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: Bradford Gill
Spire ID: 31871838

STUDENT B: Ashton Gray
Spire ID: 32000589

Report submission date: 16 February 2020

Introduction:

This lab is meant to introduce students to fundamental computer aspects. Including but not limited to setting and programing RAM, understanding the relationship between a clock and cpu, visualizing the crossover between software and hardware, using the C programing language, and referencing a data sheet to understand how to complete a task. The student will go through 4 tests, each will help the student get closer to finishing the project. The task is to set up a cpu on a breadboard with the necessary components and then write a program that will output a musical note.

Materials and Hardware Description:

Fig 1

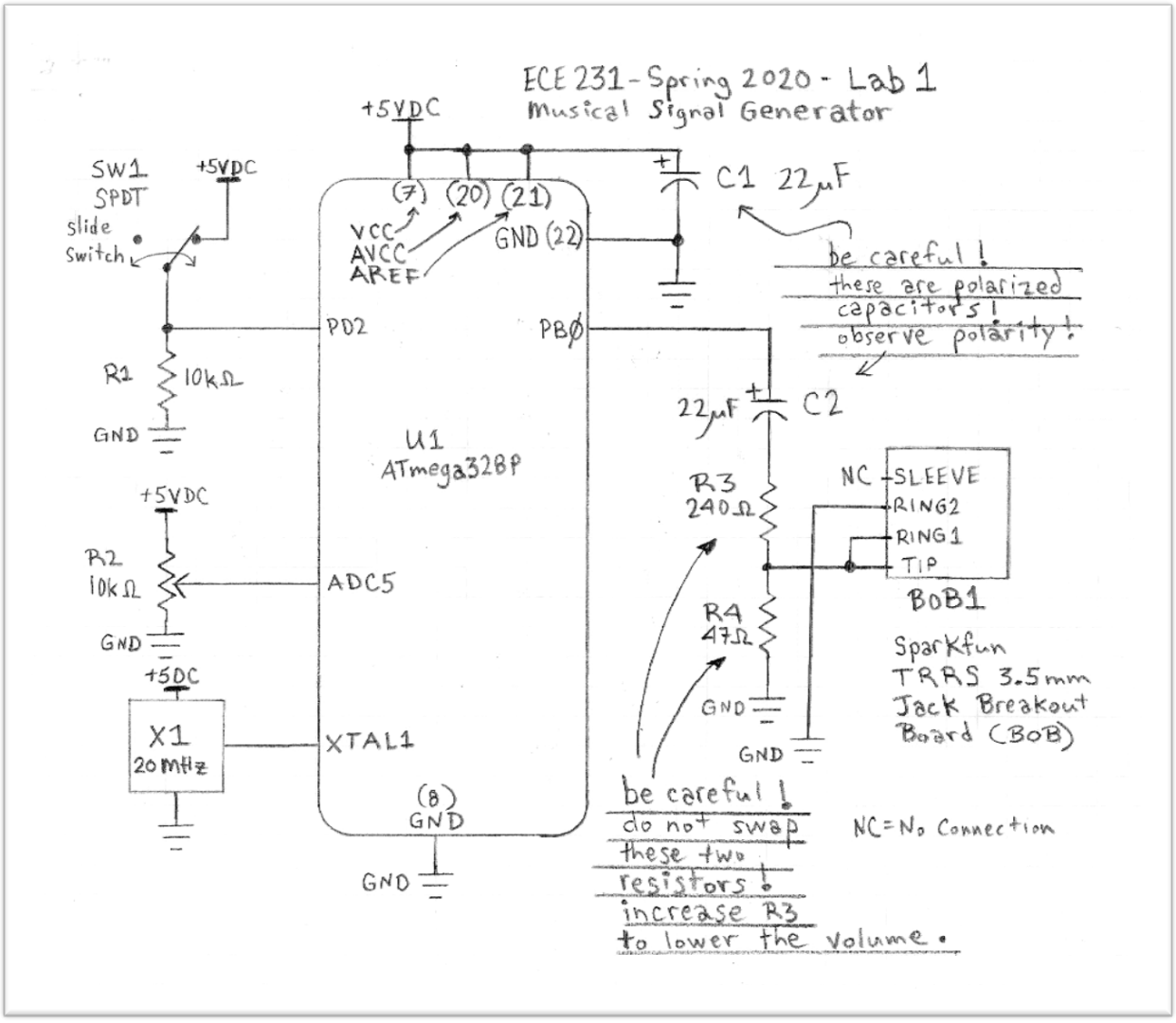
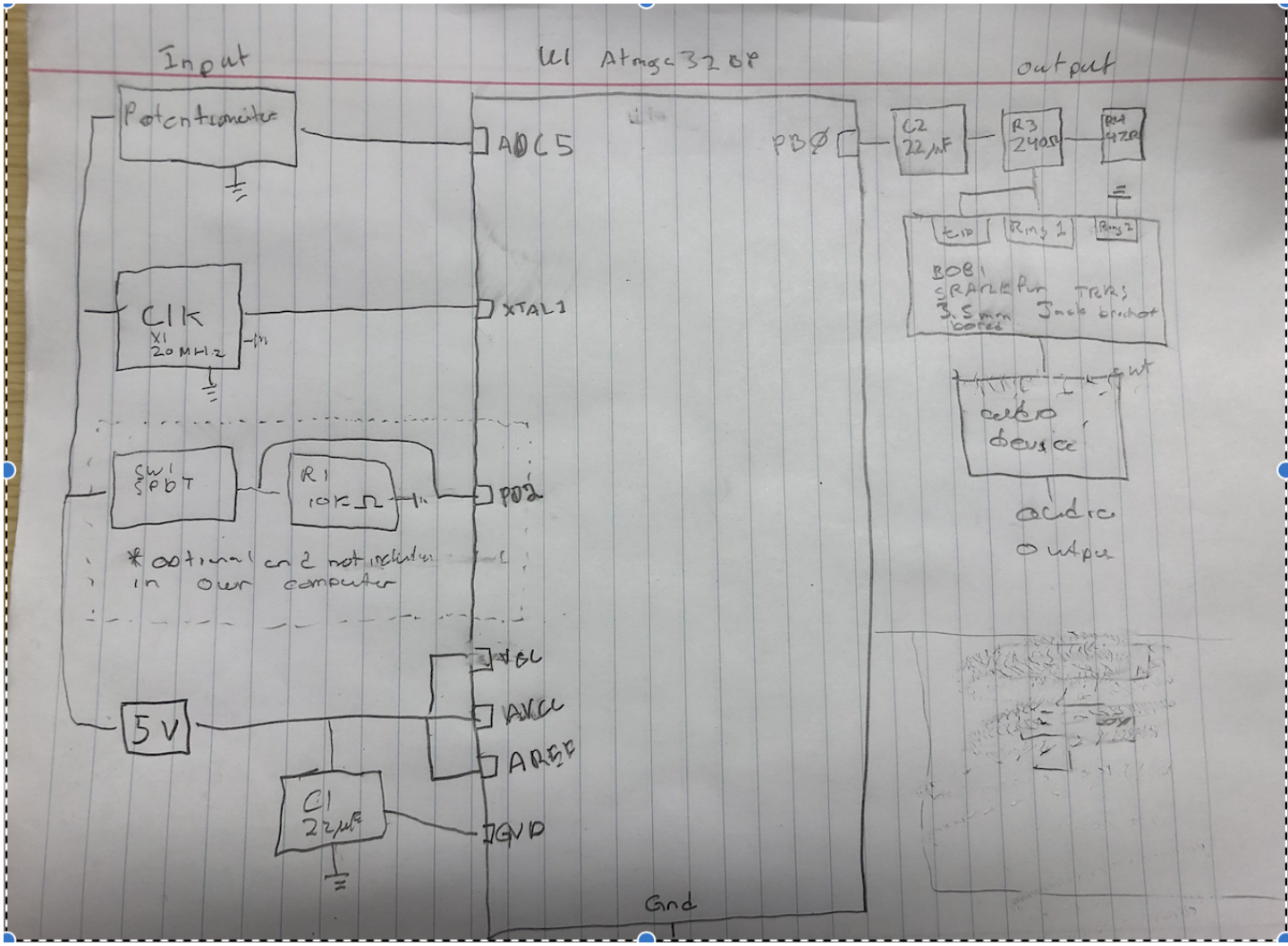


Fig 2



Part Reference	Part Description
BOB1	SparkFun TRRS 3.5mm Jack Breakout, outputs a sound wave to an auxiliary port
C1	22 microfarad electrolytic capacitor, avoid sudden changes in voltage from power to ground.
C2	22 microfarad electrolytic capacitor, regulate sudden changes in voltage
R1	10 kilo-ohm resistor, *optional and not included* allow input to PD2 to be 5v

R2 10 kilo-ohm potentiometer, change voltage based on nob position. Will control what note is being played

R3 240 ohm resistor, works with R4 to balance potential delivered to BOB1, the larger the ratio R4/R3 the higher the volume.

R4 47 ohm resistor, works with R3 to balance potential delivered to BOB1, the larger the ratio R4/R3 the higher the volume.

SW1 SPDT slide switch, *optional and not included* to toggle on and off PD2

U1 ATmega328P microcontroller, 8 bit computer chip that will run code. Outputs will vary based on inputs.

X1 20 MHz clock, external clock that cycles the computer through operations
\

Ports:

PD2 Port D bit 2, *optional and not included*, external interrupt source. Could enable and disable output

ADC5 Takes in a voltage, voltage be read at a value ranging from 0 to 2^10-1 (this is because the ADC is a 10 bit value) (voltage to be read between ground and AREF value)

XTAL1 Clock input

GND ground

PBϕ port B, dictates if the audio output is on or off

AREF AREF is the analog reference pin for the A/D Converter

AVCC the supply voltage pin for the A/D Converter

VCC Digital supply voltage

Software Description:

The C program written for this lab uses ports and delay loops to create sound waves. We were given the code for the first test, this used an infinite while loop to incase two delay loops that would force an output

port on and off. We then modified the code to operate at a certain frequency based off of the ADC reading.

The final program, found in appendix A, starts initializing variables outside of the loop, it also declares a list called half_period_map of 25 half periods that correspond to a musical note. The musical notes are organized from lowest pitch to highest pitch, this is important for the hashing function.

Next there is an infinite while loop, ADMUX and ADCSRA are set to allow the program to properly communicate with the hardware. Then the value of ADC is pulled, this will be a value between 0 and 2^10-1. The value of ADC is stored in an unsigned 16 bit int called “adc”. Variable adc is then divided by 41 and stored in char “index”. The purpose of this is to cast adc into a value ranging from 0 to 24 based off of ADC. Now that we have a value from 0 to 24 we use it to retrieve a half frequency from half_period_map and store this value in halfp. PORTB is then forced on, a delay loop lasts roughly as long as halfp in microseconds, PORTB is forced off, another delay loop lasts as long as halfp in microseconds. The infinite loop now repeats.

As you can see the force delay cycle inside the infinite loop causes the value of PORTB to export a square wave. (it goes up, delay, down, delay, repeat). This is sent to an aux jack that can export a sound wave.

Problems Encountered and Solutions:

Throughout the project, there was only one issue we ran into. This was encountered during test one, trying to make the written program work. The issue was that Atmel Studio could not recognize the microprocessor. This occurred after we had programmed the processor to use an external clock. Usually this error only occurs when the microprocessor is programmed this way and there is no external clock found. This is known as bricking the microprocessor. First we checked the circuit, and all wiring for the clock was found to be correct, so we checked the wiring for the rest of the circuit and that was found to be correct too. An oscilloscope was used to check the output of the clock to make sure it was 20Mhz, and this also was correct. We had replaced the microprocessor with a second similar one, and still had the same issues. Eventually, to solve the issue, we had replaced the clock with a new, same 20Mhz one, and the error was never replicated again.

Conclusion:

From this project, the student was able to create four working testing programs, one final script, as well as a fully functioning circuit that when put together, is able output different square waves at specific frequencies to represent twenty-five musical notes. First, for test one, the student had to build the circuit seen in Fig. 1, and create a program that could send a signal in order to flash an LED at approximately 5 Hz. Next, the student had to create test two, which using the code from test one, could create a Middle C square wave. The student was given this signal to be approximately 261.6 Hz, and in order to find this, an oscilloscope was used to measure the frequency of the output, while the half period for signal High and Low were changed until 261.6 Hz was found. Next, the student had to create test three, where the analog voltage from the 10k Potentiometer could be read, and stored. This was done by reading the ATmega328p datasheet, and finding the information to do so. Then, the student had created test four, where depending on the voltage output from the 10k potentiometer, a value for half period is selected. These half period

values were calculated by using the formula: $f_n = f_{ref} * 2^{(\frac{n}{2})}$, where f_n is the new frequency of a note, f_{ref} is the frequency of a referenced note, and n number of semitones the new note is away from the reference note. Lastly, the final program was created, combining test two and test four. This final program allowed the user to output one of any twenty-five musical notes.

In the future the lab could be enhanced by having a way to be able to play multiple notes at once. This could be done by using set resistors and buttons that when pressed, send specific voltages to the microprocessor, along with using more than one microprocessor to generate more than one output signal.

References:

[1] *Atmel Studio 7 User Guide*, Microchip Technology, Chandler, Arizona, 2018
[2] *Atmel-ICE User Guide*, Atmel, San Jose, California, 2016
[3] *megaAVR Data Sheet,* Microchip Technology, Chandler, Arizona, 2018

Appendix A.

```
/*
 * Project1GoodVersion.c
 *
 * Created: 2/10/2020 2:25:47 PM
 * Author : Ashton, Brad
 */

#include <avr/io.h>

// Main Code

int main(void)
{
	DDRB = 0b00000001;
	uint16_t index0;
	uint16_t index1;
	int halfp = 0;
	int half_period_map[25] = {3822, 3608, 3405, 3214, 3034, 2864, 2703, 2551, 2408,
2273, 2145, 2025,
		1911, 1804, 1703, 1607, 1517, 1432, 1351, 1276, 1204, 1136, 1073, 1012, 955};
	while(1)
	{
		ADMUX = 0b01000101; // [0:3] determine input (0101 => potentiometer)
		ADCSRA = 0b11000111; // bits [0;2] ar div factors, not sure what they do really
```

```
                uint16_t adc = ADC; // gives value of voltage range 0x000 to 0x03ff, it is that
range bc 3ff is 2^10-1
                char index = adc/41;
                halfp = half_period_map[index];

                PORTB = 0b00000001;

                index0 = halfp*7/6; // there are approx 7/6 loops per microsecond
                while(index0 > 0)
                {
                        index0--;
                }
                PORTB = 0b00000000;

                index1 = halfp*7/6;
                while (index1 > 0)
                {
                        index1--;
                }
        }
}
```