Homework 1

Steinarr Hrafn Höskuldsson

January 16, 2023

1 Eclipse

Getting Eclipse to work was a hassle.

Installed as per the instructions but a few things caused issues beyond what the instructions covered.

The new Arduino IDE version 2 seems to not support the old bootloader of an Arduino Nano, easy solution was to just downgrade to the old version 1.8 .

The AVR plugin on Eclipse seems to have a bug and when creating a project with the AVR Toolchain it doesn't get registered as an AVR project causing Eclipse to refuse to even try uploading, giving error along the lines of: 'AVR project not opened'

I finally found a hack at https://sourceforge.net/p/avr-eclipse/support-requests/38/ which said to paste: `<nature>de.innot.avreclipse.core.avrnature</nature>` into the `.project` file. That fixed that issue and I was on my merry way to every other issue covered in the instructions.

2 Blink Programs

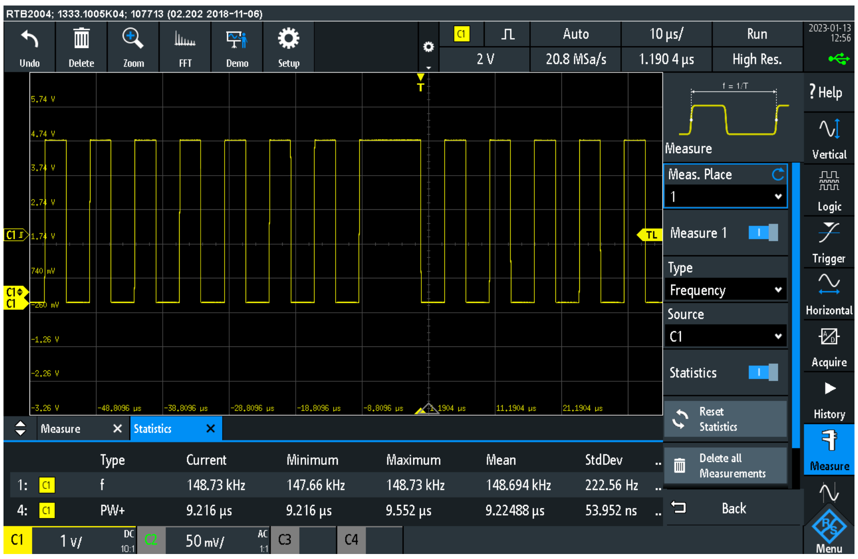
Two different blink programs were written, one for the Arduino IDE and another for the Eclipse bare bones.

The Arduino version is just the Arduino Blink example with the delays commented out.

The Eclipse bare bones version was taken from `Baldurs8bitGuide.pdf` .

3 Speed Comparisons

The Arduino Nano was connected to a RTB2004 Oscilloscope to measure the time delay between blinks. As can be seen in Figure 1 the Arduino version takes around $3\mu s$ to blink the LED, and not only that but once in a while it takes $9\mu s$. However as can be seen in figure 2 the bare bones eclipse version takes only $250ns$.


RTB2004; 1333.1005K04; 107713 (02.202 2018-11-06)

C1 | Auto | 10 µs/ | Run
2 V | 20.8 MSa/s | 1.190 4 µs | High Res.

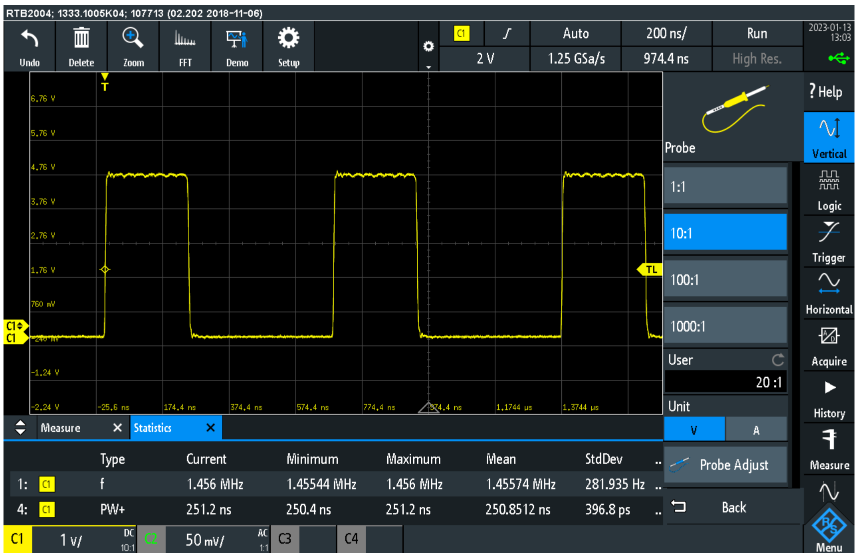
2023-01-13 12:56

Measure
Meas. Place: 1
Measure 1
Type: Frequency
Source: C1
Statistics

	Type	Current	Minimum	Maximum	Mean	StdDev
1: C1	f	148.73 kHz	147.66 kHz	148.73 kHz	148.694 kHz	222.56 Hz
4: C1	PW+	9.216 µs	9.216 µs	9.552 µs	9.22488 µs	53.952 ns

C1 1 V/ DC 10:1 | C2 50 mV/ AC 1:1 | C3 | C4

Figure 1: Arduino Blink test, runs at $150Khz$ and not consistently



		Type	Current	Minimum	Maximum	Mean	StdDev
1:	C1	f	1.456 MHz	1.45544 MHz	1.456 MHz	1.45574 MHz	281.935 Hz
4:	C1	PW+	251.2 ns	250.4 ns	251.2 ns	250.8512 ns	396.8 ps

Figure 2: Eclipse bare bones Blink test, runs consistently at $1.5Mhz$

4 Assembly Code

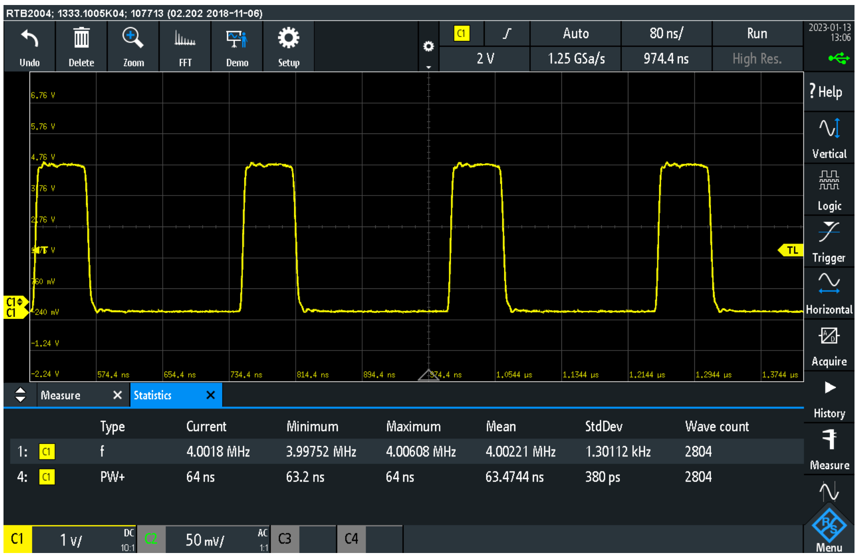
The relevant Assembly instructions for the while loop are:

```
     while (1)
{
PORTB = 1<<5 ;
  90: e5 e2        ldi r30, 0x25 ; 37
  92: f0 e0        ldi r31, 0x00 ; 0
  94: 80 e2        ldi r24, 0x20 ; 32
  96: 80 83        st Z, r24
PORTB = 0;
  98: e5 e2        ldi r30, 0x25 ; 37
  9a: f0 e0        ldi r31, 0x00 ; 0
  9c: 10 82        st Z, r1
  9e: f8 cf        rjmp .-16      ; 0x90 <main+0x10>
```

There are some hints that it might be able to run faster, flipping a register bit should be achievable in less than 4 instructions. And indeed it is. By changing the project build properties to heavily optimize the code we get the following assembly instructions.

```
     while (1)
{
PORTB = 1<<5 ;
  84: 85 b9        out 0x05, r24 ; 5
PORTB = 0;
  86: 15 b8        out 0x05, r1 ; 5
  88: fd cf        rjmp .-6       ; 0x84 <main+0x4>
```

Now each flip of the pin only takes one instruction. A screenshot of the oscilloscope with the optimized code running can be seen in Figure 3. Interestingly the loop is only 3 instructions,but runs at $4Mhz$ suggesting that the jump instruction takes 2 clock cycles to execute.



RTB2004; 1333.1005K04; 107713 (02.202 2018-11-06)

C1 2 V | Auto 1.25 GSa/s | 80 ns/ 974.4 ns | Run High Res. | 2023-01-13 13:06

		Type	Current	Minimum	Maximum	Mean	StdDev	Wave count
1:	C1	f	4.0018 MHz	3.99752 MHz	4.00608 MHz	4.00221 MHz	1.30112 kHz	2804
4:	C1	PW+	64 ns	63.2 ns	64 ns	63.4744 ns	380 ps	2804

C1 1 V/ DC 10:1 | C2 50 mV/ AC 1:1 | C3 | C4

Figure 3: Optimized Blink test, runs consistently at $4Mhz$

5 Schematic - Altium

Altium was installed and a schematic of the setup was created, it can be seen in Figure 4

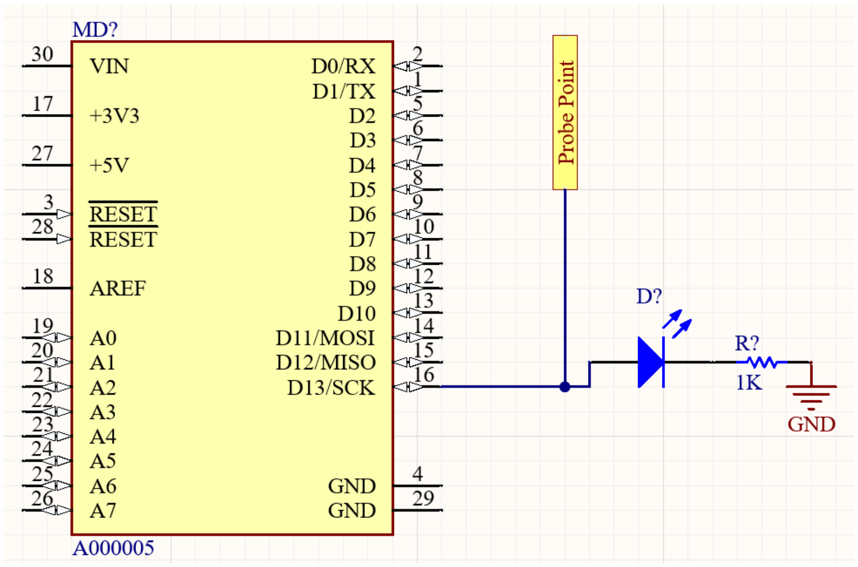


Figure 4: Schematic of the test setup used.

6 Final Project

For the final project I propose creating **AirDrums by Steinarr**, a pair of pants and possibly shoes that contain sensors capable of detecting hand slaps at certain spots as well as foot taps and output what is sensed as drum sounds.

6.1 Requirements

The requirements for the device, in order of importance, are

1. Detect foot taps on both feet
2. Detect hand slaps at certain locations, such as thighs.
3. Create drum sounds from what was sensed.
4. Allow customization of what sensor makes what sound.
5. Possibly transmit wirelessly to allow the drummer to dance around if they want.

6.2 Test plan

1. Test - Sense foot taps and hand slaps with high degree of accuracy ($> 99\%$).
2. Test - Create drum sounds with small enough latency such that it is not noticeable, aim for $5ms$ or less, $10ms$ would still be acceptable
3. Test - Test a setup sequence to allow user to customize what sensor makes what sound.
4. Test - Overall feeling when playing the AirDrums by Steinarr

6.3 Subtasks

1. Choose and test sensors for foot taps and hand slaps
2. Wire up and Program microcontroller to accurately detect foot taps and hand slaps
3. Choose and program a method to output sound.
4. Integrate the results of above steps into one program
5. Create a setup sequence to allow for customization of the system.
6. Choose and program a wireless transceiver to allow the drummer to dance around if they want.

6.4 Risk factors

1. Sensors not being precise enough, causing missed or erroneous detections.
2.

6.5 Estimated cost

Estimated Material costs for Arduino Nano, Soundboard, sensors (possibly accelerometers and contact microphones?), wires and other minor electronic components that might be needed: 20.000 ISK

Estimated hours that will be spent on the project: 60

Bringing the total estimated cost to: $60 * 5.000 + 20.000 = 320.000$ ISK

A Code

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage
    level)
  //delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage
    LOW
  //delay(1000);                       // wait for a second
}
```

Listing 1: Arduino IDE version of blink program

```
/*
 * main.c
 *
 *  Created on: Jan 12, 2023
 *      Author: Notandi
 */

#include <avr/io.h>
int main(){
   DDRB = 0x20;
   while (1)
   {
      PORTB = 1<<5 ;
      PORTB = 0;
   }
}
```

Listing 2: Eclipse bare bones version of blink program