Lab Exercise #2: Analog Output (PWM), Timers and Analog Input

Frank Li
NetID: jl13581
Email: jl13581@nyu.edu
Section **C1**

Experiment Date: October 18, 2024
Submission Date: October 31, 2024

Background

On the Playground Circuit Ciassic board, the chip using is the ATmega32U4, which has one ADC built within it. In real-worldmembedded systems, engineers are oftently required to read the analog data and convert it into digital values. In this lab session, we first generate the PWM waveform from the chip, then using the ADC port to read the converted waveform passed through the low-pass filter. ADCs operate by sampling the signal at regular intervals and converts samples into a binary number.

Setting up the Analog Signal:

After uploading the set up code to the Adafruit Circuit Playground Ciassic, I got the following output:

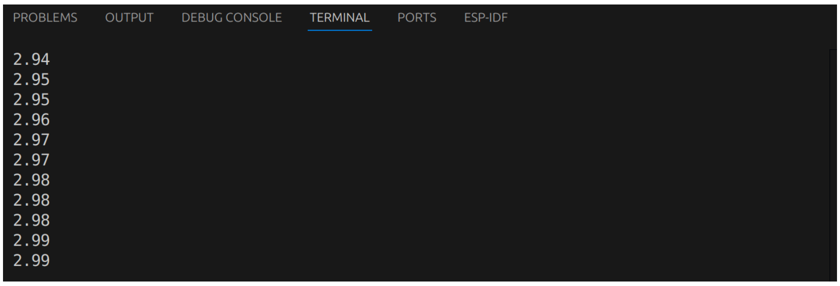

Figure 1: Serial Output After Setup

which states that I have correctly set up the device in generating the Sine wave.

Setting up the PWM Signal:

→**(1)** When we are setting up the output on the SCL(D3), the timer we must use is the **Timer0/Counter0**.
→**(2)** Code for setting up the Fast PWM signal on the board:

```
// Timer Setup
DDRD |= (1<<0);             // Set PD0 (Digital Pin 3) as output
TCCR0A = 0b00100011;        // COM0B1:0 = 10 (non-inverting),
                               WGM01:00 = 11 (Fast PWM)
TCCR0B = 0b00001011;        // WGM02 = 1 (Fast PWM with OCR0A as
                               TOP), CS02:00 = 011 (prescaler of 64)
OCR0A = 124;                // Set TOP value for PWM
OCR0B = 0;                  // Initialize duty cycle
```

For `TCCR0A`, the selection si based on: 00 - Normal Port; 10 - Set 0A as the TOP; WGM11 for Fast PWM.
For `TCCR0B`, I set the prescaler to 64 to make it easier to capture the diagram from the scope.
For `OCR0A`, I set the value to 124 to make sure it is close to 1kHz when running while not surpassing it, as described in the lab manual.
→**(3)** By using the Diligent Analog Discovery 2, I can have the PWM waveform generated by the board from the above code:

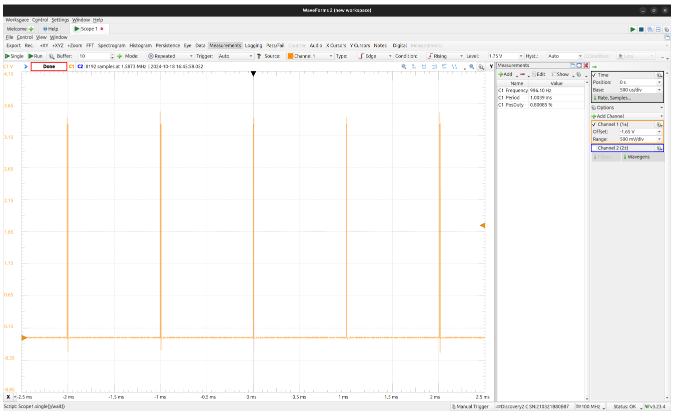

Figure 2: Scope After the PWM is set up

From the above diagram, we can see that the PWM is set to **1kHz**, as required in the manual. In order to make the duty cycle proportional to the value of the aval, I have added one line in the `loop()` function:

```
void loop() {
    // put your main code here, to run repeatedly:
    float aval;
    long x;
    x = millis();
    aval = abs(3 * sin(2 * 3.141592654 * x / 1000));
    OCR0B = (aval * OCR0A) / 3; // make the signal more discrete.
    delay(1);
    Serial.println(aval);
}
```

After this modification, the output duty cycle changes as the value of `aval` changes:

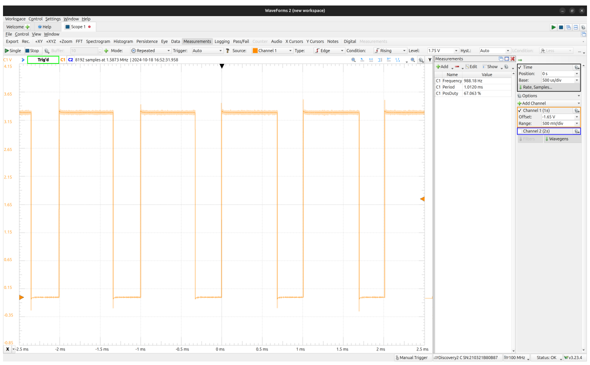

Figure 3: Scenario 1

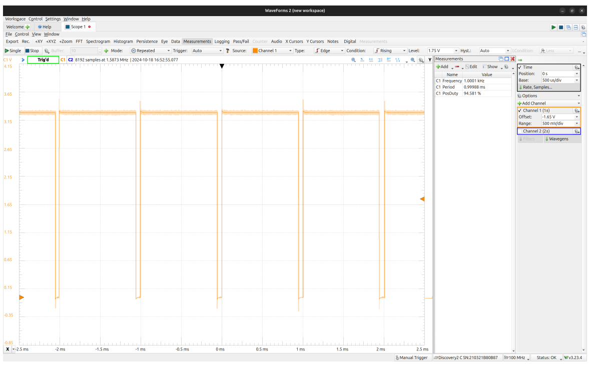

Figure 4: Scenario 2

Now the PWM signal works just like the description in the manual, that the frequency is **1kHz** and the duty cycle changes as the `aval` changes.

Setting up the Demodulator:

→**(4)** From the above diagram, we can see that the curren PWM signal frequency is 1kHz. From the code provided, we can see that the function generation is in milliseconds. Thus, the frequency should be 1Hz. The cutoff frequency, as described in the manual, should be:

$$\begin{aligned} \text{Frequency} &= \frac{1}{RC} \\ &= \frac{1}{10K\Omega \cdot 1\mu F} \\ &= 100Hz \end{aligned}$$

and we can see that PWM Signal > Cutoff Frequency > Analog Frequency, which is expected.

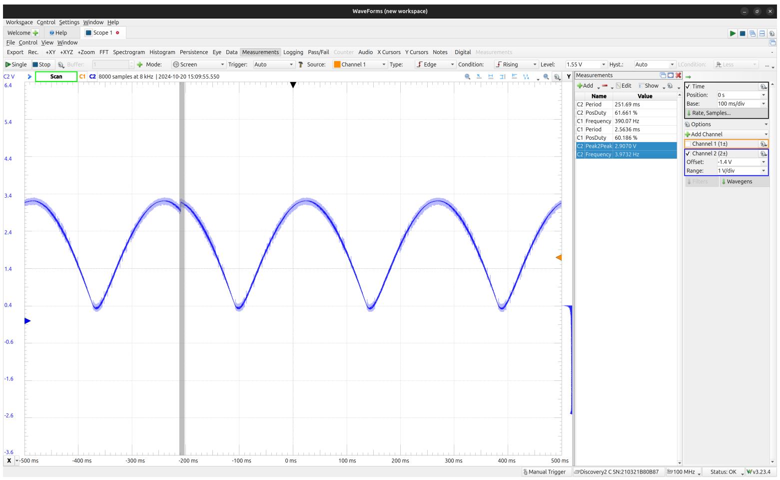

Figure 5: Scope After the Demodulator

→**(5)** From the above diagram, we can see that the analog output frequency is 4Hz, and the Peak to Peak value is around 3V. The freqneucy detected is a little bit off from the expectation but still within the error bound.

Setting up the Analog Input:

→**(6)** Now to set up the ADC on the board, I have the following code for registers:

```
// ADC Setup
DDRD &= ~(1<<6);            // Set PD6 (ADC9) as input
DIDR2 |= (1<<1);            // Disable digital input buffer on ADC9
ADMUX = 0b01000001;         // REFS1:0 = 01 (AVCC), ADLAR = 0 (right adjust),
                               MUX4:0 = 00001 (ADC1 initially)
ADCSRB = 0b00100000;        // MUX5 = 1 (for ADC8-ADC15),
                               ADTS2:0 = 000 (Free Running Mode)
ADCSRA = 0b10100111;        // ADEN = 1 (enable ADC), ADATE = 1 (auto trigger),
                               ADPS2:0 = 111 (prescaler 128)
ADCSRA |= (1<<ADSC);        // Start ADC conversions
```

Now since the ADLAR selected above is 0, the code in the `loop()` is:

```
unsigned short *ADCData;
unsigned short ADCVal;
ADCData=(unsigned short *)0x78;
ADCVal=(*ADCData & 0x3FF);
```

```
    float fADCVal;
    fADCVal=((float)ADCVal)/1023 * 3;
    Serial.print(abs(aval));

    Serial.print(" ");
    Serial.println(fADCVal);
    //Original rectified sinusoid
    //Analog voltage measured from ADC
```

→**(7)** With the above codes in the `loop()` function, I got the following results in the serial portal in the PlatformIO:

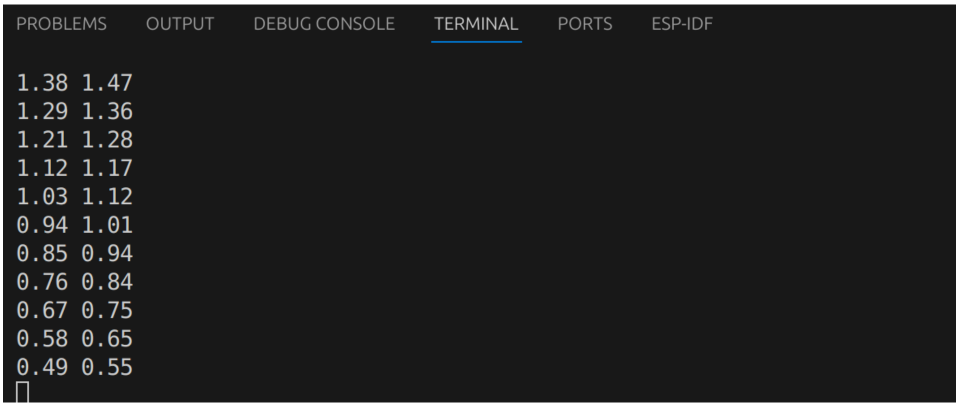

Figure 6: Serial Portal Output From the ADC

Below is the complete code for this lab:

```
#include <Arduino.h>

void setup() {
// Serial portal Setup
Serial.begin(9600);

// Timer Setup
DDRD |= (1<<0);             // Set PD0 (Digital Pin 3) as output
TCCR0A = 0b00100011;        // COM0B1:0 = 10 (non-inverting),
                               WGM01:00 = 11 (Fast PWM)
TCCR0B = 0b00001011;        // WGM02 = 1 (Fast PWM with OCR0A as TOP),
                               CS02:00 = 011 (prescaler 64)
OCR0A = 124;                // Set TOP value for PWM
OCR0B = 0;                  // Initialize duty cycle

// ADC Setup
DDRD &= ~(1<<6);            // Set PD6 (ADC9) as input
DIDR2 |= (1<<1);            // Disable digital input buffer on ADC9
ADMUX = 0b01000001;         // REFS1:0 = 01 (AVCC), ADLAR = 0 (right adjust),
                               MUX4:0 = 00001 (ADC1 initially)
ADCSRB = 0b00100000;        // MUX5 = 1 (for ADC8-ADC15),
                               ADTS2:0 = 000 (Free Running Mode)
ADCSRA = 0b10100111;        // ADEN = 1 (enable ADC),
```

```
                                ADATE = 1 (auto trigger),
                                ADPS2:0 = 111 (prescaler 128)
ADCSRA |= (1<<ADSC);         // Start ADC conversions
}

void loop() {
// put your main code here, to run repeatedly:
float aval;
long x;
x = millis();
aval = abs(3 * sin(2 * 3.141592654 * x / 1000));
OCR0B = (aval * OCR0A) / 3; // make the signal more discrete.
delay(1);
// Serial.println(aval);

unsigned short *ADCData;
unsigned short ADCVal;
ADCData = (unsigned short *)0x78;
ADCVal = (*ADCData & 0x3FF);

float fADCVal;
fADCVal = ((float)ADCVal) / 1023 * 3;

Serial.print(abs(aval));
Serial.print(" ");
Serial.println(fADCVal);
// Original rectified sinusoid
// Analog voltage measured from ADC
}
```