EE4144 – Intro to Embedded Systems

Lab Exercise 2

Analog Output (PWM), Timers and Analog Input

Objective:

This exercise demonstrates how to use timers to produce a simple PWM output signal on the Atmel 32U4 using ANSI C and how to demodulate the signal back to a true analog signal using a simple RC low-pass filter. After demodulation, the signal will then be fed back into an analog input of the controller. This part of the lab involves performing an analog-to-digital conversion of the demodulated signal by configuring the appropriate ADC registers in the microcontroller and verifying the conversion using the built in Serial debugging.

Setup:

The following equipment will be required for this lab exercise:

1. Adafruit Playground Classic
2. Interconnect wires (with alligator clips) available in Maker Space
3. 2 Channel digital storage Oscilloscope (Digilent Discovery)
4. A 10k Ohm resistor and a 1uF Capacitor (available in Maker Space)

Setting up the Analog Signal:

To set up the PWM signal we will use a rectified 1Hz sine wave with amplitude 3V. In setup () the loop() function you can add the following:

```
void setup() {
  // put your setup code here, to run once:
Serial.begin(9600);
}

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));
delay(1);
Serial.println(val);
}
```

→(0) Show the lab instructor the Serial Terminal output of the above to demonstrate that the correct sinusoid is being generated.

Setting up the PWM Signal:

In this step we will use the val "aval" in our sketch to generate a PWM output with the following characteristics:

1. PWM frequency of 1kHz (should be within 1Hz) and duty cycle proportional to "aval"
2. Fast PWM Mode
3. PWM Output on D3 (SCL) on the Playground Classic (0C0B, see schematic on Adafruit's website)

→(1) Explain which timer must be used for this PWM setup.

Now, using ANSI C, configure the DDR, TCCRxA ,TCCRxB , OCRxA, and OCRxB (if necessary) registers in the setup() function. (Addresses are in the Atmel 32U4 datasheet)

→(2) Show the code that sets up these registers and explain your bit selections.

Download your code to the Playground Classic and confirm that there are no errors.

→(3) Have your lab instructor verify using an oscilloscope (channel 1) that output pin #3 is indeed the correct PWM and has the correct characteristics as previously described.

Setting up the Demodulator:

Now, set up a single pole RC low-pass filter by putting a 10k Ohm and a 1uF capacitor in series connected from the #3 pin to ground. See Figure 1. on the following page.

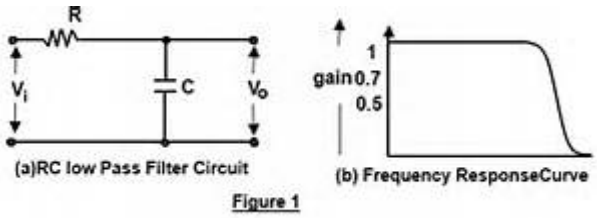

Figure 1

Recall the bandwidth of such a filter is 1/RC. In order for this to be an effective demodulator, the frequency (bandwidth) of the original signal must be in the pass band, that is, much less than 1/RC. Also, the frequency of the PWM signal must be in the reject band, that is, much more than 1/RC

→(4) Indicate the frequency of the PWM signal, the frequency of the analog signal, and the cutoff frequency (1/RC) of the filter. Confirm that the values are adequate.

Connect Vo (in diagram) to channel 2 of the oscilloscope.

→(5) Show the lab instructor the trace and the P-P and frequency values for channel 2. Is this analog signal a good representation of the original analog signal? Explain.

Setting up the Analog Input:

Now we will connect Vo to analog input 9 (ADC9) of the Playground Classic. We want to read the analog values and use the built in ADC to quantify the analog input.

First we need to set up the following registers:

1. Set up the ADC9 pin (#12) (Port D, pin 6) to an input (DDR)
2. Configure the ADMUX register
    a. Use Vcc as the reference since the voltage level is between 0-3V.
    b. You may select ADLAR to be 0 or 1 (Right Justified or Left Justified)
3. Configure ADCSRA and ADCSRB
    a. You may use free running mode if you wish
    b. Set an appropriate pre-scaler so that the register is updated at a sufficient frequency.
    c. Make sure to enable Start Conversion (ADSC)
4. Configure the DIDR register to disable digital access to pin ADC9

→(6) Show the lab instructor the code to set up the 5 registers described above.

In your loop function, read the ADC value in ADCL and ADCH (0x78 and 0x79). Make sure you read it correctly depending on the ADLAR bit. If ADLAR= 0, use:

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

If ADLAR=1, use:

```
unsigned short *ADCData;
unsigned short ADCVal;
ADCData=(unsigned short *)0x78;
ADCVal=(*ADCData & 0xFFC0) >> 6;
```

The short value will be in the range 0 to 1023. This range will be linearly mapped to the analog voltage range that you used when you set up the PWM signal. For example, if your PWM duty cycle was mapped from 0 (0% duty cycle) to 3V (100% duty cycle), then the analog conversion would be:

Measured analog voltage = (ADC register / 1023 * 3)

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

Add Serial.print() lines (such as the ones below) in the loop() function to print both values.

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

Download your new code and open the serial monitor. To inspect the data you can toggle the "scroll" function. Do the two data points have approximately the same value?

→(7) Show the lab instructor your final code and the serial output described above

Conclusion:

In this lab we began with a rectified analog sine wave and encoded it using PWM. The digital PWM signal was sent out a digital pin to an external LPF (Low Pass Filter) where the PWM is demodulated back into an analog signal. We then input this analog signal to an analog pin of our controller and use the ADC to quantify it digitally. We then map the 10 bit output of the ADC to an analog voltage which should indeed be close to the original rectified sinusoid. This demonstrates the complete analog capabilities of our processor.