**University of Massachusetts Amherst**

**College of Engineering**

**Department of Electrical and Computer Engineering**

**ECE 231 - Introduction to Embedded System - Spring 2020**

**Instructors: B. Soules and F. Anwar**

**Lab Report**

**Lab 1**

**From C to C to C; programmed in C: being a Musical Signal Generator**

STUDENT A: Luan Vo

Spire ID: 30553600

STUDENT B: Jeffrey Matheson

Spire ID: 31587042

Report submission date: 17 February 2020

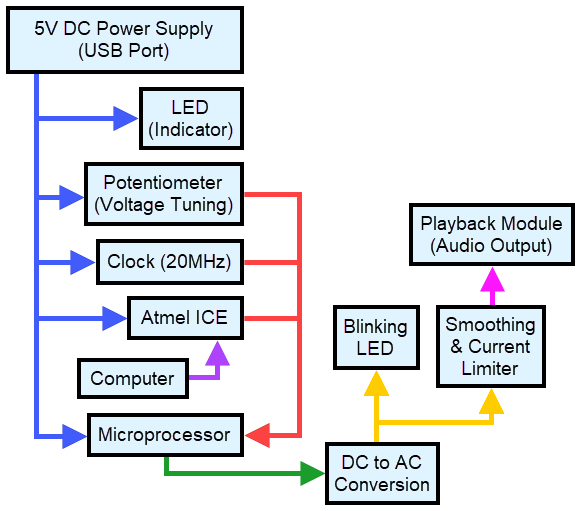
**Introduction:**

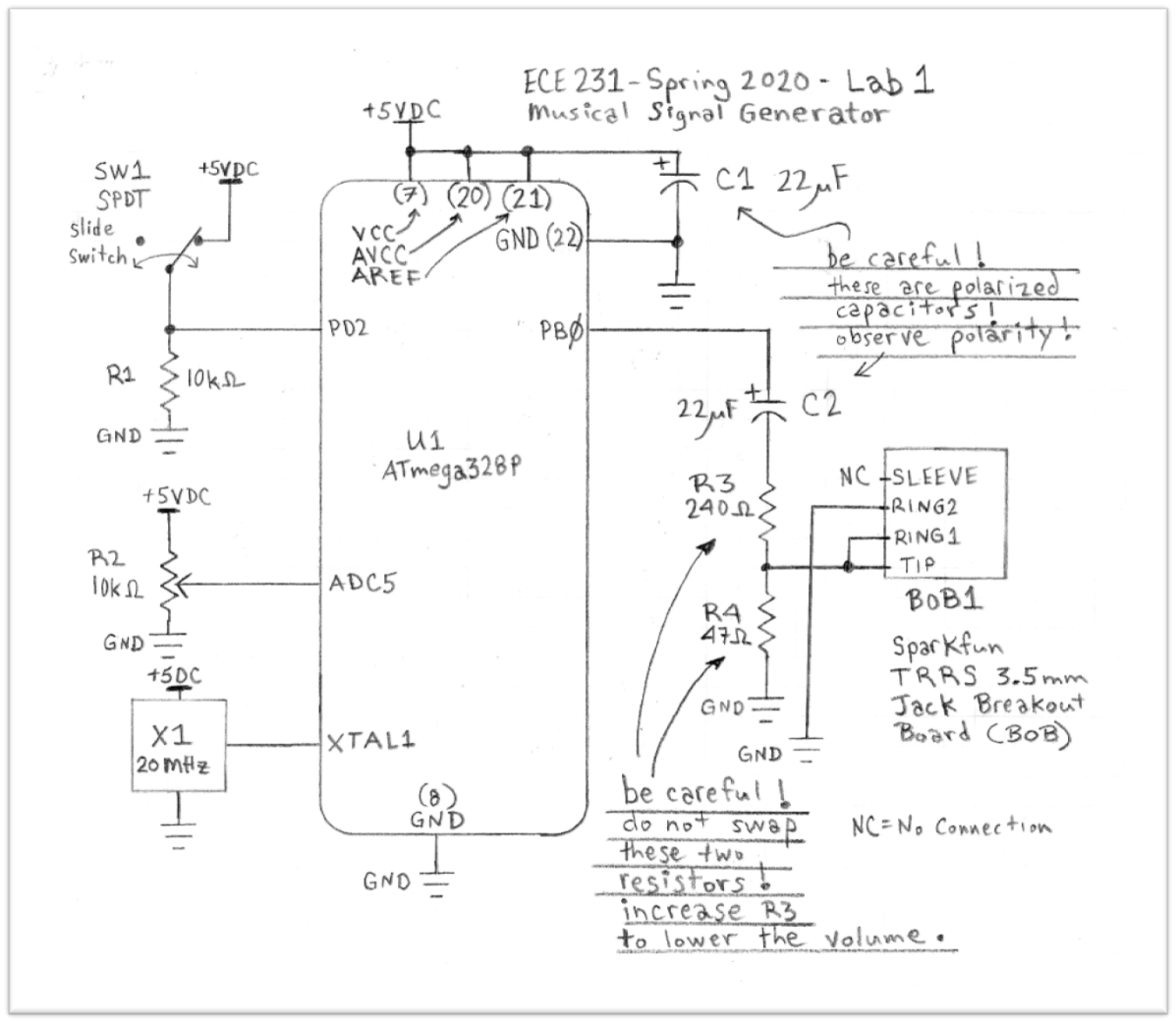
The objective of this lab is to be able understand the hardware ATmega328p microcontroller, its’ port and the topology of a simple ADC circuit. This lab allows us to get ourselves familiarized with C# assembly language and introduced us to working with technological datasheets and manuals, an experience that we can never have in a classroom environment. In addition, this lab is modelled after a real workplace, which I found very helpful, as when we set out to work, we will have to find and read these manuals ourselves. The project allows us to understand how inputs are being read and getting more hands-on experience with hardware which is usually not taught in classes. Ultimately, the project is about using ADC hardware and by changing the voltage via a potentiometer and with the help of the external clock, we can produce a desired frequency that will play a set of 25 notes.

**Materials and Hardware Description**

* **Main Components:**
* U1 ATmega328P microcontroller Digi-key Part Number: ATMEGA328P-PU-ND:
  + This is the 8-bit AVR microcontroller, with flash memory of 32KB, the ability to read-while-write, with 23 input pins and operational voltage range from 1.8V-5.5V. This microprocessor is responsible for the analog to digital conversion of the circuit board.
  + Sources[[1]](#footnote-1)
* 20 MHz clock (Abracon ACH-20.000MHZ-EK) Digi-key Part Number: 535-9173-5-ND
  + Instead of using the internal clock, we instructed the ATmega328p to use this external clock in order to let the circuit to be synchronized. The circuit relies on this 20MHz clock to know when and how to execute given instructions. This clock controlled the on and off state as well as the fluctuation in voltages possible within the circuit.
  + The crystal inside of the clock has been specifically cut to set its frequency to 20 MHz, the clock itself makes use of this crystal to send control signals to the microcontroller.
  + Source[[2]](#footnote-2)
* Atmel ICE debugger with Atmel ICE flat cable Digi-Key Part Number: ATATMEL-ICE-ND
  + This tool allows us to “tap in” and program the microprocessors.
* Adafruit USB micro-B breakout board (to power your circuit) Digi-key Part Number: 1528-1383-ND
  + This is the power source of the circuit that drives 5V into the circuit board. Since our microprocessor can only take inputs from 1.8-5.5V, USB powered source seemed to be the best choice.
* Solderless Breadboard Terminal Strip (No Frame) Digi-Key Part Number: 1528-2143-ND
  + This is the circuit board, where we performed the wiring of parts, and is the foundation of our lab.
* 10 kilo-ohm potentiometer Digi-key Part Number: 3310R-125-103L-ND
  + The potentiometer allows us to change the voltage or current going into the input pin of the microprocessor by changing the resistance. According the Charles Platt “When a voltage is applied across a potentiometer, it can deliver a variable fraction of that voltage. It is often used to adjust sensitivity, balance, input, or output, especially in audio equipment and sensors such as motion detectors.”
  + Source[[3]](#footnote-3)
* Resistors and Capacitors:
  + 22 microfarad electrolytic capacitor x 2
  + 10 kilo-ohm resistor Digi-key Part Number: 10KH-ND
  + 240 ohm resistor
  + 47 ohm resistor
    - Resistors: mainly use in the job of voltage divider as well as maintaining the currents running through the circuits and the components. Maintaining the proper current is important as we don’t want to burn too much power and fry anything in the circuit.
    - Sources[[4]](#footnote-4)
    - Capacitor job is to block DC currents while passing AC currents (i.e: pulses). It can also acts as small rechargeable batteries in circuit. Smoothening out the output voltage provided by power supplies.
    - Sources[[5]](#footnote-5)
* **Extra Components and Test Equipment:**
* SPDT slide switch (single pole, double throw) Digi-key Part Number: EG1903-ND
  + Extra component that was not necessary nor used in lab and our final product.
* Oscilloscope
  + To power and to measure the circuit’s voltage, frequencies.

**Schematics and Block Diagram:**





* **Circuit description:**
  + 5VDC inputs into pin 7, 20 and 21 which allows the ATmega328p to take in input voltages for the microprocessor to use.
  + The capacitor is used for smoothing out any noise, blocking DC so that only AC signals get passed.
  + Resistors prevent the current and power from exceeding the maximum limit of the various components which could permanently damage them.
  + The clock is used to synchronize the circuit, so that every component will run by at the designated frequency.
  + The potentiometer is used to change the input current or voltage to the microprocessor (which we used to change the note that the circuit played).
  + The switch was optional, and we did not use it for this lab.
  + The output pin PB0 is connected to capacitor, resistors and the earbud connector. The reason for this connection is the capacitor smoothing out the incoming signal (by eliminating any signal that is too high or too low from what we wanted, the frequency elimination is dependent on capacitors). Behind the capacitor are the two resistors, their job is to create a voltage divider and decrease the incoming current, meaning the higher R3 is the lower the volume, the lower R3 is the higher the volume. If R3 is too small, there is a risk of ruining the earbud as the earbud can only allow/accept so much current going through it.

**Software**:

* Software description:
  + For our final program, we used a while loop with 25 if and else statements in order to specify the range of each voltage with their associated hexadecimal numbers. Within those ranges, we turn the output (PB0) on and off to produce a square wave and this is done at specific delays that correspond to each of the 25 notes. These delays create different frequencies thus generating different notes.
    - Our Code:

if ((input >= 0x0000) & (input <= 0x0028))

{

//note: 1.25 is the fudge factor

PORTB = 0b00000001;

*\_delay\_us*(7645\*1.25);

PORTB = 0b00000000;

*\_delay\_us*(7645\*1.25);

}

* + - This code corresponds to the note C3. Here, we used the full period instead of half, and how we came up with the fudge factor (the \*1.25) will be explained in the ***problems encountered and solutions*** section***.***
  + We initialized ADMUX and ADCSRA in accordance to the ATMEL datasheet
    - ADMUX |= (1<<REFS0) | (1<<MUX2) |(0<<MUX1) |(1<<MUX0);

ADCSRA |= (1<<ADEN)|(1<<ADSC)|(0<<ADATE)|(0<<ADIE)|(1<<ADPS2) | (0<<ADPS1)| (1<<ADPS0);

* + - Through the use of left shift operators (i.e: <<), we shifted each input of ADCSRA by 1 to the left whenever ADCSRA is initialized.
    - Source[[6]](#footnote-6)
  + After that, we created an unsigned 16-bit integer to store the hex of ADC.
  + We keep the loop running by adding the statement inside the while loop, so that whenever the 6th bit of ADCSRA is not 1, left shift ADSC by 1 again. This is due to ADSC, the conversion bit, needed to be 1 in order to perform the initialization of ADC.
    - Our Code:
    - while(!(ADCSRA & 0b01000000)){

ADCSRA |= (1<<ADSC); }

* + We wrote this program based on the ATMEL datasheet and the C manual that was provided to us. (Source: ATMEL datasheet pg. 258)
  + This is the table that we modelled our program after:



**Problems Encountered and Solutions:**

* When we first started, we could not get access to the chip after setting the microprocessor to use the external clock, even though the circuit was correctly wired (the TA confirmed it and this was further proven later on when it worked with the same setup).
  + Solution: we solved it by changing the clock, and the chip then set the program to use the internal clock again. After which, we swapped out the clock and put the old chip back in. The problem seems to be that the old clock was damaged or just simply not functional.
* We were once again blocked out of our program the second time.
  + Solution: We connected the fuse to the wrong port, after we switched it back to AVR, everything worked out.
* As mentioned above, we used the full period because when we use the in-built function “\_delay\_us(microsecond)”
  + Solution: the frequency of the function is 1/10us = 10 MHz, but our external clock is 20 MHz, that is why we were off by the factor of 2. We realized this later and compensated the error by multiplying every half period by 2 which is why we used the full period.
  + Alternative Solution:
    - #define F\_CPU = 2000000UL

#include <util/delay.h>

* + - This will set the frequency of the delay function to be the same as the external clock. (which was tested and worked)
  + Because our codes have a lot of if and else statements, it created a slight delay in our output frequency, therefore the fudge factor was calculated to compensate for this redundancy.
    - Solution: we found the fudge factor of the lowest and highest voltage that gave us the range of [LOW, HIGH] = [1.25, 1.202], after we get the range for the compensation variable, we decrease each step by 0.02 from 1.25 to 1.202.

**Code:**

**Final Code**

#include <avr/io.h>

#include <stdio.h>

#include <stdlib.h>

#include <util/delay.h>

/\*void ADC\_init(void)

{

ADMUX |= (1<<REFS0) | (1<<MUX2) |(0<<MUX1) |(1<<MUX0);

ADCSRA |= (1<<ADEN) | (1<<ADSC) | (0<<ADATE) | (0<<ADIE) | (1<<ADPS2) | (0<<ADPS1)| (1<<ADPS0);

}\*/

int main (void)

{

ADMUX |= (1<<REFS0) | (1<<MUX2) |(0<<MUX1) |(1<<MUX0);

ADCSRA |= (1<<ADEN) | (1<<ADSC) | (0<<ADATE) | (0<<ADIE) | (1<<ADPS2) | (0<<ADPS1)| (1<<ADPS0);

DDRB = 0b00000001;

*uint16\_t* input = ADC;

while(1)

{

PORTB = 0b00000001;

*uint16\_t* input = ADC;

//Test: Full time

if ((input >= 0x0000) & (input <= 0x0028))

{

PORTB = 0b00000001;

*\_delay\_us*(7645\*1.25);

PORTB = 0b00000000;

*\_delay\_us*(7645\*1.25);

}

//Below cases are half time until commented otherwise

else if ((input >= 0x0029) & (input <= 0x0051))

{

PORTB = 0b00000001;

*\_delay\_us*(7215\*1.248);

PORTB = 0b00000000;

*\_delay\_us*(7215\*1.248);

}

else if ((input >= 0x0052) & (input <= 0x007A))

{

PORTB = 0b00000001;

*\_delay\_us*(6810\*1.246);

PORTB = 0b00000000;

*\_delay\_us*(6810\*1.246);

}

else if ((input >= 0x007B) & (input <= 0x00A3))

{

PORTB = 0b00000001;

*\_delay\_us*(6428\*1.244);

PORTB = 0b00000000;

*\_delay\_us*(6428\*1.244);

}

else if ((input >= 0x00A4) & (input <= 0x00CC))

{

PORTB = 0b00000001;

*\_delay\_us*(6067\*1.242);

PORTB = 0b00000000;

*\_delay\_us*(6067\*1.242);

}

else if ((input >= 0x00CD) & (input <= 0x00F5))

{

PORTB = 0b00000001;

*\_delay\_us*(5727\*1.240);

PORTB = 0b00000000;

*\_delay\_us*(5727\*1.240);

}

else if ((input >= 0x00F6) & (input <= 0x011E))

{

PORTB = 0b00000001;

*\_delay\_us*(5405\*1.238);

PORTB = 0b00000000;

*\_delay\_us*(5405\*1.238);

}

else if ((input >= 0x011F) & (input <= 0x0147))

{

PORTB = 0b00000001;

*\_delay\_us*(5102\*1.236);

PORTB = 0b00000000;

*\_delay\_us*(5102\*1.236);

}

else if ((input >= 0x0148) & (input <= 0x0170))

{

PORTB = 0b00000001;

*\_delay\_us*(4816\*1.234);

PORTB = 0b00000000;

*\_delay\_us*(4816\*1.234);

}

else if ((input >= 0x0171) & (input <= 0x0199))

{

PORTB = 0b00000001;

*\_delay\_us*(4545\*1.232);

PORTB = 0b00000000;

*\_delay\_us*(4545\*1.232);

}

else if ((input >= 0x019A) & (input <= 0x01C2))

{

PORTB = 0b00000001;

*\_delay\_us*(4290\*1.230);

PORTB = 0b00000000;

*\_delay\_us*(4290\*1.230);

}

else if ((input >= 0x01C3) & (input <= 0x01EB))

{

PORTB = 0b00000001;

*\_delay\_us*(4050\*1.228);

PORTB = 0b00000000;

*\_delay\_us*(4050\*1.228);

}

else if ((input >= 0x01EC) & (input <= 0x0214))

{

PORTB = 0b00000001;

*\_delay\_us*(3822\*1.226);

PORTB = 0b00000000;

*\_delay\_us*(3822\*1.226);

}

else if ((input >= 0x0215) & (input <= 0x023D))

{

PORTB = 0b00000001;

*\_delay\_us*(3608\*1.224);

PORTB = 0b00000000;

*\_delay\_us*(3608\*1.224);

}

else if ((input >= 0x023E) & (input <= 0x0266))

{

PORTB = 0b00000001;

*\_delay\_us*(3405\*1.222);

PORTB = 0b00000000;

*\_delay\_us*(3405\*1.222);

}

else if ((input >= 0x0267) & (input <= 0x028F))

{

PORTB = 0b00000001;

*\_delay\_us*(3214\*1.220);

PORTB = 0b00000000;

*\_delay\_us*(3214\*1.220);

}

else if ((input >= 0x0290) & (input <= 0x02B8))

{

PORTB = 0b00000001;

*\_delay\_us*(3034\*1.218);

PORTB = 0b00000000;

*\_delay\_us*(3034\*1.218);

}

else if ((input >= 0x02B9) & (input <= 0x02E1))

{

PORTB = 0b00000001;

*\_delay\_us*(2863\*1.216);

PORTB = 0b00000000;

*\_delay\_us*(2863\*1.216);

}

else if ((input >= 0x02E2) & (input <= 0x030A))

{

PORTB = 0b00000001;

*\_delay\_us*(2703\*1.214);

PORTB = 0b00000000;

*\_delay\_us*(2703\*1.214);

}

else if ((input >= 0x030B) & (input <= 0x0333))

{

PORTB = 0b00000001;

*\_delay\_us*(2551\*1.212);

PORTB = 0b00000000;

*\_delay\_us*(2551\*1.212);

}

else if ((input >= 0x0334) & (input <= 0x035C))

{

PORTB = 0b00000001;

*\_delay\_us*(2408\*1.210);

PORTB = 0b00000000;

*\_delay\_us*(2408\*1.210);

}

else if ((input >= 0x035D) & (input <= 0x0385))

{

PORTB = 0b00000001;

*\_delay\_us*(2273\*1.208);

PORTB = 0b00000000;

*\_delay\_us*(2273\*1.208);

}

else if ((input >= 0x0386) & (input <= 0x03AE))

{

PORTB = 0b00000001;

*\_delay\_us*(2145\*1.206);

PORTB = 0b00000000;

*\_delay\_us*(2145\*1.206);

}

else if ((input >= 0x03AF) & (input <= 0x03D7))

{

PORTB = 0b00000001;

*\_delay\_us*(2025\*1.204);

PORTB = 0b00000000;

*\_delay\_us*(2025\*1.204);

}

else if ((input >= 0x03D8) & (input <= 0x03FF))

{

PORTB = 0b00000001;

*\_delay\_us*(1911\*1.202);

PORTB = 0b00000000;

*\_delay\_us*(1911\*1.202);

}

//Set the ADSC bit to 1 again when it’s not

while(!(ADCSRA & 0b01000000))

{

ADCSRA |= (1<<ADSC);

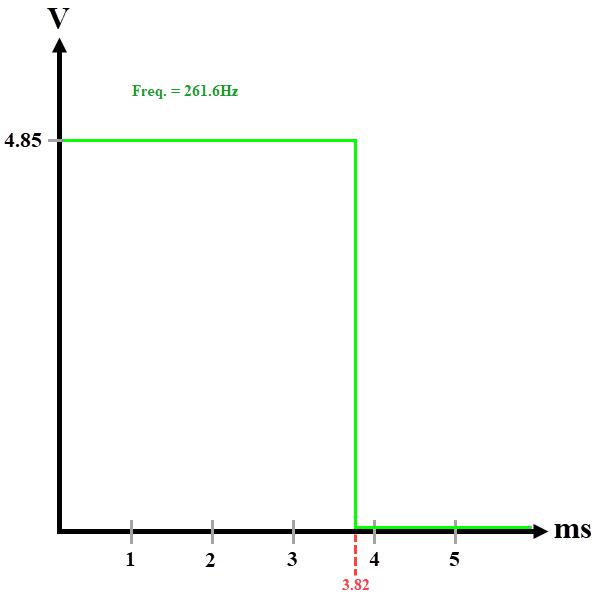
}

}

}

**Conclusion:**

After this project, we felt that we learned a lot, because in the actual workplace, no one will be providing us with instructions and we will have to utilize our previous knowledge and experiences to read through these manuals and apply new found knowledge to our circuits. We had a lot of fun with this project, and with what we learned by manipulating the on and off state, we essentially created a frequency which will then be able to turn into sound. We also made the Mario theme song and another song titled *Megalovania* as well as syncing the LED with the output frequency to make it flash at the same frequency as the sound output. There are a lot of ways that we can improve our work, such as using the counter function to reduce the redundancy in our codes, because as of now, our code is what is currently making the time a bit off compared to what it should be.



**References:**

Abracon, *Half Size DIP Low Voltage 5.0V Crystal Clock Oscillator Datasheet* (Abracon, 2016). <https://abracon.com/Oscillators/ACH.pdf>

Charles Platt, *Encyclopedia of electronics components Vol. 1* (O’REILLY 2013) <https://media.digikey.com/pdf/Data%20Sheets/O'Reilly_PDFs/Encyclopedia%20of%20Electronic%20Components%20Volume%201_9781449333898.pdf>

Microchip, *ATmega48A/PA/88A/PA/168A/PA/328/P megaAVR® Data Sheet* (Microchip Technology Inc., 2018) <https://piazza.com/class_profile/get_resource/k5r5dqqc6ia12n/k5zo787vk1cfh>

1. Microchip, *ATmega48A/PA/88A/PA/168A/PA/328/P megaAVR® Data Sheet* (Microchip Technology Inc., 2018) 16, 286 [↑](#footnote-ref-1)
2. Abracon, *Half Size DIP Low Voltage 5.0V Crystal Clock Oscillator Datasheet* (Abracon, 2016). [↑](#footnote-ref-2)
3. Charles Platt, *Encyclopedia of electronics components Vol. 1* (O’REILLY 2013) 89 [↑](#footnote-ref-3)
4. Charles Platt, *Encyclopedia of electronics components Vol. 1* (O’REILLY 2013) 75 [↑](#footnote-ref-4)
5. Charles Platt, *Encyclopedia of electronics components Vol. 1* (O’REILLY 2013) 97 [↑](#footnote-ref-5)
6. Microchip, *ATmega48A/PA/88A/PA/168A/PA/328/P megaAVR® Data Sheet* (Microchip Technology Inc., 2018) 257 [↑](#footnote-ref-6)