Lab 2 Analog Output (PWM), Timers and Analog Input

Simon Hu

yh4970

1. Background

Embedded systems integrate digital logic into real-world applications, bridging the gap between digital computation and analog signals prevalent in our environment. Since most physical phenomena are analog—varying continuously over a range—there arises a fundamental requirement for systems that convert analog signals into digital data and vice versa.

Microcontrollers, such as the Atmel ATMEGA32U4 used in this lab, typically incorporate built-in Analog-to-Digital Converters (ADC) to transform analog signals into digital format. However, many microcontrollers, including the ATMEGA32U4, do not contain a dedicated Digital-to-Analog Converter (DAC). To circumvent this limitation, Pulse Width Modulation (PWM) can be employed to simulate DAC functionality.

PWM generates an analog voltage level by switching digital outputs on and off at high frequencies, where the resulting voltage is proportional to the duty cycle. By adjusting the PWM duty cycle in software, various analog voltage levels can be approximated. To smooth out these rapid digital transitions into a stable analog signal, an external RC low-pass filter is utilized.

2. Experimental Procedure

2.1 Direct Digital Synthesis

To set up the PWM signal, the 1Hz sine wave with amplitude of 3V is created. It uses the millis() function in order to return the number of milliseconds elapsed starting from the program. The sin wave can be expressed as the following formula:

$3 * \sin\left(\frac{2\pi t}{1000}\right)$. The data passed into the DAC is the absolute value of the previously calculated sin wave value. The code is shown here:

```
float aval;

long x;
```

```
x = millis();

aval =abs(3 * sin(2 * 3.141592654 * x / 1000));

delay(1);
```

2.2 PWM DAC

Using the value generated in the previous step, a PWM frequency of 1 KHz, with the duty cycle proportional to the value, is created here. Because the PWM output is on D3 on the Playground classic, with corresponding 0C0B pin used, the timer 0 is used here. OCR0A is used at top register enabling the auto reset of OCR0B value to 0.

According to the following formula (1), the PWM frequency is calculated.

$$f_{PWM} = \frac{f_{cpu}}{Clk_{ps} * (OCR0A + 1)}$$

The frequency of CPU here is 8MHz, the prescaler is 64, and OCR0A is set as 125.

The control registers TCCR0A and TCCR0B are set to Fast PWM mode, Clear-on-match, with OCR0A at top and prescaler at 64. The code is shown below.

```
TCCR0A = 0b00100011;

TCCR0B = 0b00001011;

OCR0A = 125;

OCR0B = 0;

// output

DDRD |= ( 1 << 0);
```

The duty cycle starting in the PWM waveform is at 0%, in order to make its duty cycle proportional to the sin wave value, it needs to be controlled by OCR0B. The code in loop is shown below.

```
OCR0B = aval * (OCR0A / 3);
```

The value of sin wave is divided by three so that the generated wave cannot exceed maximum value (3V). As the OCR0B value increasing or decreasing with aval, when it hits maximum, it would be reset to 0. This corresponds to PWM duty cycle change.

Then a constructed low-pas filter including a 10kΩ resistor and 1μF capacitor in series is connected to the output.

The frequency of the PWM signal is 1kHz and the frequency of the analog signal is 1Hz. The Cutoff frequency of the RC filter is 15.92 Hz. The PWM signal (1 kHz) is significantly higher than the cutoff frequency (approximately 15.92 Hz), ensuring it is effectively filtered out. The analog signal frequency (1 Hz) is well below the cutoff frequency, ensuring that the desired analog waveform is preserved with minimal distortion.

The scope measurement of the filtered PWM signal is shown in Figure 1.

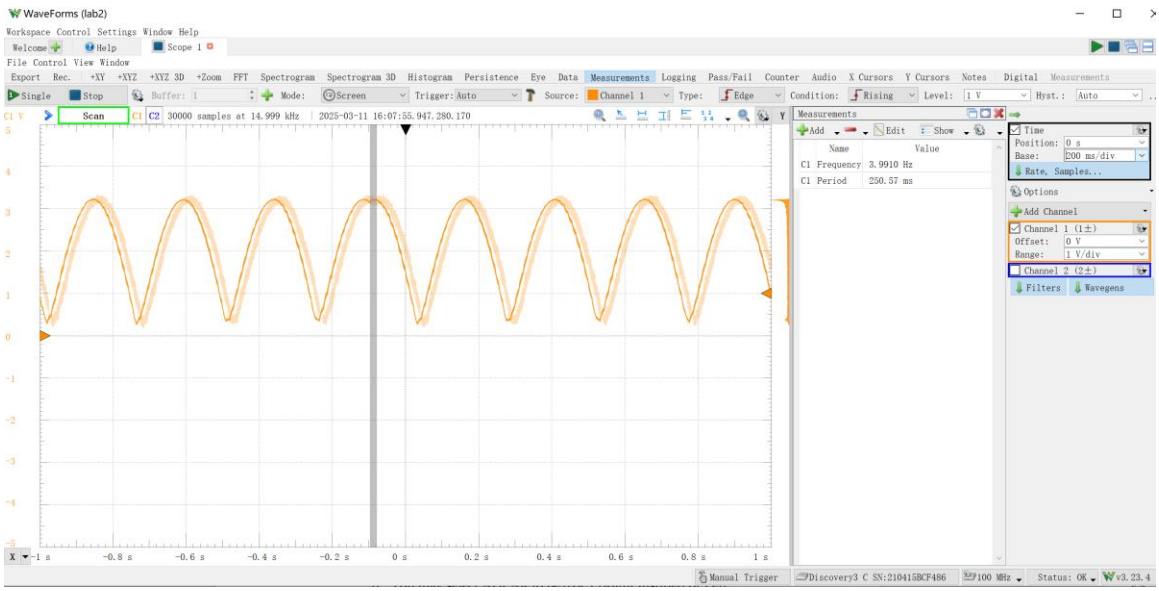

Figure 1: Oscilloscope Measurement of filtered PWM signal

There is some error in the scope data. The frequency should be 2Hz, but it is 3.99 Hz, which is almost twice as expected. From this, it is reasonable to conclude that there is a factor of 2 error present in the digital synthesis calculations.

2.3 ADC

The ADC input pin is D12 since ADC 9 is used here. The ADC was run in free running mode, with a prescaler division of 128 and high-speed mode. The code for configuring the ADC is shown below.

```
// ADC

ADMUX = 0b01000001; // 100001
```

```
ADCSRA = 0b11100111; // 128 pre

ADCSRB = 0b00100000;

DIDR2 |= (1 << 1);
```

The ADC data is read by 0x78 in 16 bits. The value is then multiplying by the maximum voltage of 3 voltages. The code is shown here.

```
unsigned short *ADCData;

unsigned short ADCVal;

ADCData=(unsigned short *)0x78;

ADCVal=(*ADCData & 0x3FF);


float fADCVal;

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

The final step is reading the data and comparing it to original input sin wave value. The code is shown here

```
Serial.print(abs(aval)); //Original rectified sinusoid

Serial.print(" ");

Serial.println(fADCVal); //Analog voltage measured from ADC
```

The final voltage is almost same as the initial value. This proves that the ADC is working.

3. Conclusion

In this lab exercise, a rectified analog sine wave was successfully generated and encoded into a PWM signal. This digital PWM signal was output via a microcontroller pin and then demodulated into an analog waveform using an RC low-pass filter composed of a 10kΩ resistor and a 1μF capacitor. The resulting analog signal was subsequently digitized using the built-in ADC of the microcontroller, demonstrating the complete analog-digital conversion cycle. Although the process generally worked as intended, discrepancies such as an observed frequency of approximately

4 Hz instead of the expected 2 Hz indicated a possible software calculation or timer configuration error. These observations highlight the importance of carefully verifying both hardware connections and software parameters during embedded system implementations to ensure accuracy and reliability.

4. Complete Code Listing

C: > Users > Simon > Documents > PlatformIO > Projects > ece4144lab2 > src > main.cpp > setup()

```cpp
#include <Arduino.h>
void setup() {
// run once:
Serial.begin(9600);

// pwm timer 0 fast pwm, 1khz, OCRA top
TCCR0A = 0b00100011;
TCCR0B = 0b00001011; // 64 presclar

OCR0A = 125;
OCR0B = 0;
// output
DDRD |= ( 1 << 0);

// ADC
ADMUX = 0b01000001; // 100001
ADCSRA = 0b11100111; // 128 pre
ADCSRB = 0b00100000;
DIDR2 |= (1 << 1);
}
void loop() {
// run repeatedly:
float aval;
long x;
x = millis();
aval =abs(3 * sin(2 * 3.141592654 * x / 1000));
delay(1);

// pwm duty

delay(20);
OCR0B = aval * (OCR0A / 3);

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

}
```