WORCESTER POLYTECHNIC INSTITUTE
ROBOTICS ENGINEERING PROGRAM

Lab 1 - Intro to AVR

SUBMITTED BY
Joseph Agresta
Mead Landis
Casey Kracinovich

Date Submitted: 1/27/16
Date Completed: 1/27/16
Course Instructor: C. Putnam
Lab Section: RBE 3001 C'16

Abstract

In this lab, familiarity of the hardware and programming of the ATmega644 AVR processor was increased through the implementation of analog-to-digital conversion (ADC), USART serial communication, timer interrupt, and pulse width modulation (PWM) techniques. The experiment was successful in actualizing each system through programming the AVR processor in the Eclipse development environment using the C programming language.

Introduction

To synthesize fully functional embedded systems using a processor such as the ATmega644, it is necessary to understand how to communicate with a processor to control hardware devices through programming. This lab served to provide background in embedded programming and to build a basis for further work with the AVR processor. By interacting directly with the data registers on the AVR, desired functions were able to be constructed for time critical systems that were used with a variety of applications including, reading analog voltage values, communicating serially with a PC, and using timer interrupts to both sample and produce time accurate waveforms. Through USART, serial communication with a PC was made possible to provide diagnostic support as well as methods of logging data. The ADC allowed for the sampling of waveforms and the measurement of analog sensors such as a potentiometer. Each system was further supported by the use of timer interrupts that provided a technique to carry out operations in a quick and time accurate manner

Methodology

To carry out the desired goals of the lab, three major systems were implemented: the ADC, Timer/Counter0, and the USART serial communications system. The three systems were then combined to perform pulse width modulation, sampling of analog signals, and logging of data.

Analog-to-Digital Converter

On the AVR processor, the ADC is capable of sampling at up to 15 kSPS with any of four selectable reference voltages. To interface with the ADC, the analog input channels on Port A were used to sample analog voltage inputs. The ADC was configured by the setting of three registers: ADMUX, ADCSRA, and ADCSRB. ADMUX controls the source of the samples (which pins in Port A), the voltages they are compared to, and the justification/left adjustment of the output data. The justification is of note because it determines how the 10 bit result is stored in two 8 bit registers; ADCL and ADCH. ADCSRA, the first control and status register, allows the enabling of the chip, the starting of a conversion, the enabling of auto triggering, interrupts (including the interrupt flag), and lastly the ADC prescaler select bits. ADCSRB, the second control and status register, allows the auto-trigger source to be selected, and generally the mode of the ADC. Aside from free-running mode, options pertain to sources of events that result in a rising edge when conversion is intended to begin.

For the purposes of this lab, ADMUX was set in the function initADC(), that was implemented from RBELib, to use the voltage AREF (+5V) instead of an internal reference voltage. ADMUX was also selected to not use the left adjust (ADLAR) and could be customized when calling the function to be used on a specific input channel. ADCSRA was additionally set with the ADC enabled, the start conversion bit activated, auto trigger enabled, the ADC interrupt enabled, and the clock divider set to 128. ADCSRB was configured to use free running mode that would cause conversions to occur continuously.

Timer/Counter 0

Like the ADC, the timer was also configured using its registers. On the AVR, Timer/Counter0 is an 8-bit timer capable of firing an interrupt on overflow or when a compare value is matched by the timer count value. For the desired applications, overflow mode was used exclusively because of its simplicity. The timer, Timer0, is controlled via several registers; TCCR0A, TCCR0B TIMSK0, and OCR0A. TCCR0A provides selection of two compare match output (CTC) modes as well as waveform generation. Since these applications were not desired, the entire register was set to 0x00. TCCR0B controls similar functions but in addition also controls the I/O clock divider select. In this case the divider was set to 8. The clock source, I/O clk, was found to equal the system clock speed divided by 256. With an addition divider of 8, the 18,432,000 Hz system clock was divided down to 9000 Hz. The TIMSK0 register was found to be necessary to enable overflow mode interrupts and was set accordingly such that when global interrupts were enabled, the timer interrupt would fire correctly. Finally the OCR0A register was looked at but was largely not needed for overflow interrupt mode. Once configured, the timer was used with an interrupt service routine (ISR) in the main program that would increment a counter called ‘time’ every 1/9000th of a second. This time value was further divided down to provide for a flexible timing mechanism for sampling and pulse width modulation.

USART Serial Communications

A universal synchronous/asynchronous receive/transmit (USART) system was implemented to carry out serial communications between the AVR and the PC, as well as to allow for data logging and transmission of diagnostic messages. Serial communications were also used to start one program by waiting for the AVR to receive a character input in PuTTY from the PC keyboard. The USART was configured using the debugUSARTInit() function also implemented from RBELib’s USARTDebug.h. The function, along with confirming a valid baudrate, was used to set the registers necessary to configure the USART system. The registers that were used were UCSR1A, UCSR1B, UCSR1C, and UBRR1. UCSR1A is mostly used to indicate errors and when transmissions occur, however, the register also can select double transmission speed mode and was set to not use double speed. UCSR1B was used for USART1 to control character size, enable receiving or transmitting, as well as interrupt enables for the USART. UCSR1B was set to enable transmit and receive purely. Control register C (UCSR1C) was also set to select 8-bit character size, 1-stop bit, and asynchronous mode with parity disabled. The last register to set was UBRR1 which configured the baudrate for serial communication. It was set to produce a desired 115,200 Hz baudrate. With the methods (functions) implemented for USARTDebug,h, the standard printf() could then be used to transmit data serially and have it output to the terminal in PuTTY.

Results

For the first portion of the experiment, the ADC functionality was realized in reading the analog voltage value from a potentiometer. The program involved called the function, getADC(4), which read a value from the potentiometer connected to analog input 4, (a 10-bit value from 0-1023), bit shifted the high and low registers into one integer, and set a variable called ADCcount with the corresponding value. Using casts to long integers to perform operations on large numbers, the ADCcount value was changed into a value in millivolts (0-5000), and a potentiometer angle (0-270). All of the fields were printed to the terminal using printf() with an included timestamp of millisecond accuracy.

For the next program, (SquareWaveTest.c), the pins on Port D were configured as inputs and the ribbon cable was used to connection the buttons/switches bus to Port D. Port B was also configured as an output and connected to the LED bus. Timer0 was also initialized and set to have two compare value ceilings (although not used in compare mode, a similar technique was used). When a button was pressed, the modulus divider for the timer ISR was changed to create waveforms of different frequencies and 50% duty cycle. Using the two separate ceiling values, this program was later modified to provide an adjustable duty cycle. PWM was able to be produced at 1 Hz, 20 Hz, and 100 Hz frequencies with a duty cycle range from 10-90%.

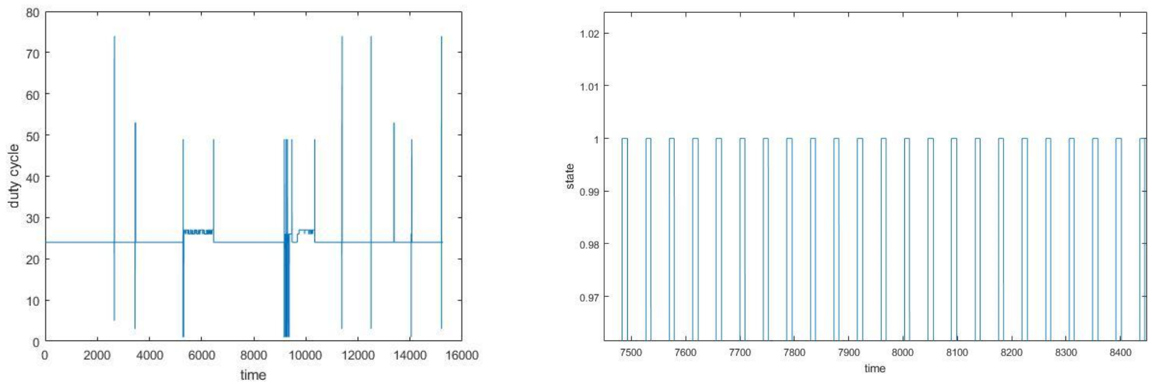
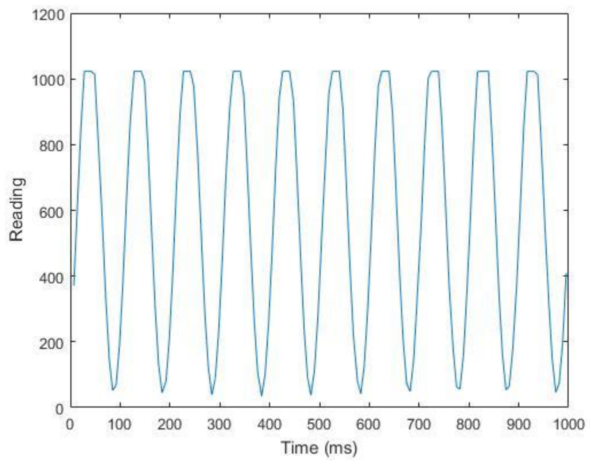


Figure 1: Left: duty cycle maintained within 1%. Right: Corresponding state output

An additional separate source program was made (SignalSample.c) to sample analog signals using the ADC. This program simplified the technique to read the ADC and stored ADC values in an array (called 'data') that would later be output to PuTTY and logged. The timer modulus divider used earlier was removed and the timer was set up to go at its natural 9000 Hz frequency (natural to us). Separate modulus dividers were added in the while loop to control the sampling frequency and were configured in separate attempts to frequencies of 225 Hz, 140 Hz, 125Hz, and 20 Hz. The program also toggled the Port B pins (set as outputs) to produce a 0.5x sampling frequency waveform. For each sampling frequency, the sampling period was held constant at 1s (9000 cycles) which produced less samples for slower sampling frequencies. A timestamp was also added as every other element of the data array for the ADC values to be recorded and timed in a more optimal manner. A push button from Port D was additionally configured to begin the sampling period. Screenshots of the sampling waveforms, sampled waveforms, and data logs were also recorded.



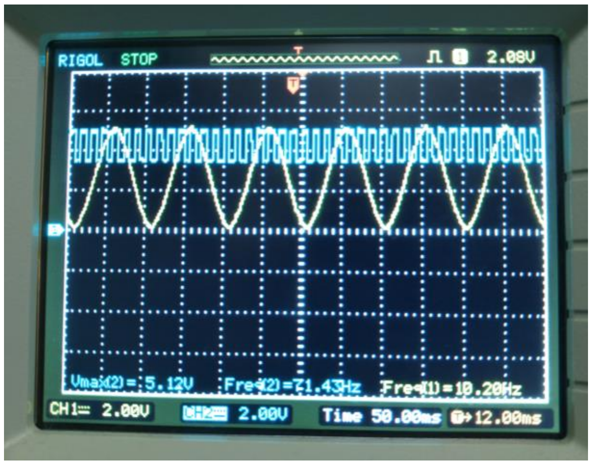


Figure 2: 10 Hz sine wave sampled at 140 Hz

Discussion

The lab functioned well as an exercise in perseverance, and was generally successful. The extremely steep learning curve was beneficial in simulation the multi-faceted challenge that is work in industry.The most notable points include:

Software:

Many issues were had in attempting to install all of the necessary software and drivers, in the end, it took us 2 days before all of us had functioning software.

RBELib Oversight:

The morning the lab was due, we were informed we had misused RBELib by making our own header files with the same naming conventions as RBELib. While this did not seem to impact performance, it may become an issue in the future, particularly when requesting assistance from others. To remedy this, we will be overhauling our current code to better conform to and make use of RBELib.

Code curation:

In attempting to make our code functional to the extent that we could program the board, it became apparent that the intended central code had failed in some undiscovered manner. Currently, our central code is denoted under a different name than originally intended. While this was to be a temporary measure until we finished the lab aspects, it remains an issue yet to be addressed.

ADC and other Code:

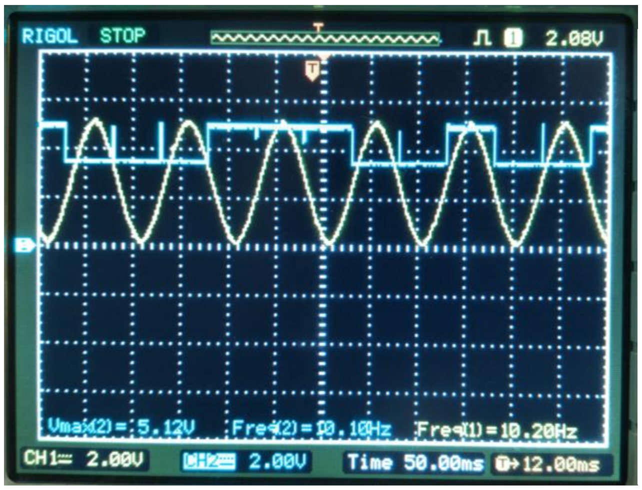
Through issues with eclipse as well as odd and interesting code problems, approximately 3-4 days of work were spent attempting to return to our previous amount of progress. The most interesting issue we solved was that no matter what revision we used, our ADC would only function at a very slow rate. It was discovered, after about 48 hours, 3 SA/TA's, and countless colleagues, that we had set the system clock prescaler sometime previously, but assumed it would default back to normal operation afterward. This proved to be false and explicitly setting the prescaler register was required.

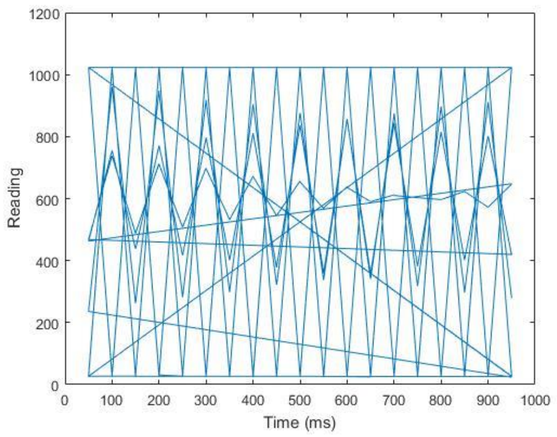
Latex Formatting

Learning Latex was placed at a lower priority than finishing the lab for this week. Additionally with this compromise we did not have a suitable amount of freedom to explore MATLab. We plan to pursue proficiency in both categories in the coming labs, after suitable progress during this lab.

Aliasing:

Our results were consistent with the theory that data sample rates should be more than double the highest frequency to be measured. When we analyzed our results using exactly double the input frequency, our graphs were mostly meaningless and chaotic, while our higher sample rates proved effective.





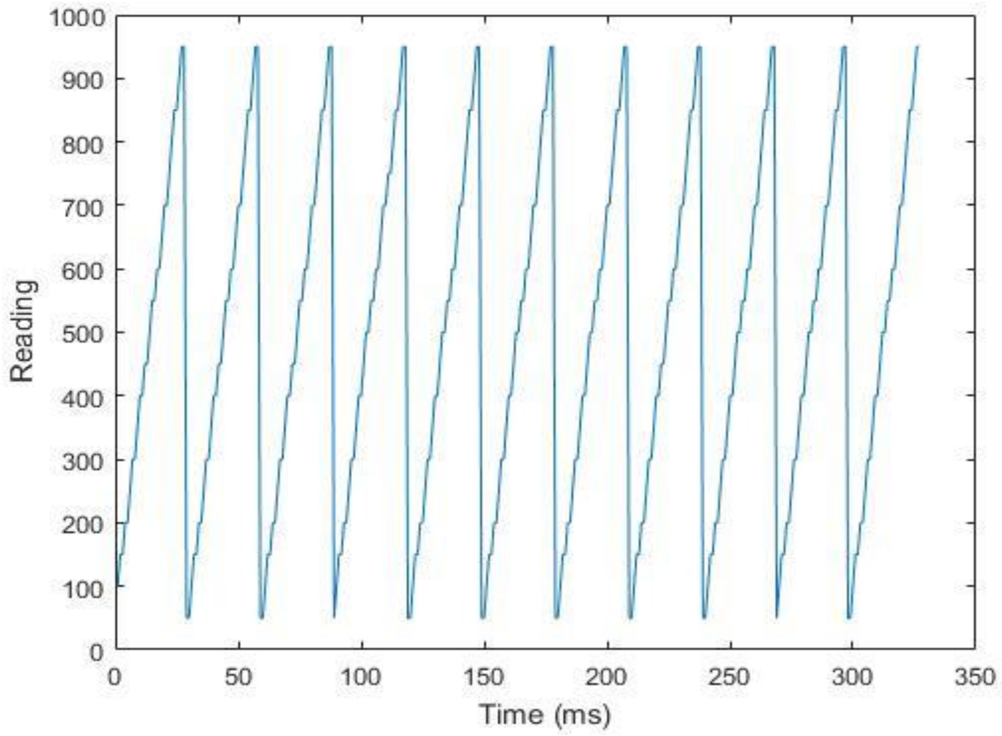


Figure 3: Left: 10 hz sine wave sampled at 20 hz   Right: aliasing occurring when time is used

Bottom: aliasing reduced by graphing against index instead of timestamp

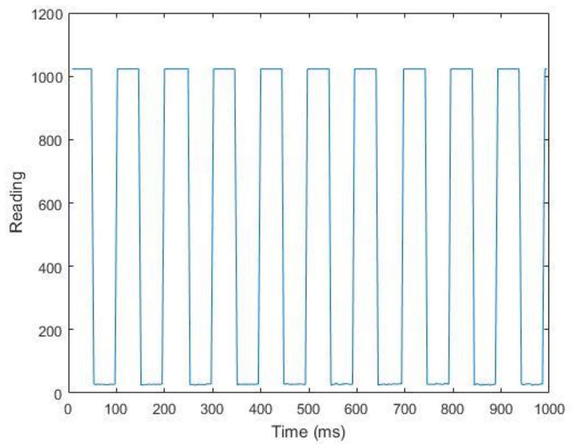
Conclusions:

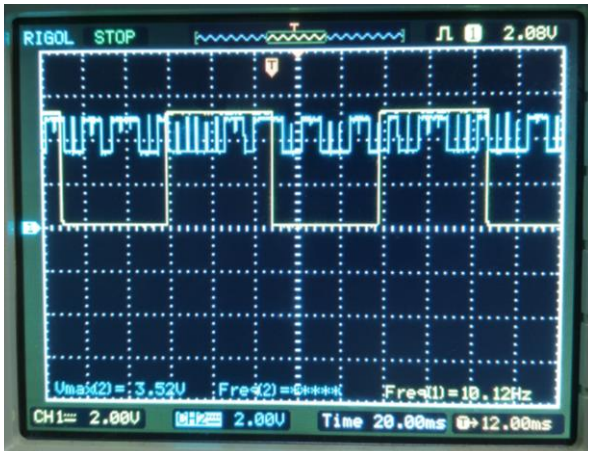
While many things were learned during the course of this lab, success was the clear set of standards to be met and objectives to be accomplished. By this measure, some aspects of this lab were abject failures.

Appendix A: Authorship

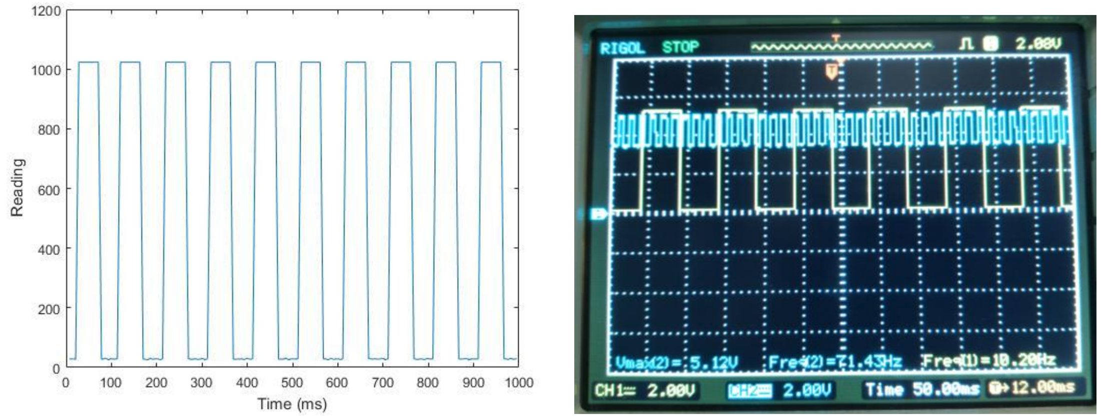
Section	Author
Introduction	Joe Agresta
Methodology	Joe Agresta
Results	Joe Agresta Mead Landis
Discussion	Mead Landis Casey Kracinovich
Conclusion	Mead Landis Casey Kracinovich

Appendix B: Figures

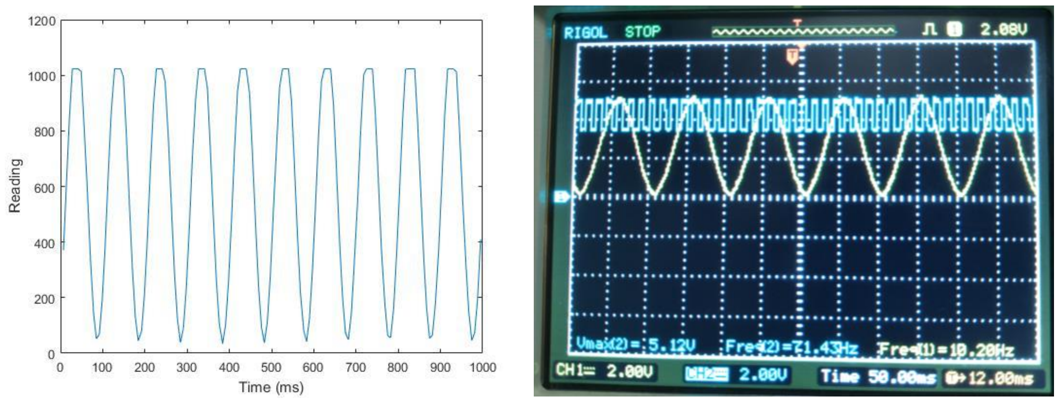




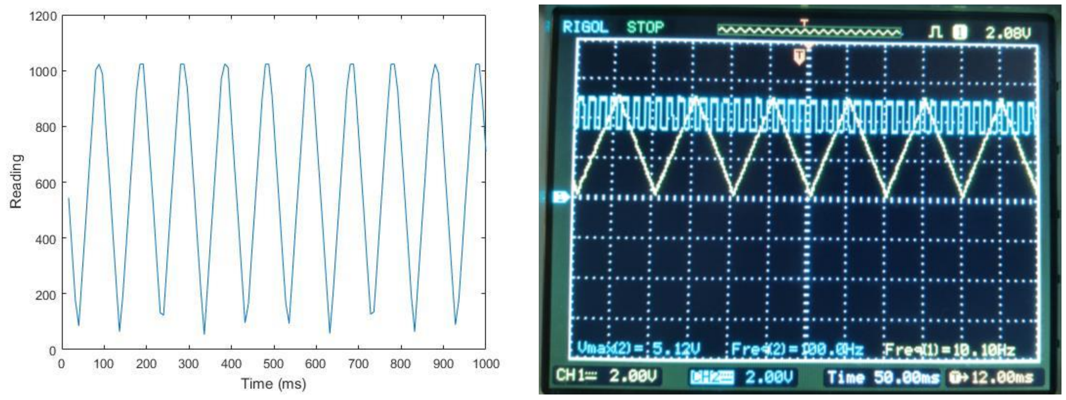
10 Hz square wave sampled at 225 Hz



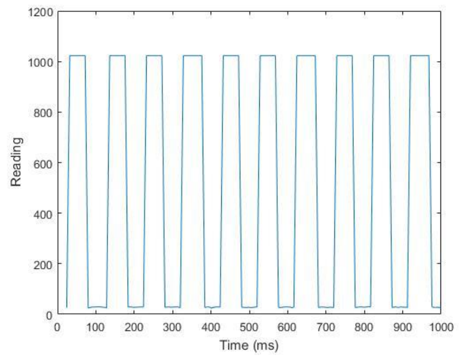
10 Hz square wave sampled at 140 Hz

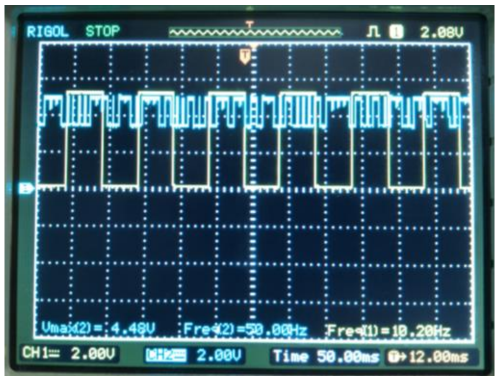


10 Hz sine wave sampled at 140 Hz

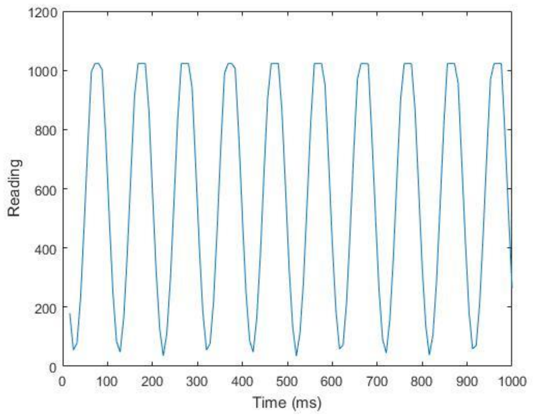


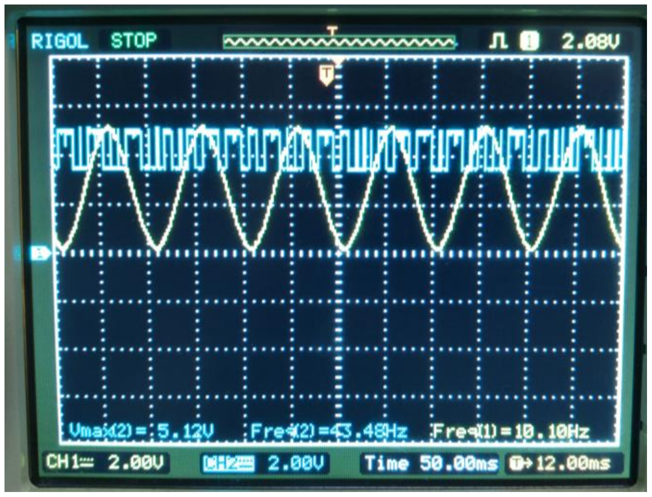
10 Hz triangle wave sampled at 140 Hz



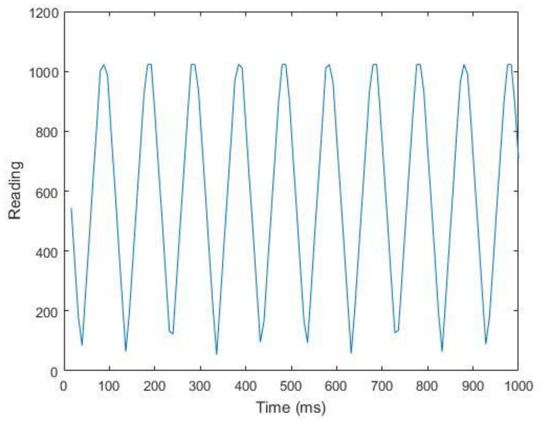


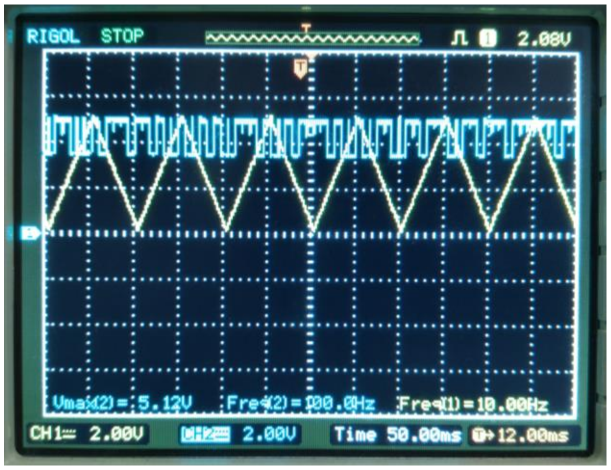
10 Hz square wave sampled at 125 Hz



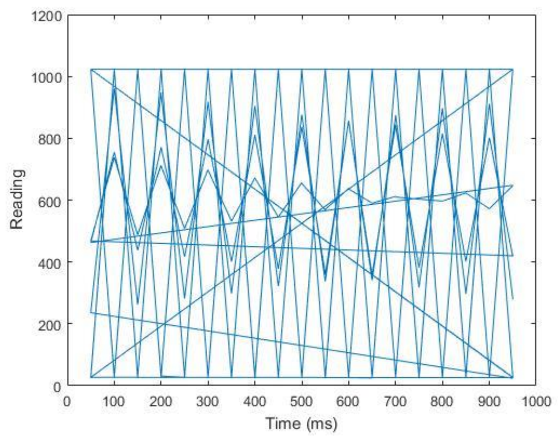


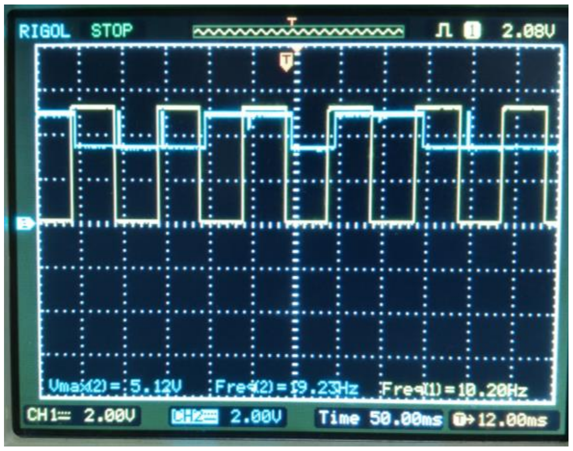
10 Hz sine wave sampled at 125 Hz



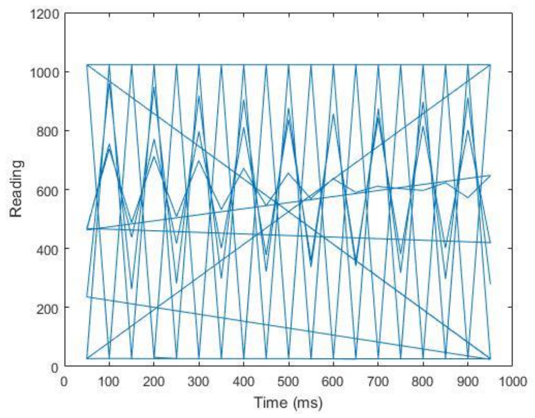


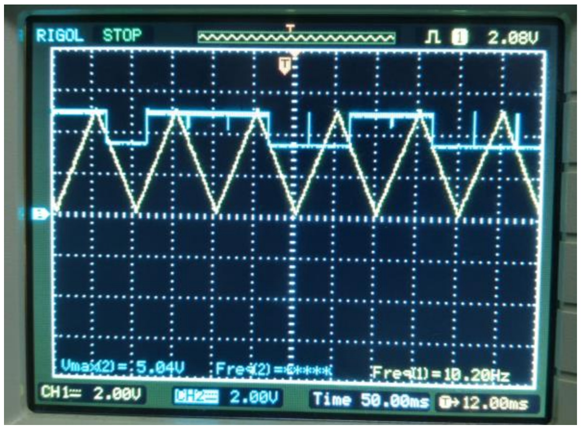
10 Hz triangle wave sampled at 125 Hz



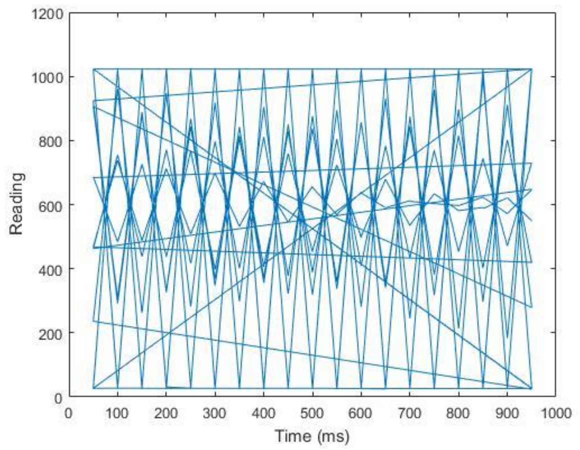


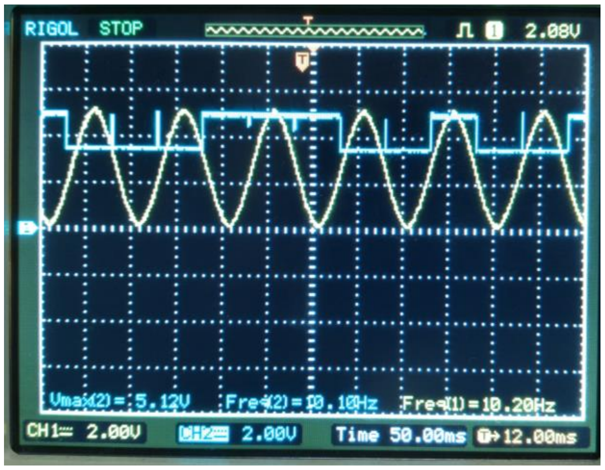
10 Hz square wave sampled at 20 Hz





10 Hz triangle wave sampled at 20 Hz





10 Hz sine wave sampled at 20 Hz