

# **ECE-388**

## **Lab 3: Pattern Generator**

**Marcel Vieira**

**November 22, 2020**

I Marcel Vieira certify that this lab report consists only of work that is my own.

# Contents

<b>1</b>	<b>Abstract</b>	<b>2</b>
<b>2</b>	<b>Problem Statement</b>	<b>2</b>
<b>3</b>	<b>Procedures</b>	<b>2</b>
3.1	Materials . . . . .	2
3.2	Test Process . . . . .	2
<b>4</b>	<b>Discussion</b>	<b>9</b>
<b>5</b>	<b>Conclusion</b>	<b>10</b>
<b>6</b>	<b>Reflection</b>	<b>10</b>

# List of Figures

1	Pattern Gen Multisim V1 . . . . .	4
2	Pattern Gen Eagle Schematic V1 . . . . .	5
3	Pattern Gem BOM 1 . . . . .	5
4	Pattern Gen Code V1 PG 1 . . . . .	7
5	Pattern Gen Code V1 PG 2 . . . . .	8

# **1 Abstract**

This document describes the process taken in planning designing and constructing a pattern generator. The steps detailed currently include identification and defining the problem, creating a bill of materials, creating simulations, prototyping, and drafting PCB design layouts. This document also details the problems encountered along with how they were solved or proposed solutions. Due to the project still being worked on this document serves to document the progress.

## **2 Problem Statement**

Use surface mount components and a ATMEGA 328pb to generate binary patterns at either 5V or 3.3V, with a user interface that allows a user to change the pattern as well as the logic level across 8 channels.

## **3 Procedures**

### **3.1 Materials**

- Multisim
- Eagle
- Web Browser
- Atmel Studio

### **3.2 Test Process**

In order to begin the process of creating a pattern generator the problem must be properly defined to set the guidelines for what the solution will be. To define the problem the constraints must be laid out in an Engineering note book and clarified with the TAs and professor. They required that

the ATMEGA328PB chip be used, and all components that are chosen must be surface mount. These two constraints are not enough to clearly define the problem so more must be added such as an ability to switch logic levels and a user interface that allows people to interact with the board and adjust the the output pattern. Now the problem can be defined with a problem statement, in this case it is done with the one above.

Begin brain storming possible ways to resolve the problem. The ATMEGA328PB is capable of outputting digital signals in patterns so one of the biggest issues is changing the 5V is outputs from 5V to 3.3V when desired. This could be done with a dc converter, using resistors or diodes to drop the voltage, or using MOSFETS to toggle the flow of electricity form whatever logic source is selected. The second issue is in getting the 3.3V form the 5V that will come into the board. This could be done using a voltage divider, a DC to DC converter, or a Zener diode. The third issue was how was the user interface going to be made. The interface could be made either using hard ware switches, buttons, and rotary encoders or use serial port to type in the pattern. The last problem was how was information going to be shown to the user? The information could be shown using as series of LEDs or on a screen.

The next step is to decide which brainstormed solutions will be used to solve the problem. For this project the user interface will use a serial input with a screen for the display. For the power logic level selection a Zener diode will be used to get the 3.3V with a sp3t switch to switch between the power states which will include 5V 3.3V and 0V. The voltage that comes out of the switch then will be passed to MOSFETS that will be toggled by the ATMEGA328PB chip.

Now that the plan has been completed the parts must be selected and tested in Multisim (figure 1). For this case the displays cannot be properly tested so they will be tested physically later on. The MOSFETS when input in Multisim showed that the were capable of using 5V at the gate to toggle both 5V and 3.3V. The Zener diode also showed it was a valid solution to getting 3.3V.

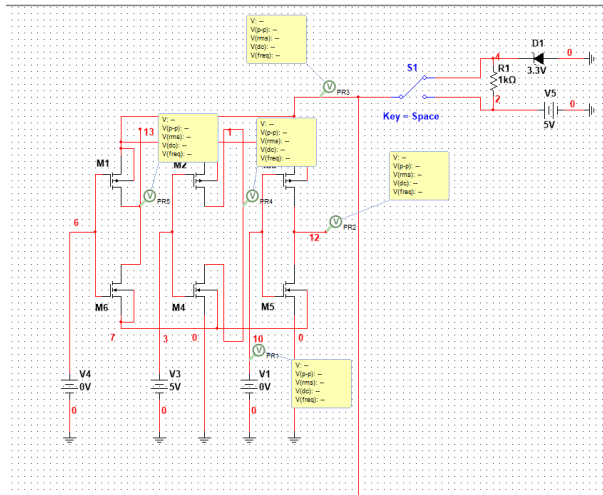


Figure 1: Pattern Gen Multisim V1

Then the parts that are required to build the circuit are searched in LCSC and Digikey. When a part was found it was inserted into Eagle (figure 2). Once all the parts were placed in Eagle a bill of materials is exported and opened in excel in order to add purchase information and remove irrelevant information (figure 3).



After the parts were selected and the circuit was successfully simulated it was time to begin construction of the prototype. The prototype allows for the circuit to be tested prior to having everything on the PCB. The use of a prototype especially one on a bread board allows for parts to be more easily isolated and replaced. The prototype is constructed based on the circuit design used in Multisim but the prototype is broken into sections to test the functionality of all the individual smaller parts prior to testing the circuit in its entirety. The prototype consisted of a voltage regulator, a switching circuits made with 2 PMOSs and an inverter, and a set of CMOS inverters.

The first test conducted was connect a voltmeter to the the voltage regulator and check if the proper voltages are produced. The expected values were 5V at the input and 3.3V at the output. This ensures that the voltage does not damage the circuit and sets a base line for the voltage to expect in the inverters.

The second test is to put 5V and 3.3V to the switching circuit use the voltmeter to check if it is outputting the correct input depending on what signal is sent to it. If a high is put into the circuit it should output 3.3V, if a low is put into the circuit it should output 5V.

The final test for the first prototype was to check if the CMOS inverters could produce the expected signal. To test this a clock pulse is put into the inverter and the response as well as the input are measured and compared with an oscilloscope. The result showed be the opposite of the input with the exception of the high being what voltage is put to the sources of the PMOSs in the CMOS inverters.

After taking notes on the first prototype the second was built. This prototype is focused on the user interface. This prototype consists of a rotary encoder with a built in button, several LEDs, and a 328pb. For this prototype code was written in Atmel Studio. The code for the prototype used various interrupts with 2 pin change interrupts being used to determine the direction the rotary encoder is turned on. There is also another pin change interrupts that is used to change the button press. This user interface was tested using debugging and visual observation to tell whether it is operational.

The user interface should have LEDs that start off displaying the current binary signal, this

```

1  /*
2   * Pattern Generator.c
3   *
4   * Created: 10/30/2020 12:13:11 PM
5   * Author : pphantom
6   */
7
8  #define F_CPU 16000000
9  #include <avr/io.h>
10 #include <stdio.h>
11 #include <avr/interrupt.h>
12 #include <util/atomic.h>
13 #include <util/delay.h>
14
15 int rotorCheck();
16
17 static uint8_t temp = 0;
18 static uint8_t i = 0;
19 static uint8_t turned = 0;
20 static uint8_t clockwise = 0;
21 static uint8_t mode = 0;
22 static uint8_t buttonPress = 0;
23 static uint8_t bit = 0;
24 static uint8_t channelNum = 0;
25 static uint8_t voltage = 0;
26 static uint8_t channel[8] = {0x65,0x81,0xff,0xf6,0x82,0x80,0x34,0x75};
27
28 int main(void)
29 {
30     cli();
31
32     DDRA = 0b11111110;
33     PORTA = 0b11111111;
34
35     DDRC = 0b00000000;
36     PORTC = 0b11111111;
37
38     DORD = 0b01111111;
39     PORTD = 0b01111111;
40
41     DDRE = 0b11111111;
42     PORTE = 0b11111110;
43
44     PCICR = 0b00000111;
45     PCMSK0 = 0b00000001;
46     PCMSK1 = 0b00000001;
47     PCMSK2 = 0b10000000;
48
49     OCR1A = 0x3C08;
50     TCCR1B = (1 << WGM12);
51     TIMSK1 = (1 << OCIE1A);
52     TCCR1B = (1 << CS12) | (1 << CS10);
53
54     sei();
55
56     while (1)
57     {
58         switch (mode)
59         {
60             case 0:
61                 channel[channelNum] = (channel[channelNum] + rotorCheck()) & 0b00111111;
62                 PORTD--(channel[channelNum]);
63                 break;
64
65             case 1:
66                 channelNum = (channelNum + rotorCheck()) % 8;
67                 PORTD--(channelNum);
68                 break;
69
70             case 2:
71                 voltage = (voltage + rotorCheck()) % 2;
72                 if (voltage == 0)
73                     break;
74         }
75     }
76 }

```

Figure 4: Pattern Gen Code V1 PG 1



```

76         }
77         else
78         {
79             PORTD = 0b11011111;
80         }
81         break;
82     }
83 }
84 }
85
86
87
88 int motorCheck()
89 {
90     if(turned)
91     {
92         turned = 0;
93         if(clockwise)
94         {
95             return 1;
96         }
97         else
98         {
99             return -1;
100         }
101     }
102     return 0;
103 }
104
105
106
107
108 ISR(PCINT0_vect)
109 {
110     if(!turned)
111     {
112         clockwise = 1;
113     }
114     turned = 1;
115     _delay_ms(150);
116 }
117
118
119
120 ISR(PCINT1_vect)
121 {
122     if(!turned)
123     {
124         clockwise = 0;
125     }
126     turned = 1;
127     _delay_ms(150);
128 }
129
130
131
132 ISR(PCINT2_vect)
133 {
134     if (!buttonPress)
135     {
136         mode = (mode + 1) % 3;
137     }
138     buttonPress = ~buttonPress;
139 }
140
141
142
143 ISR(TIMER1_COMPA_vect)
144 {
145     cli();
146     i=0;
147     temp = 0;
148 }

```

Figure 5: Pattern Gen Code V1 PG 2

is referred to as mode 1. In mode 1 the rotary encoder changes the binary value going up when turned clockwise and down when turned counter clockwise. When the button is pressed it enters mode 2 which allows the user to view the the current channel number in binary displayed on the LEDs. Mode 2 also allows for the user to switch between channels using the rotary switch. When the button is pressed again it goes into mode 3 which simulates the display of the voltage settings which is adjustable with the rotary encoder as well.

Once the final design has been decided on the the design must properly be translated into a schematic that will be uploaded to Github and checked by Tim Chase. The schematic must contain all parts that are to be used, in this instance that would be the micro USB, the CMOS chips, the serial chip, the LEDs, the OLED display, and the rotary encoder. Power must flow in the schematic from the left to right and be clearly labeled. The schematic must finally pass all ERC checks with no errors or only errors approved by Tim Chase.

Once the schematic is approved by Tim Chace the board lay out can then be worked on. The board layout must include all parts, ground planes, fiducials to show the alignment of the board, and must pass all DRC checks in the file Dirty8TS.dru. On the back of the board the board must have a name, date, version number, and the Git repository link printed on the back of the board.

## **4 Discussion**

Originally the circuit was going to use a voltage divider to drop the voltage to 3.3V, and N-type MOSFETS to toggle to logic voltages on and off but they presented several issues in the Multisim simulations. For the voltage divider applying the 3.3V to a load changed the resistance on altering the voltage and because the resistance was going to be difficult to predict the Zener diode was used. For the N-type MOSFETS they worked perfectly but each of the MOSFETS was there own chip, but after a suggestion by Tim Chace CMOS inverters were used instead which proved to be just as affective and required 3 times less chips saving space. These test processes lead to the revision of the bill of materials 3 times.

For prototyping all test ran as expected proving that all systems individually are operational. This is with the exception of the linear regulator, the MCP2200 serial chip, and the OLED display. For the display and serial chip they have not come in time to be properly tested, but the voltage regulator was tested and failed. The failure at first was due to the wrong chip being supplied without a part number which meant that the wrong data sheet was used to wire the chip. This in turn damaged the chip and because there was only one further tests could not be done. An attempt was made to test a complete prototyping but there were complications with the 328pb which made it unable to be identified by any computer. This resulted in the 328pb not being able to be programmed. This caused the loss of another test session. Fortunately Ben Vial was able to repair the board so even if the issue occurs again there is a know solution for the problem now.

## **5 Conclusion**

For the bill of materials it is completed but it still may change because no solution has been found to electronically switch between the sources being used for the logic levels. There are still issues with the voltage regulator and a full prototype replicating the entire system must still be tested.

## **6 Reflection**

The planning and simulation had no major issues. There only three minor issues. The first was that it was unclear early on what the pattern generator was and what it was supposed to do. The second was that simulating the circuits proved difficult at first because the necessary chips were not in Multisim so they had to be built out of equivalents that had to be found in Multisim. The third issue was that no clear way to use the MOSFETS to switch between logic sources was found.

In the prototyping stage one of the major issues was a lack of parts to work with which caused a lot of time to be lost and a lot of confusion. This problem was difficult to avoid due to the current pandemic and the restrictions placed as a consequence. Another issue encountered with prototyping was time management. Due to issues with the regulator a majority of lab time was

spent around trouble shooting it. Because so much time was lost the rest of the test were done quickly and pictures were not taken. This is why there are no images of the first and second prototyping stages.