<<COM2B0) | // OC2B disconnected

(1<<WGM21) | // Fast PWM mode

(1<<WGM20);

return;

}

void TIMER1\_INIT()

{

TCCR1A = (1<<COM1A1) | // Non Inverting Mode

(0<<COM1A0) | // Clear OC2A on Compare match, set OC2A at Bottom

(0<<COM1B1) | // Normal port operation

(0<<COM1B0) | // OC2B disconnected

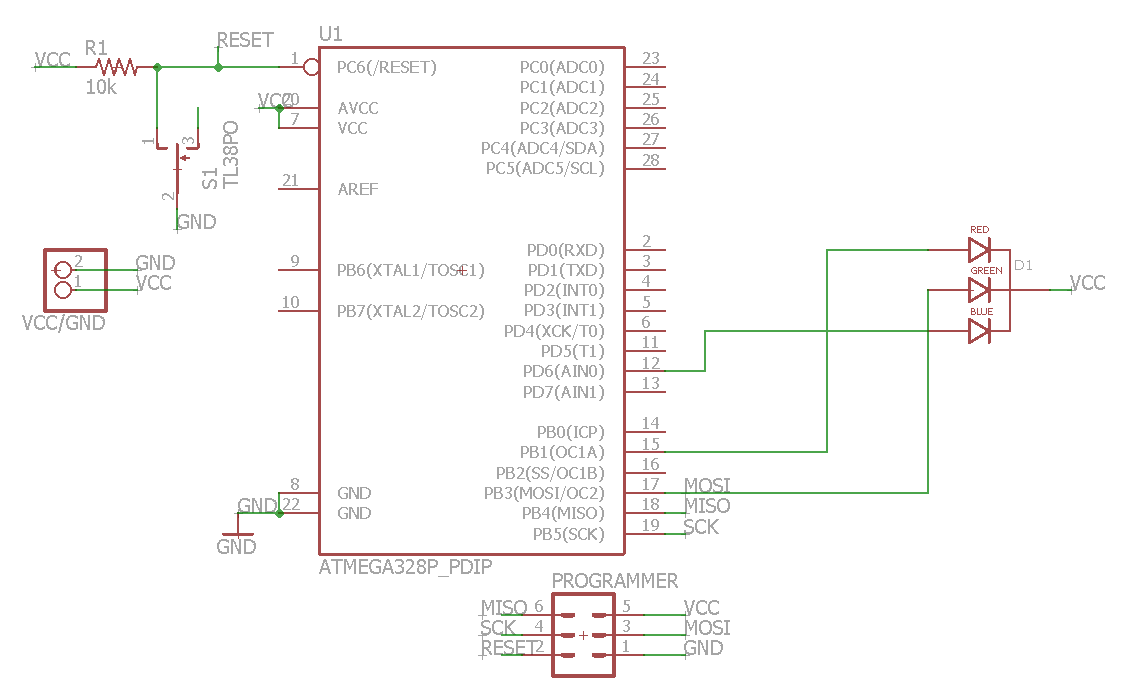
(0<<WGM11) | // Fast PWM mode

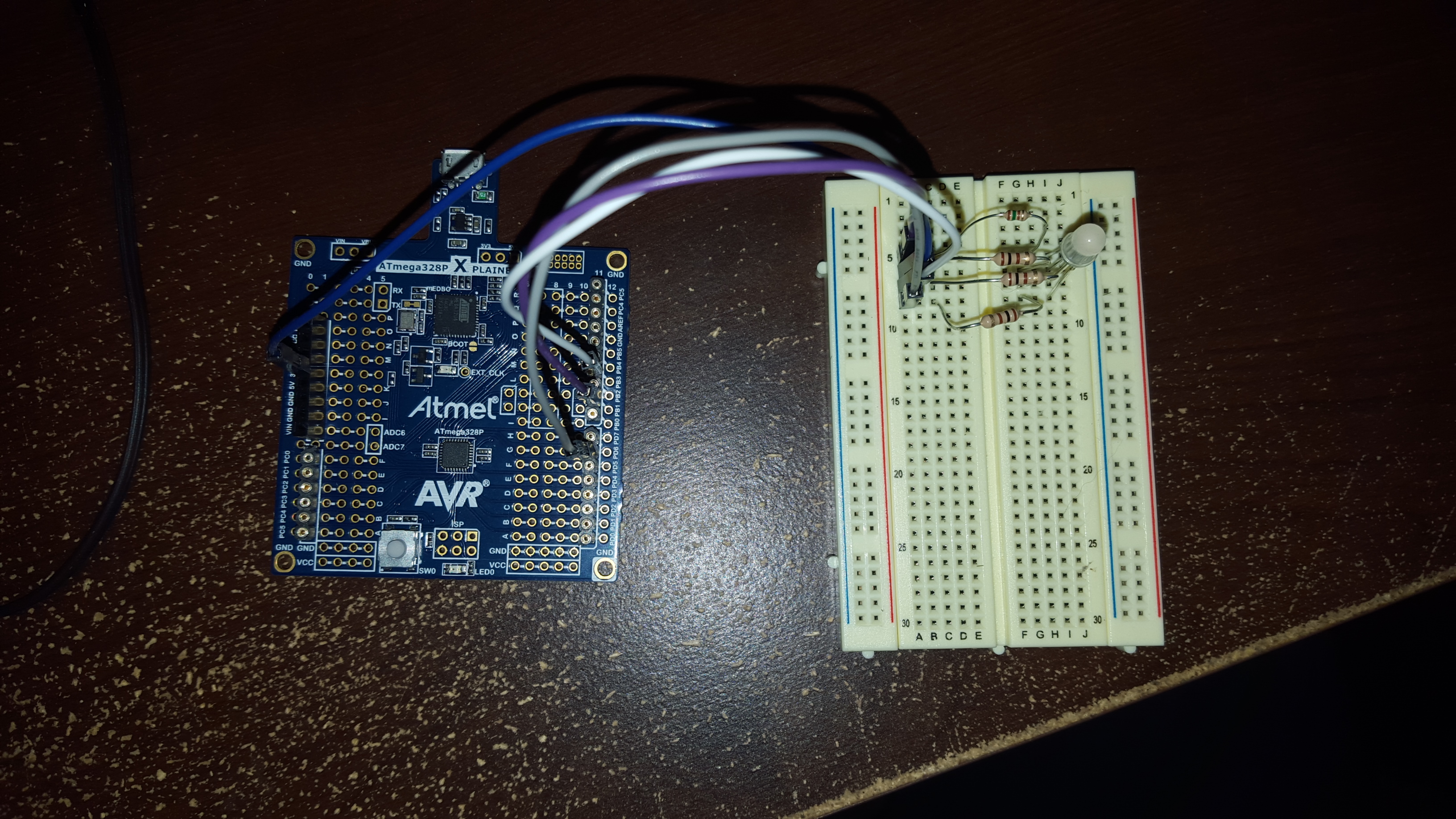
(1<<WGM10);

return;

}

|  |  |  |  |
| --- | --- | --- | --- |
| 6. | SCHEMATICS |  |  |

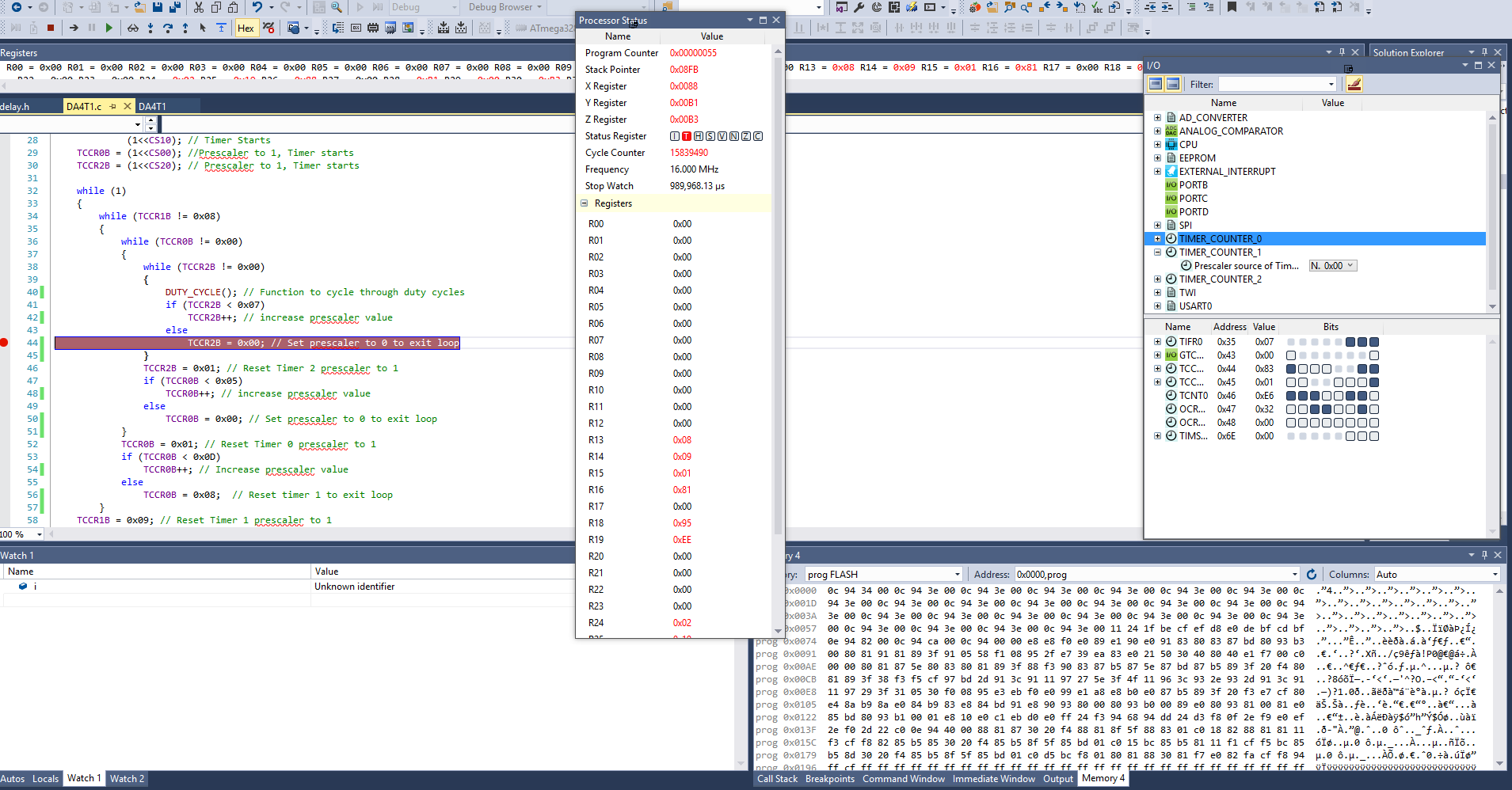




|  |  |  |  |
| --- | --- | --- | --- |
| 7. | SCREENSHOTS OF EACH TASK OUTPUT |  |  |

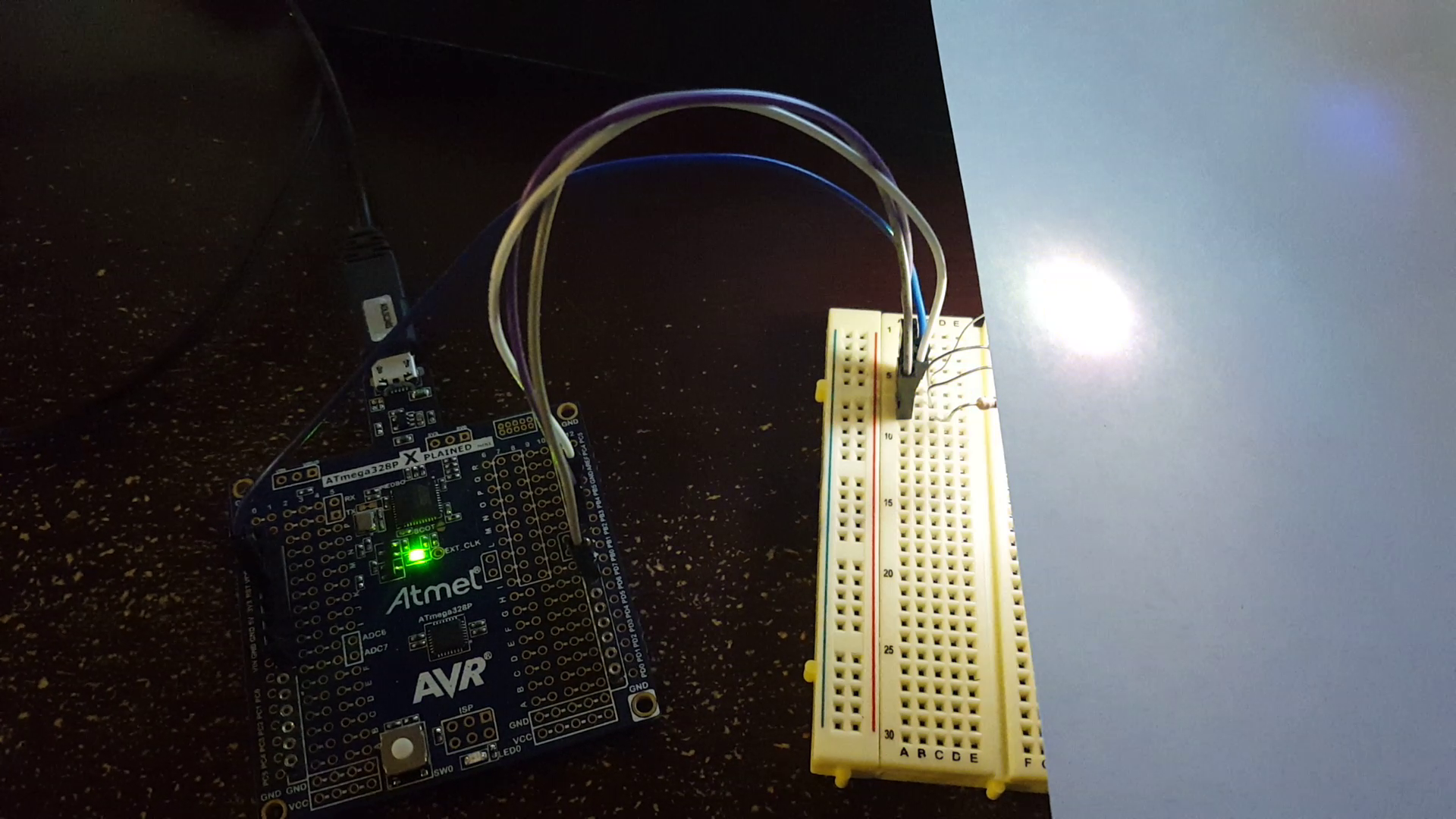
TASK 1:

Write an AVR C program to generate three PWM signals to drive the RGB LED using TIMERs. Use the OCnX pins to generate the output. Increment individually each PWM period from min. (10%) to max (90%) value, at the same time alter the each PWM duty cycle. The RGB LED will display different colors as the PWM periods are changed and the brightness of the LED with vary with the change in duty cycle.



|  |  |  |  |
| --- | --- | --- | --- |
| 8. | SCREENSHOT OF EACH DEMO |  |  |

TASK 1:



**Flowchart**

START

OCR1A = 25

OCR0A = 25

OCR2A = 25

Start Timers

Initialize Timers

DDRB.1 = 1

DDRB.3 = 1

DDRD.6 = 1

OCR2A = OCR2A + 25

150 ms DELAY

TRUE

OCR2A < 249

FALSE

OCR2A = 25

OCR0A = OCR2A + 25

TRUE

OCR0A < 249

FALSE

OCR0A = 25

OCR1A = OCR1A + 25

TRUE

OCR1A < 249

FALSE

TRUE

TCCR2B < 0x07

TCCR2B++

TCCR2B = 0

FALSE

TCCR2B = 0x00

TRUE

TCCR2B = 0x01

TCCR0B < 0x07

TCCR0B++

TCCR0B = 0

FALSE

TCCR2B = 0x00

TRUE

TCCR0B = 0x01

TCCR1B < 0x0D

TCCR1B = 0x08

TCCR1B++

FALSE

TCCR2B = 0x08

TRUE

TCCR0B = 0x09

|  |  |  |  |
| --- | --- | --- | --- |
| 9. | VIDEO LINKS OF EACH DEMO |  |  |
| https://youtu.be/\_Ij-k3H20Z0 | | | |
| 10. | GOOGLECODE LINK OF THE DA |  |  |
| https://github.com/Wellsa15/wellsa\_unlv | | | |

**Student Academic Misconduct Policy**

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

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

Andrew Wells