B31DD – Embedded Systems

Lab 2
ADC, PWM and Motor Control

SUBMISSION DEADLIN 7 nov 2021

JANNIN Sylvain

H00387879

Msc Robotics

Date of submission: 31 oct 2021

Task 1

The goal of this task is to create a PWM with a frequency of 61.04Hz with 4 different pulse width: 0%, 25%, 62.5% and 87.5%.

For this we use fast PWM mode with timer0 with the non-inverting mode. Now we need to calculate the prescaler.

$$F_{OC0} = \frac{f_{clk}}{N(256)}$$

$$N = \frac{f_{clk}}{F_{OC0}(256)}$$

$$N = \frac{16 \times 10^6}{61.04 \times 256}$$

$$N = 1023.91 \approx 1024$$

We choose the closet value possible, so 1024. For TCCR0B register, we need to set CS00 and CA02 to 1, the rest to 0.

We set fast PWM with non-inverted mode, for this we need on the TCCR0A register:

- Fast PWM: WGM01 and WMGM00 set to 1.
- COM0A1 set to 1: non-inverting mode.
- Everything else is set to 0.

Now, we have to configure the duty cycle (=α):

$$\frac{\alpha}{100} = \frac{OCR0A + 1}{256}$$

$$OCR0A = \frac{256\alpha}{100} - 1$$

- α = 0 → $OCR0A = \frac{256 \times 0}{100} - 1 = 0$
- α = 25 → $OCR0A = \frac{256 \times 25}{100} - 1 = 63$
- α = 62.5 → $OCR0A = \frac{256 \times 62.5}{100} - 1 = 159$
- α = 87.5 → $OCR0A = \frac{256 \times 87.5}{100} - 1 = 223$

To summarize:

- TCCR0A = 0x83
- TCCR0B = 0b00000101
- $OCR0A$

- α = 0 → $OCR0A = 0$
- α = 0 → $OCR0A = 63$
- α = 0 → $OCR0A = 159$
- α = 0 → $OCR0A = 223$

The program is very simple. Once everything has been initialized, we enter the while loops, it checks the switch connected to PC0 is pressed or not. If yes it changes the value of the PWM thanks to the "*switch*_duty" function. (for this task the switch is connected on PC0, for task 2, this button is connected on PC3).

Code

```
/*
 * Lab2_task1__JANNIN_Sylvain.c
 *
 * Created: 27/10/2021 10:50:14
 * Author : Sykvain Jannin
 */

#include <avr/io.h>
#define F_CPU 16000000UL


void init_PWM()
{
	DDRD |= 1 << 6 ;	//set PD6 as enable output,
	PORTB |= (1 << PINB1);
	TCCR0A = 0x83;				//set fast PWM, non-inverted
	TCCR0B = 0b0000101;	//prescaler of 1024
}


void switch_dutycycle(int counter)
{
	switch(counter)
	{
		case(0):
			OCR0A = 0;	// stop the PWM
			break;

		case(1):
			OCR0A = 63;	//duty cycle of 25%
			break;

		case(2):
			OCR0A = 159;	//duty cycle of 62.5%
			break;

		case(3):
			OCR0A = 223;	//duty cycle of 87.5%
			break;

		default:		//in case the number is wrong
```

```
                    OCR0A = 0;    // stop the PWM
                    break;
        }

}


int main(void)
{
        int counter = 0;

        //init in and output
        DDRC = 0;     //Declare all as input
        init_PWM();
    while (1)
    {
                if(PINC & (1 << 0)) // if the button is pressed oh PC0
                {
                        while(PINC & (1 << 0));    //wait for the user to unleash the
butotn
                        switch_dutycycle(counter);
                        counter = counter +1;

                        if(counter >= 4)     // if counter is equal to 3 to higher, we
reset its value
                        {
                                counter = 0;
                        }

                }
    }
        return(0);
}
```

Task 2

Electric circuit

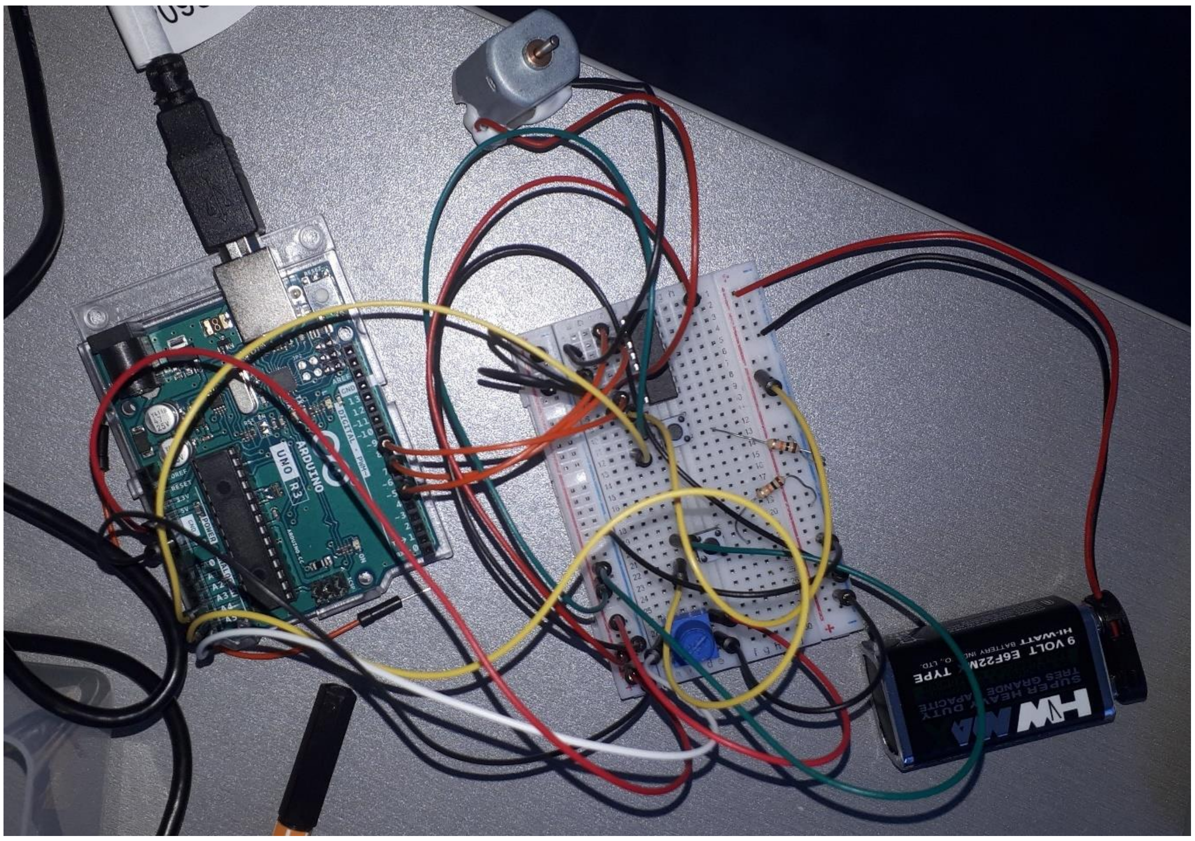
Figure 1: Picture of task 2.

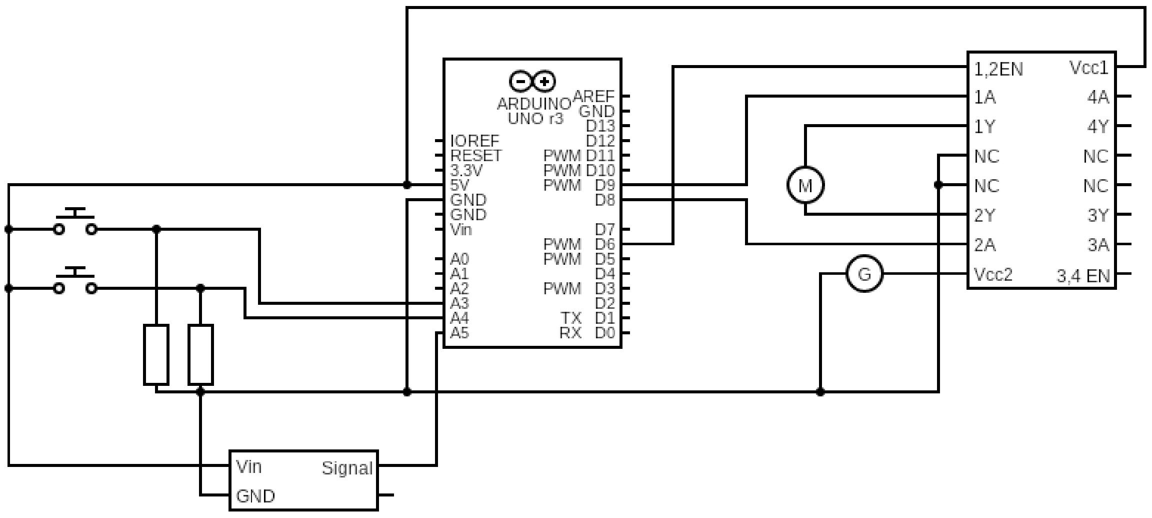

Figure 2: Schematic of task 2.

The two switches are inputs of the Arduino, one controlled the duty cycle the second one the rotation. For both of this switch, there is a pulldown resistor in order to have a better signal. The potentiometer is an input of the Arduino(it is represented on the schematic by a 4 pins chip), it controls the duty cycle when the "*Analog_control*" variable is not set to 1 (more information the in following section).

The circuit is based on the circuit showed as example in class. From the Arduino there are 3 wires, two to control the rotation (PB0 and PB1) and one is the PWM (PD6 because we use timer0). The motor is connected on the H-bridge. We have to power the H-Bridge with both 5V from the Arduino and 9V from an external battery. The motor is connected on the H-bridge on pin 1Y and 2Y.

Program idea

The goal is to control the motor by using the H-Bridge. There are two switches and one potentiometer: one switch controls the motor rotation, one switch the duty cycle of the PWM and the potentiometer also controls the duty cycle of the PWM. Since it will not make sense to control the duty cycle with both a switch and a potentiometer, there is the variable "*Analog_control*" in the code to control it: if this value is set to 1 the potentiometer controls the PWM, otherwise it is the switch. The switch for the PWM works the same way as in task 1.

For the rotation control, as in task 1, there is a switch, if the user presses it the rotation of the motor will change thanks to "*Change_Direection*" function. The function checks which of the two output is set to 1 then set this output to 0 and the other one to 1.

About the potentiometer in the while loops, we check the value, gives by the potentiometer and we change the OCR0A accordingly. Since the value get by Arduino is coded on 10 bits on the ADC register and OCR0A is codded is 8 bits, we need to shift the value of two bits.

ADC Registers

The registers of the ADC are set in the "*initADC*" function. We take as reference voltage AVCC (5V) and we plug the output of the potentiometer in PC5. We also want to use a prescaler of 8 and we want the ADC to be enable

- ADMUX:
    - REFS0 is set to 1 to take AVCC as reference.
    - MUX0 and MUX2 are set to 1 to use ADC on PC5.
- ADCRSA:
    - ADPS0 and ADPS1 are set to one to have a prescaler of 8.
    - ADEN is set to 1 to enable the ADC.

Once the registers are set, we set to 1 ADSC in ADCRSA to enable ADC. To read the value we read registers ADCL and ADCH, the ADCL is the low value and ADCH the high value. In the end, the value given by the ADC is “ADCH + ADCL”. But, since The ADC is coded on 10 bits, we have to do an extra step to have the correct value. Before adding them to obtain the value we need to shift ADCH of 8 bits.

Code :

```c
/*
 * Lab2_task2__JANNIN_Sylvain.c
 *
 * Created: 27/10/2021 10:50:14
 * Author : Sykvain Jannin
 */

#include <avr/io.h>
#include <avr/interrupt.h>
#define F_CPU 16000000UL



void init_PWM()
{
	DDRD |= 1 << 6 ;     //set PD6 as enable output,
	DDRB |= 1 << PINB0 ;//set PB0 as output,
	DDRB |= 1 << PINB1 ;//set PB1 as output,
	PORTB |= (1 << PINB0);      // init the direction
	TCCR0A = 0x83;
	TCCR0B = 0b0000101;  //prescaler of 1024
}


void initADC(void)
{
	ADMUX |= (1 << REFS0) | (1 << MUX2) | (1 << MUX0); //reference voltage on AVCC, 
MUX = 0101 -> ADC5
	ADCSRA |= (1 << ADPS1) | (1 << ADPS0); //ADC clock prescaler / 8
	ADCSRA |= (1 << ADEN); //enables the ADC
}

void switch_dutycycle(int counter)
{
	switch(counter)
	{
		case(0):
			OCR0A = 0;    // stop the PWM
			break;

		case(1):
			OCR0A = 63;   //duty cycle of 25%
			break;

		case(2):
			OCR0A = 159;  //duty cycle of 62.5%
			break;

		case(3):
```

```
                        OCR0A = 223;  //duty cycle of 87.5%
                        break;

                default:
                        OCR0A = 0;    // stop the PWM
                        break;        //in case the number is wrong
        }

}

void Change_Direction()
{
        if( PINB & (1<< 0))          // if PB0 is is on
        {
                PORTB &= ~(1<<0);      //turn off PB0
                PORTB |= (1<<1);       //turn on PB1
        }

        else                                 //Otherwise
        {
                PORTB &= ~(1<<1);     //turn off PB1
                PORTB |= (1<<0);      //turn on PB0
        }

}

uint16_t get_PotentiometerValue()
{
        uint16_t potentiometerValue;
        initADC();
        ADCSRA |= (1 << ADSC); //start ADC conversion
        while((ADCSRA & (1 << ADSC))) //wait until ADSC bit is clear, i.e., ADC
conversion is done
        {}
        //read ADC value
        uint8_t theLowADC = ADCL;
        potentiometerValue = ADCH << 8 | theLowADC;           //shift ADCH of 8 bits
        return potentiometerValue;
}



int main(void)
{
        int counter = 0;
        int Analog_control = 1;
        uint16_t potentiometerValue;

        //init in and output
        DDRC = 0;     //Declare all as input
        init_PWM();


    while (1)
    {
                if(Analog_control)   //Used the potentiometer to control the motor
                {
                        potentiometerValue = get_PotentiometerValue();
                        unsigned char dutycycle = potentiometerValue >> 2;           //
Arduino is coded on 10 bits, the ADC on 8bits --> We change the value, we shift it of
two bits inorder for the value to be fit on 8 bits and have something more accruate
                        OCR0A = dutycycle;
```

```
              }

              else                  //Used the switch to control the motor
              {
                     if(PINC & (1 << 3))  // if PC3 button is pressed, change the duty
cycle
                     {
                            while(PINC & (1 << 3));           //wait for the user to
unleash the switch
                            switch_dutycycle(counter);
                            counter = counter +1;

                            if(counter >= 4)     // if counter is equal to 3 to higher,
we reset its value
                            {
                                   counter = 0;
                            }
                     }
              }

              if(PINC & (1 << 4))  // if PC4 is pressed, change the direction
              {
                     while(PINC & (1 << 4));     //wait for the user to unleash the
switch
                     Change_Direction();

              }


    }
       return(0);
}
```

Task 3

Electric circuit

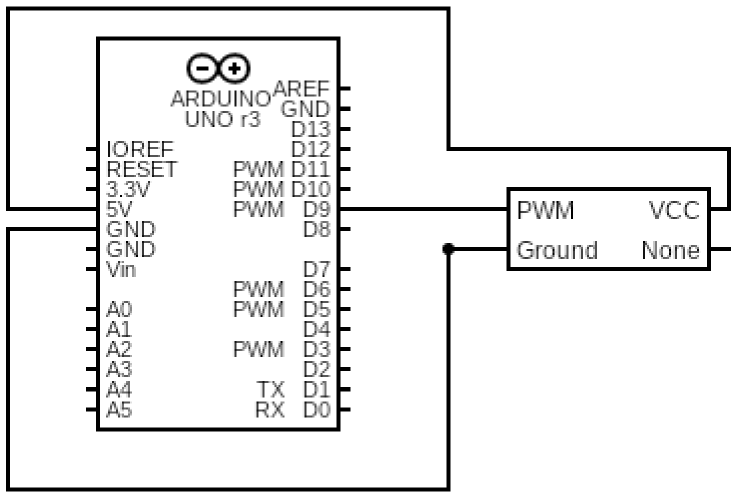

Figure 3: Schematic of task 3.

The servo motor is represented by the 4 pins chip.

Duty cycle and registres

For this task we need a PWM of 50Hz, and our clock is 16MHZ. For this task it is easier to use timer1 coded on 16 bits with a prescaler of 8 like in the example in the saw during the class.

To calculate the duty cycle we use the same method as we did in task1 but now we want to count until 40000 and not until 256. It becomes:

$$\frac{\alpha}{100} = \frac{OCR1A}{40000}$$

$$OCR1A = \frac{40000\,\alpha}{100} = 400\,\alpha$$

Our servo motor works for a duty cycle from 2.5% to 12.5%

- Angle = -90° → α = 2.5 → $OCR1A = 400 \times 2.5 = 1000$
- Angle = -45° → α = 5 → $OCR1A = 400 \times 5 = 2000$
- Angle = 0° → α = 7.5 → $OCR1A = 400 \times 7.5 = 3000$

- Angle = 45° → α = 10 → $OCR1A = 400 \times 10 = 4000$
- Angle = 90° → α = 12.5 → $OCR1A = 400 \times 12.5 = 5000$

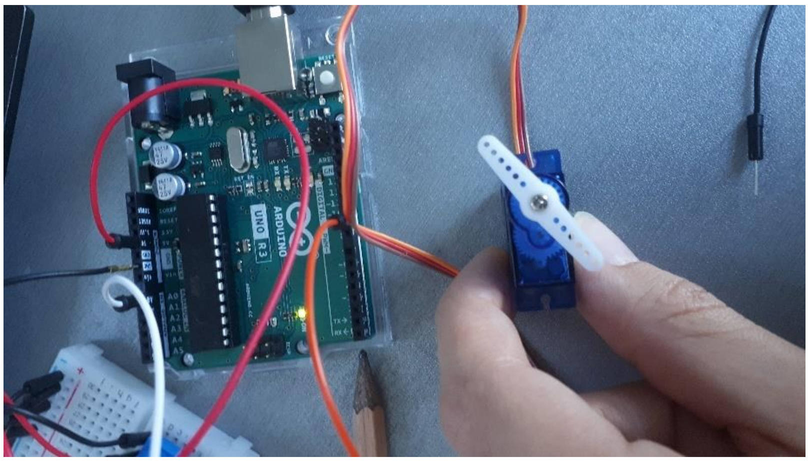
Figure 4: picture of task 3, angle = -45°.

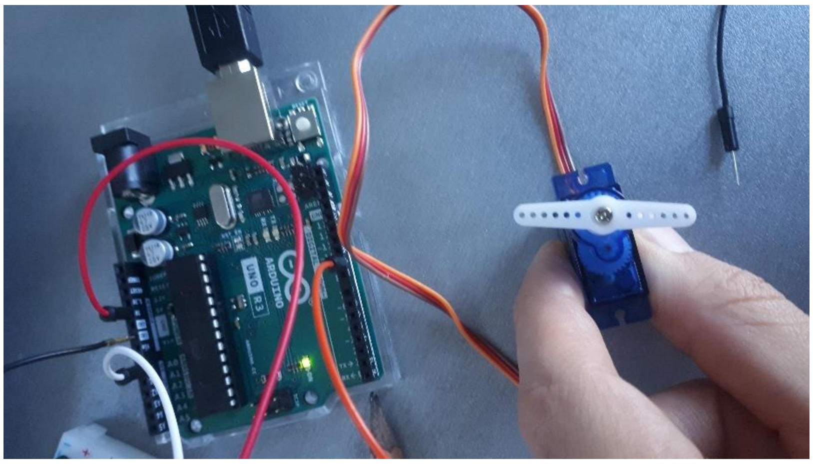
Figure 5: picture of task 3, angle = 0°.

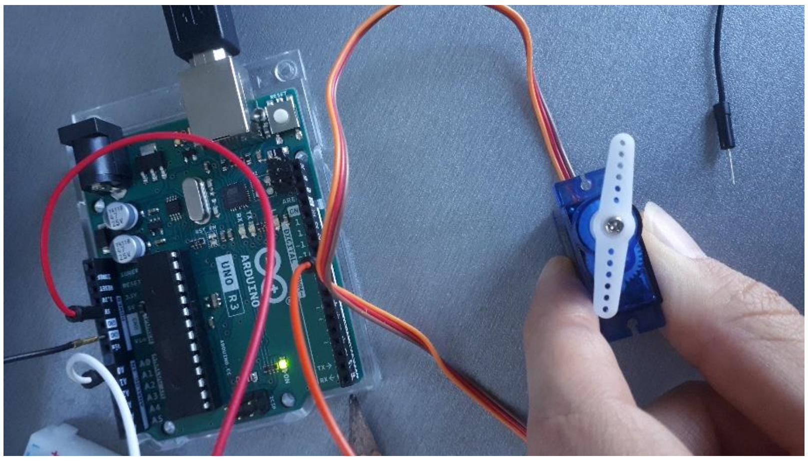
Figure 6: picture of task 3, angle = 90°.

Code

```c
/*
 * Lab2_task3__JANNIN_Sylvain.c
 *
 * Created: 27/10/2021 23:23:39
 * Author : Sylvain JANNIN
 */

#define F_CPU 1600000UL
#include <avr/io.h>
#include <util/delay.h>



void init_PWM()
{
	DDRB |= (1 << PINB1); // Set pin 9 to output PWM
	/* 1. Set Fast PWM mode 14: WGM11, WGM12, WGM13 to 1*/
	/* 2. Set pre-scale of 8 */
	/* 3. Set Fast PWM non-inverting mode */
	TCCR1A |= (1 << WGM11) | (1 << COM1A1);
	TCCR1B |= (1 << WGM13) | (1 << WGM12) |(1 << CS11);
	/* 4. Set ICR1: ICR1 is the top defining PWM period */
	ICR1 = 40000;
}


int main(void)
{
	init_PWM();
	while(1)
	{

		OCR1A = 1000;		// duty cycle of 2.5%
		_delay_ms(8000);

		OCR1A = 2000;		// duty cycle of 5%
		_delay_ms(8000);

		OCR1A = 3000;		// duty cycle of 7.5%
		_delay_ms(8000);

		OCR1A = 4000;		// duty cycle of 10%
		_delay_ms(6000);

		OCR1A = 5000;		// duty cycle of 12.5%
		_delay_ms(8000);

		OCR1A = 4000;		// duty cycle of 10%
		_delay_ms(8000);

		OCR1A = 3000;		// duty cycle of 7.5%
		_delay_ms(8000);

		OCR1A = 2000;		// duty cycle of 5%
		_delay_ms(8000);
```

```
        }
        return 0;
}
```