Design and Implementation of High Bandwidth Power Amplifier for Characterization of Magnetic Materials

Group Members -
Omkar Nitsure - 210070057
Ojas Karanjkar - 210070040
Kushal Gajbe - 210070048
Vaibhav Garg - 21d070085

Contents

1 Introduction

Magnetic Materials are used in many application and in many different equipment for their special properties. But before using them, we need to have in-depth knowledge of their characteristics. This requires performing BH characterization. In BH characterization, we need to apply a sinusoidal voltage of varying amplitude and frequency across the magnetic material and check for the current it draws. Then the current and voltage information can be used to compute the value of **B** and **H**. The area enclosed by this curve is an indication of the amount of power dissipation in the material. This and other information is vital for choosing the right material for a specific application.

So, now we understand why BH characterization is important. We need a source which is able to generate a sinusoidal voltage with varying amplitude and frequency and source the current drawn by the magnetic material. The aim of our project is to design such a source. This problem can be tackled either using a **digital + analog** or **only analog** approach. We tried both approaches. We designed an analog only circuit comprising of **Class-AB** amplifier and for the digital + analog we have used a Class-D amplifier based circuit. Let's discuss both the approaches in the following sections -

2 Target Specifications

We were required to build a source satisfying the above constraints -

- **Output Frequency** = 2-3 kHz
- **Output Sine amplitude** = upto 30 V_{pp}
- **Current Sourcing capacity** = 1-1.5 A
- Added functionality of allowing user to change frequency on the go using a knob.
- Display the current frequency on a LCD screen for ready reference

3 Class AB based Circuit

In this approach, we designed a circuit which can be broken down in a **Voltage amplification circuit** and a **Current sourcing circuit** and connected them in cascade. The circuit diagram is as follows -

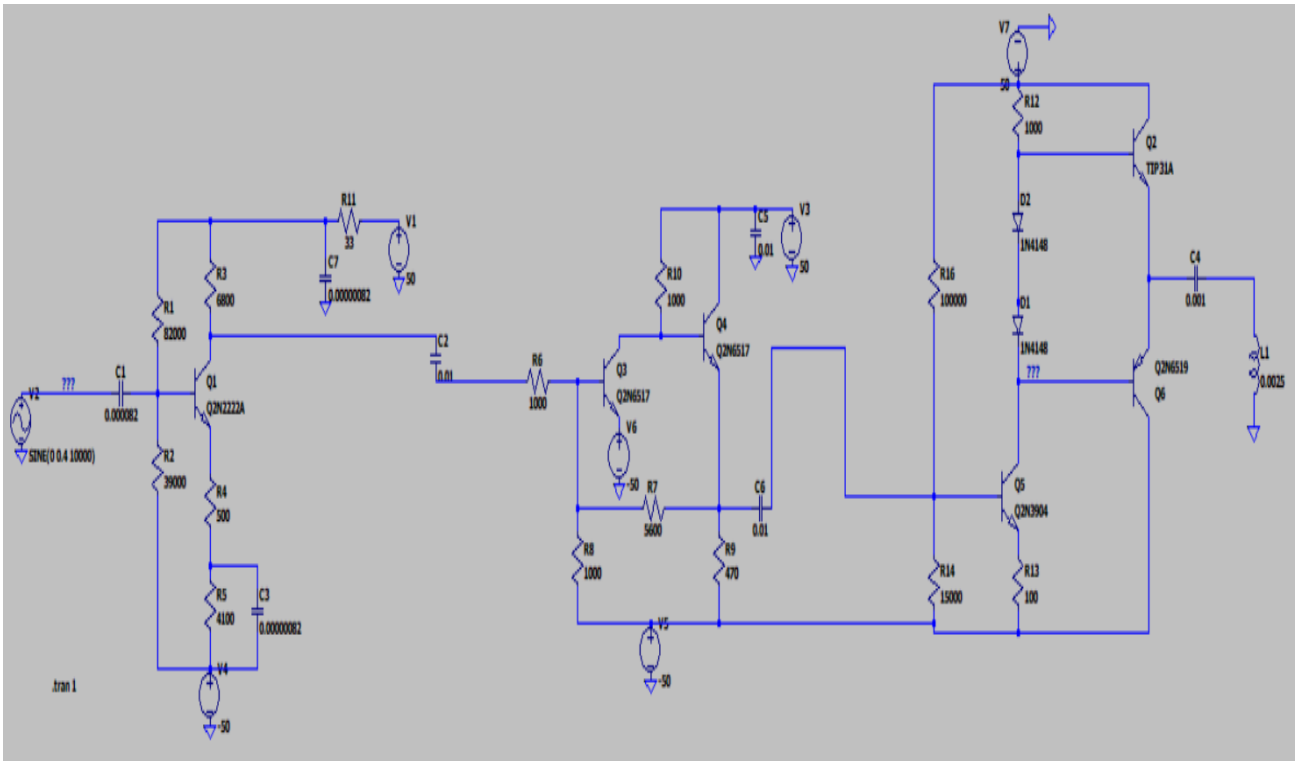


Figure 1: Circuit diagram for Class AB amplifier based circuit

The above circuit was giving the correct required output but there was an issue. There was a lot of power dissipation in the different BJTs which would have required big heat sinks and also a lot of power would have got wasted along with the unavoidable power loss due to eddy currents. We tried changing a lot of different parameters but we were not able to reduce the power-loss to the desired levels. While we were searching for the potential solution, we came across **class-D amplifier** which was not having the issue of power loss. After a lot of brainstorming, we decided to scrap this approach and continue with the approach involving Class-D amplifier.

4 Class D Amplifier

Class D amplifier consists of a full-bridge MOSFET based circuit. Each of the 4 MOSFETs are fired by 4 different PWM signals across their gate-source. Two MOSFETs in the same leg get complementary PWM signal such that there ON times are separated by a non-zero dead time. This dead-time is provided to ensure that there is sufficient time for the MOSFETs to change their states ensuring that both MOSFETs don't turn-on together and short V_{dd} with GND. The duty ratio of the PWM signals vary sinusoidally which enables the output to be sinusoidal.

Duty ratios of the 4 PWM signals are as follows -

$$d_1 = \frac{1}{2} + msin(\omega t)$$

$$d_2 = \frac{1}{2} - msin(\omega t)$$

Then d_3 and d_4 are just complements of d_1, d_2 respectively seperated by dead time.

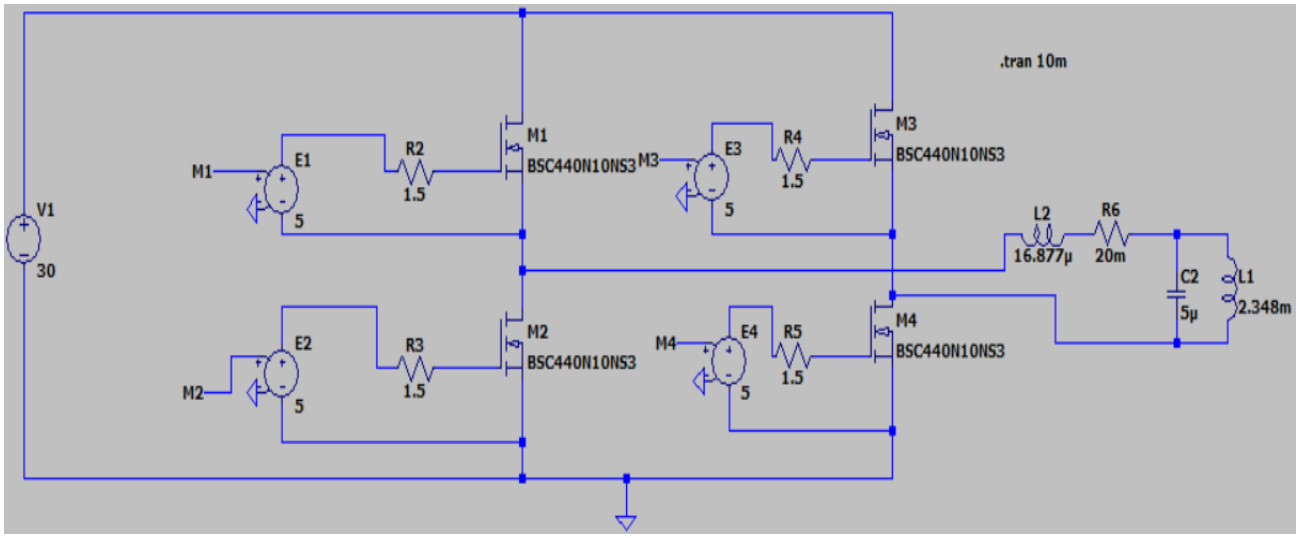


Figure 2: full-bridge MOSFET based circuit

4.1 Simulations

We performed the LTSpice simulations of class D amplifier for our required specifications and got desired results. We generated the sinusoidally varying PWM signals by giving a triangular wave and a sine wave as inputs to a comparator. We kept **zero** dead time assuming perfect switching of MOSFETs. The MOSFETs were switching at a frequency of **100kHz** and we tested the sinusoidal outputs for frequencies till **3kHz** which was our target specification. We also checked for the **power dissipation** in the entire circuit(in all components) and found that they were in **mW**. Thus this approach allowed us to generate sine waves of large amplitudes for frequencies upto **3kHz** without requiring any heat-sink at all!! This approach has some cons but that is a trade-off for excellent efficiency. The main limitation of this approach is that it generates **saw-tooth noise** because of MOSFET switching whereas a purely analog approach will not add any such noise. The results of our simulations are given below where the output voltage across the inductive load is given at the edge of our required specifications. We also give the corresponding current drawn by the inductive load which is well within our specifications.

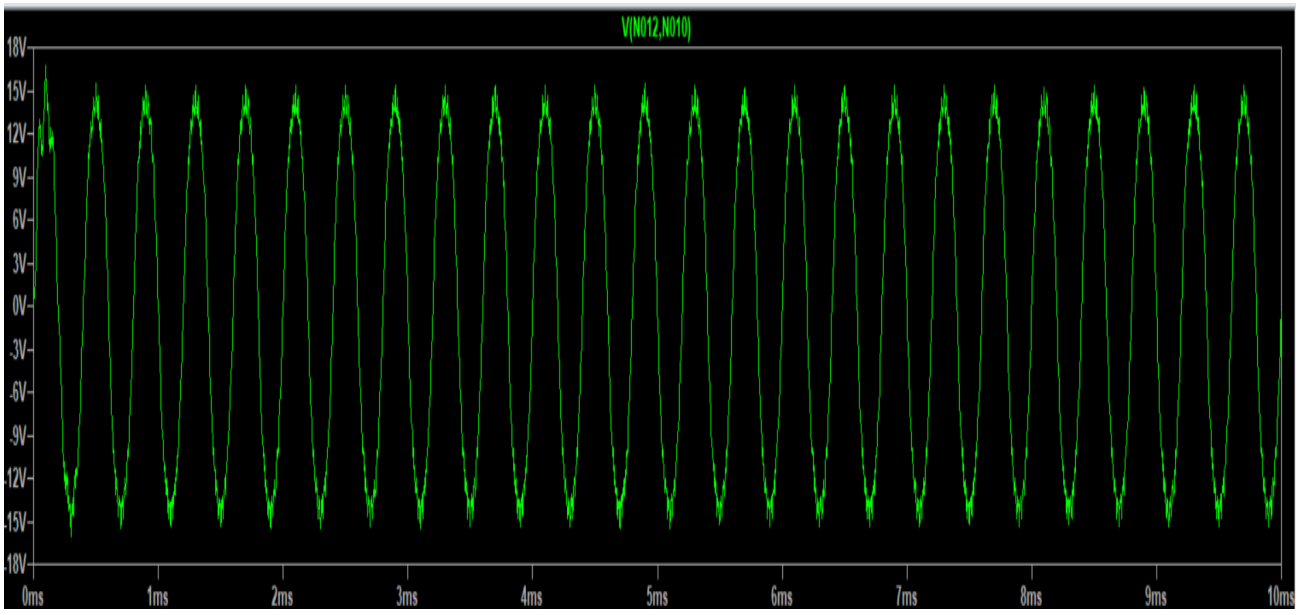


Figure 3: output voltage across load

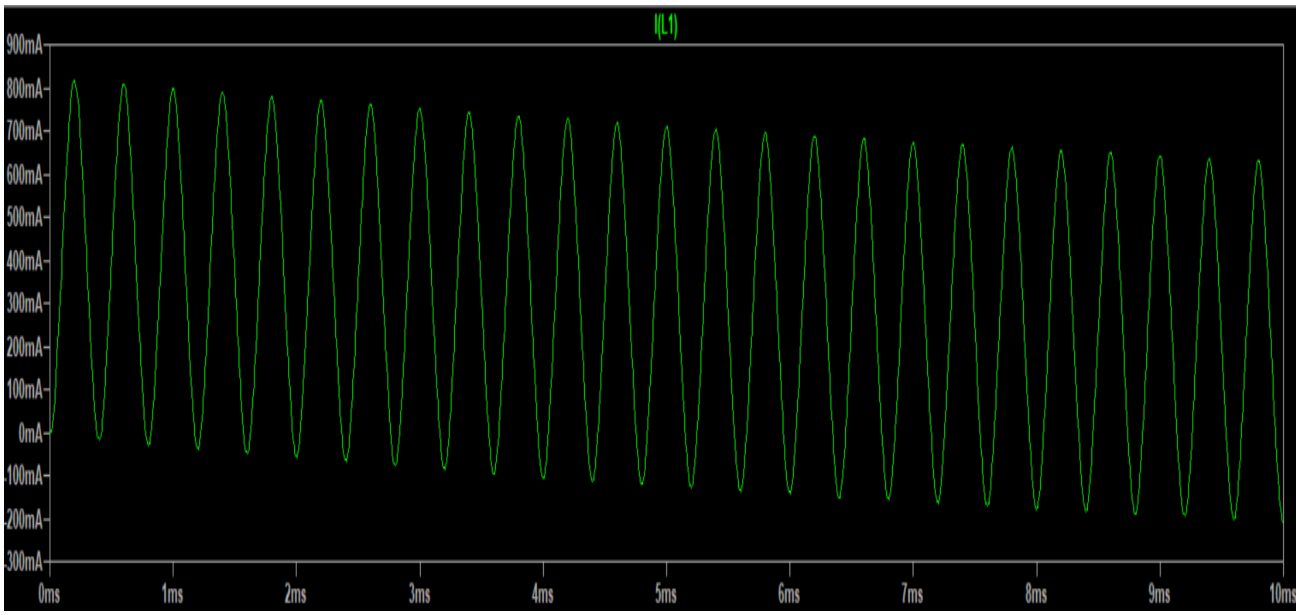


Figure 4: Current through the inductive load

4.2 Choice of BOM and justifications

The Bill of Materials required for our circuit is as follows -

Component Count:	44						
Ref	Qnty	Value	Cmp name	Footprint	Descriptio	Vendor	DNP
C1	1	470u	C_Polarized	Capacitor_THT:CP_Radial_D10.0mm_P5.00mm	Polarized capacitor		
C2, C3, C6, C7, C10, C11	6	20p	C	Capacitor_SMD:C_0805_2012Metric_Pad1.18x1.45mm_HandSolder	Unpolarized capacitor		
C5, C12, C13, C14, C15	5	470n	C	Capacitor_SMD:C_1206_3216Metric_Pad1.33x1.80mm_HandSolder	Unpolarized capacitor		
C8, C9	2	470p	C	Capacitor_SMD:C_0805_2012Metric_Pad1.18x1.45mm_HandSolder	Unpolarized capacitor		
CN7, CN10	2	Conn_02x19_Odd_Even	Conn_02x19_Odd_Even	Connector_PinSocket_2.54mm:PinSocket_2x19_P2.54mm_Vertical	Generic connector, double row, 02x19, odd/even pin numbering scheme (row		
D1, D2	2	1N5822	1N5822	Diode_THT:D_DO-201AD_P15.24mm_Horizontal	40V 3A Schottky Barrier Rectifier Diode, DO-201AD		
H1, H2, H3, H4	4	MountingHole	MountingHole	MountingHole:MountingHole_3mm	Mounting Hole without connection		
J1, J3	2	Conn_01x02	Conn_01x02	Connector_PinSocket_2.54mm:PinSocket_1x02_P2.54mm_Vertical	Generic connector, single row, 01x02, script generated (kicad-library-utils/sch		
J2, J4	2	Screw_Terminal_01x02	Screw_Terminal_01x02	TerminalBlock:TerminalBlock_Altech_AK300-2_P5.00mm	Generic screw terminal, single row, 01x02, script generated (kicad-library-utils		
LCD1	1	Conn_01x16	Conn_01x16	Connector_PinSocket_2.54mm:PinSocket_1x16_P2.54mm_Vertical	Generic connector, single row, 01x16, script generated (kicad-library-utils/sch		
Q1, Q2, Q3, Q4	4	IRF840	IRF840	IRF840-final:TO-220-1			
R1, R3	2	47	R	Resistor_SMD:R_0805_2012Metric_Pad1.20x1.40mm_HandSolder	Resistor		
R2, R4	2	330	R	Resistor_SMD:R_0805_2012Metric_Pad1.20x1.40mm_HandSolder	Resistor		
R5, R6, R7, R8, R9, R10	6	10	R	Resistor_THT:R_Axial_DIN0309_L9.0mm_D3.2mm_P12.70mm_Horizontal	Resistor		
RV1	1	1k	R_Potentiometer	Potentiometer_THT:Potentiometer_Bourns_3386X_Horizontal	Potentiometer		
U1, U2	2	2ED020I12-FI	2ED020I12-FI	Gate-Driver-final:PG-DSO-18-2_INF			

Figure 5: Bill of Materials

Gate Drivers - The PWM output cannot be directly given to the gates of MOSFETS as the back-currents generated while a MOSFET is turning-OFF will certainly damage the GPIO pins on the Micro-controller. We wanted to have a gate driver which can amplify the rail-to-rail logic of **3.3 V** coming from the micro-controller to rail-to-rail logic of **15V**. We also wanted the current sinking capacity to be high which will allow us to switch at much higher frequency. Thus we chose the **2ED02012-FI** gate driver from Infenion

Bluepill - STM32 minimum package called Bluepill was more than sufficient for our application. It was able to generate the **4** independent PWM signals. It also had ADC which was used to provide the functionality of a knob for changing the output frequency to the user. We also needed 4 additional **analog pins** to write current frequency value on the LCD display. All of these requirements were satisfied by Bluepill.

RC filter at the input of the gate driver - This low-pass filter was used to remove the high-frequency peaks in the PWM outputs of the microcontroller.

MOSFET - We wanted to produce output sinusoidal signals of 30 V_{pp} and thus the V_{ds} specifications of the MOSFETs must be larger than this value. We also wanted to source currents upto

even **2A** and thus the I_{ds} rating must be larger than this value. We also wanted the MOSFET to be a power MOSFET having the capability of very fast switching. MOSFET named **IRF840** satidfied all these constraints and thus we chose that MOSFET for the circuit.

LC circuit at the output - We had to use a low-pass filter at the output of a **Class-D** amplifier to remove the saw-tooth noise added due to switching. We could have used a **RC** filter but that would have resulted in large power-loss in the resistor, so we chose to use **LC filter** of appropriate value which would not cause any power-loss.

5 Microcontroller Programming

We generate the PWM signal with Sinusoidally varying dead time using the **STM32 Bluepill** microcontroller. The microcontroller clock frequency is **8MHz**. The internal circuitry increases it such that the **timers**(which we used) get a frequency of **64MHz** while the ADC sampling frequency is **8MHz**. We decided to keep the switching frequency to be **100kHz** which is almost **33** times larger than the maximum rated frequency of our system. It is also well within the maximum attainable switching frequency by the Power MOSFET. We have kept the cut-off frequency to be **12kHz** which is less than **10** times the maximum rating. Let us now see how did we generate the PWM signal with sinusoidally varying duty ratio.

5.1 PWM generation

We used **2** channels from **TIM1** on the STM32. Each of these channels have a register named **CCR** which can be used to manipulate the duty ratio of the microcontroller. So, we stored the values for 1 cycle of the sinusoidal signals corresponding to **d1** and **d2** in the memory and read those values after a frequency dependent delay. This allowed us to generate sinusoidally varying PWM. Dead-time was also provided to ensure proper switching without any shorting which might damage the MOSFETs. We used a constant dead time of **0.125 us**

5.1.1 Lookup Table for Sine

```
const uint32_t sine_pwm1[NS] = {
  460, 469, 478, 486, 495, 503, 512, 520, 527, 535, 542, 549, 556, 562, 568, 573,
    578, 583, 587, 590, 593, 596, 598, 599, 600, 600, 600, 599, 598, 596, 593,
   590, 587, 583, 578, 573, 568, 562, 556, 549, 542, 535, 527, 520, 512, 503,
   495, 486, 478, 469, 460, 451, 442, 434, 425, 417, 408, 400, 393, 385, 378,
   371, 364, 358, 352, 347, 342, 337, 333, 330, 327, 324, 322, 321, 320, 320,
   320, 321, 322, 324, 327, 330, 333, 337, 342, 347, 352, 358, 364, 371, 378,
   385, 393, 400, 408, 417, 425, 434, 442, 451
};
```

```
4
5 const uint32_t sine_pwm2[NS] = {
6   180, 171, 162, 154, 145, 137, 128, 120, 113, 105, 98, 91, 84, 78, 72, 67, 62,
    57, 53, 50, 47, 44, 42, 41, 40, 40, 40, 41, 42, 44, 47, 50, 53, 57, 62, 67,
    72, 78, 84, 91, 98, 105, 113, 120, 128, 137, 145,154, 162, 171, 180, 189,
    198, 206, 215, 223, 232, 240, 247, 255, 262, 269, 276, 282, 288, 293, 298,
    303, 307, 310, 313, 316, 318, 319, 320, 320, 320, 319, 318, 316, 313, 310,
    307, 303, 298, 293, 288, 282, 276, 269, 262, 255, 247, 240, 232, 223, 215,
    206, 198, 189
7 };
```

5.1.2 Updating Duty Cycle of PWM Signal

```
1 while(idx < NS)
2     {
3       TIM1->CCR1 = sine_pwm1[idx];
4       TIM1->CCR2 = sine_pwm2[idx];
5       DELAY_US(delay_cont);
6       idx++;
7     }
8
9     idx = 0;
```

5.1.3 Delay Function

```
1 #define SYSTICK_LOAD (SystemCoreClock/1000000U)
2 #define SYSTICK_DELAY_CALIB (SYSTICK_LOAD >> 1)
3
4 #define DELAY_US(us) \
5     do { \
6           uint32_t start = SysTick->VAL; \
7           uint32_t ticks = (us * SYSTICK_LOAD)-SYSTICK_DELAY_CALIB;  \
8           while((start - SysTick->VAL) < ticks); \
9     } while (0)
```

5.2 ADC interfacing

We wanted to give a simple knob-based control to the user which he can use to set the output frequency he wants. This knob was made using a potentiometer circuit where the ends of the potentiometer are connected to the **GND** and **3.3V** of the microcontroller. The middle pin is connected to the **ADC** and the values of voltage read using the ADC are then mapped to appropriate frequencies. The ADC is sampled every **1000** cycles to ensure that we don't sample "too often". The code snippet used is as follows -

```
iters += 1;
    if(iters % 1000 == 0)
    {
      HAL_ADC_PollForConversion(&hadc1, 100);
      pot_value = HAL_ADC_GetValue(&hadc1);
      freq = (int)((pot_value*3000)/4095);
      delay_cont = (int)((10000 - freq)/freq);
```

5.3 LCD interfacing

We have also interfaced the LCD with the system so that user can see the current frequency output from the system in real time and adjust the knob accordingly to set the frequency of his choice. We implement the interfacing with **6** pins, **4** of which are the actual data pins. We write the frequency value after every **1000** cycles right after sampling the ADC. Thus we ensure that we don't write to the LCD "too often". The CAD model was then made accordingly to ensure that the LCD is clearly visible to the user.

```
  Lcd_PortType ports[] = {GPIOA, GPIOA, GPIOA, GPIOA};
  Lcd_PinType pins[] = {GPIO_PIN_2, GPIO_PIN_3, GPIO_PIN_4, GPIO_PIN_5};
  Lcd_HandleTypeDef lcd;
  lcd = Lcd_create(ports, pins, GPIOA, GPIO_PIN_1, GPIOA, GPIO_PIN_0,
   LCD_4_BIT_MODE);
  Lcd_cursor(&lcd, 0,3);
  Lcd_string(&lcd, "Frequency");


        iters += 1;
        if(iters % 1000 == 0)
        {
            HAL_ADC_PollForConversion(&hadc1, 100);
            pot_value = HAL_ADC_GetValue(&hadc1);
            freq = (int)((pot_value*3000)/4095);
            delay_cont = (int)((10000 - freq)/freq);

            Lcd_clear(&lcd);
            Lcd_cursor(&lcd, 0,3);
            Lcd_string(&lcd, "Frequency");
```

```
            Lcd_cursor(&lcd, 1, 7);
            Lcd_int(&lcd, freq);
            iters = 0;
        }
```

5.4 LCD functions code:

```
 #include "lcd.h"
const uint8_t ROW_16[] = {0x00, 0x40, 0x10, 0x50};
const uint8_t ROW_20[] = {0x00, 0x40, 0x14, 0x54};
/*********************************** Static declarations
    ************************************/

static void lcd_write_data(Lcd_HandleTypeDef * lcd, uint8_t data);
static void lcd_write_command(Lcd_HandleTypeDef * lcd, uint8_t command);
static void lcd_write(Lcd_HandleTypeDef * lcd, uint8_t data, uint8_t len);


/*********************************** Function definitions
    ************************************/

/**
 * Create new Lcd_HandleTypeDef and initialize the Lcd
 */
Lcd_HandleTypeDef Lcd_create(
    Lcd_PortType port[], Lcd_PinType pin[],
    Lcd_PortType rs_port, Lcd_PinType rs_pin,
    Lcd_PortType en_port, Lcd_PinType en_pin, Lcd_ModeTypeDef mode)
{
  Lcd_HandleTypeDef lcd;

  lcd.mode = mode;

  lcd.en_pin = en_pin;
  lcd.en_port = en_port;

  lcd.rs_pin = rs_pin;
  lcd.rs_port = rs_port;

  lcd.data_pin = pin;
  lcd.data_port = port;

  Lcd_init(&lcd);

  return lcd;
}
```

```
38
39 /**
40  * Initialize 16x2-lcd without cursor
41  */
42 void Lcd_init(Lcd_HandleTypeDef * lcd)
43 {
44   if(lcd->mode == LCD_4_BIT_MODE)
45   {
46       lcd_write_command(lcd, 0x33);
47       lcd_write_command(lcd, 0x32);
48       lcd_write_command(lcd, FUNCTION_SET | OPT_N);        // 4-bit mode
49   }
50   else
51     lcd_write_command(lcd, FUNCTION_SET | OPT_DL | OPT_N);
52
53
54   lcd_write_command(lcd, CLEAR_DISPLAY);              // Clear screen
55   lcd_write_command(lcd, DISPLAY_ON_OFF_CONTROL | OPT_D);    // Lcd-on, cursor-off
       , no-blink
56   lcd_write_command(lcd, ENTRY_MODE_SET | OPT_INC);      // Increment cursor
57 }
58
59 /**
60  * Write a number on the current position
61  */
62 void Lcd_int(Lcd_HandleTypeDef * lcd, int number)
63 {
64   char buffer[11];
65   sprintf(buffer, "%d", number);
66
67   Lcd_string(lcd, buffer);
68 }
69
70 /**
71  * Write a string on the current position
72  */
73 void Lcd_string(Lcd_HandleTypeDef * lcd, char * string)
74 {
75   for(uint8_t i = 0; i < strlen(string); i++)
76   {
77     lcd_write_data(lcd, string[i]);
78   }
79 }
80
81 /**
82  * Set the cursor position
83  */
84 void Lcd_cursor(Lcd_HandleTypeDef * lcd, uint8_t row, uint8_t col)
```

```
85 {
86   #ifdef LCD20xN
87   lcd_write_command(lcd, SET_DDRAM_ADDR + ROW_20[row] + col);
88   #endif
89 
90   #ifdef LCD16xN
91   lcd_write_command(lcd, SET_DDRAM_ADDR + ROW_16[row] + col);
92   #endif
93 }
94 
95 /**
96  * Clear the screen
97  */
98 void Lcd_clear(Lcd_HandleTypeDef * lcd) {
99   lcd_write_command(lcd, CLEAR_DISPLAY);
100 }
101 
102 void Lcd_define_char(Lcd_HandleTypeDef * lcd, uint8_t code, uint8_t bitmap[]){
103   lcd_write_command(lcd, SETCGRAM_ADDR + (code << 3));
104   for(uint8_t i=0;i<8;++i){
105     lcd_write_data(lcd, bitmap[i]);
106   }
107 
108 }
109 
110 
111 /************************************ Static function definition
    ************************************/
112 
113 /**
114  * Write a byte to the command register
115  */
116 void lcd_write_command(Lcd_HandleTypeDef * lcd, uint8_t command)
117 {
118   HAL_GPIO_WritePin(lcd->rs_port, lcd->rs_pin, LCD_COMMAND_REG);     // Write to
     command register
119 
120   if(lcd->mode == LCD_4_BIT_MODE)
121   {
122     lcd_write(lcd, (command >> 4), LCD_NIB);
123     lcd_write(lcd, command & 0x0F, LCD_NIB);
124   }
125   else
126   {
127     lcd_write(lcd, command, LCD_BYTE);
128   }
129 
130 }
```

```
131
132 /**
133  * Write a byte to the data register
134  */
135 void lcd_write_data(Lcd_HandleTypeDef * lcd, uint8_t data)
136 {
137   HAL_GPIO_WritePin(lcd->rs_port, lcd->rs_pin, LCD_DATA_REG);      // Write to
      data register
138
139   if(lcd->mode == LCD_4_BIT_MODE)
140   {
141     lcd_write(lcd, data >> 4, LCD_NIB);
142     lcd_write(lcd, data & 0x0F, LCD_NIB);
143   }
144   else
145   {
146     lcd_write(lcd, data, LCD_BYTE);
147   }
148
149 }
150
151 /**
152  * Set len bits on the bus and toggle the enable line
153  */
154 void lcd_write(Lcd_HandleTypeDef * lcd, uint8_t data, uint8_t len)
155 {
156   for(uint8_t i = 0; i < len; i++)
157   {
158     HAL_GPIO_WritePin(lcd->data_port[i], lcd->data_pin[i], (data >> i) & 0x01);
159   }
160
161   HAL_GPIO_WritePin(lcd->en_port, lcd->en_pin, 1);
162   DELAY(1);
163   HAL_GPIO_WritePin(lcd->en_port, lcd->en_pin, 0);     // Data receive on falling
      edge
164 }
```

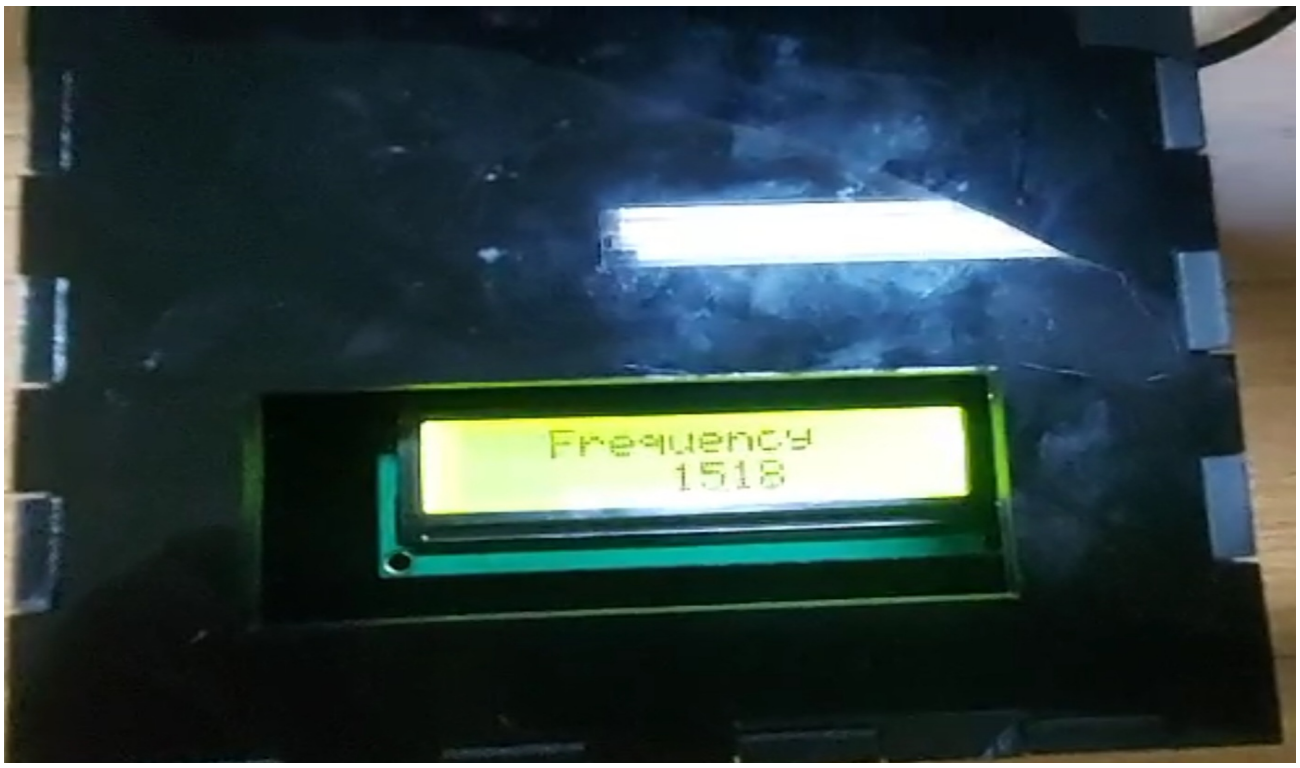


Figure 6: LCD in the final prototype

5.5 Microcontroller Pinout

The final microcontroller pinout after setting all the GPIO pins and interfacing the ADC and LCD is as follows -

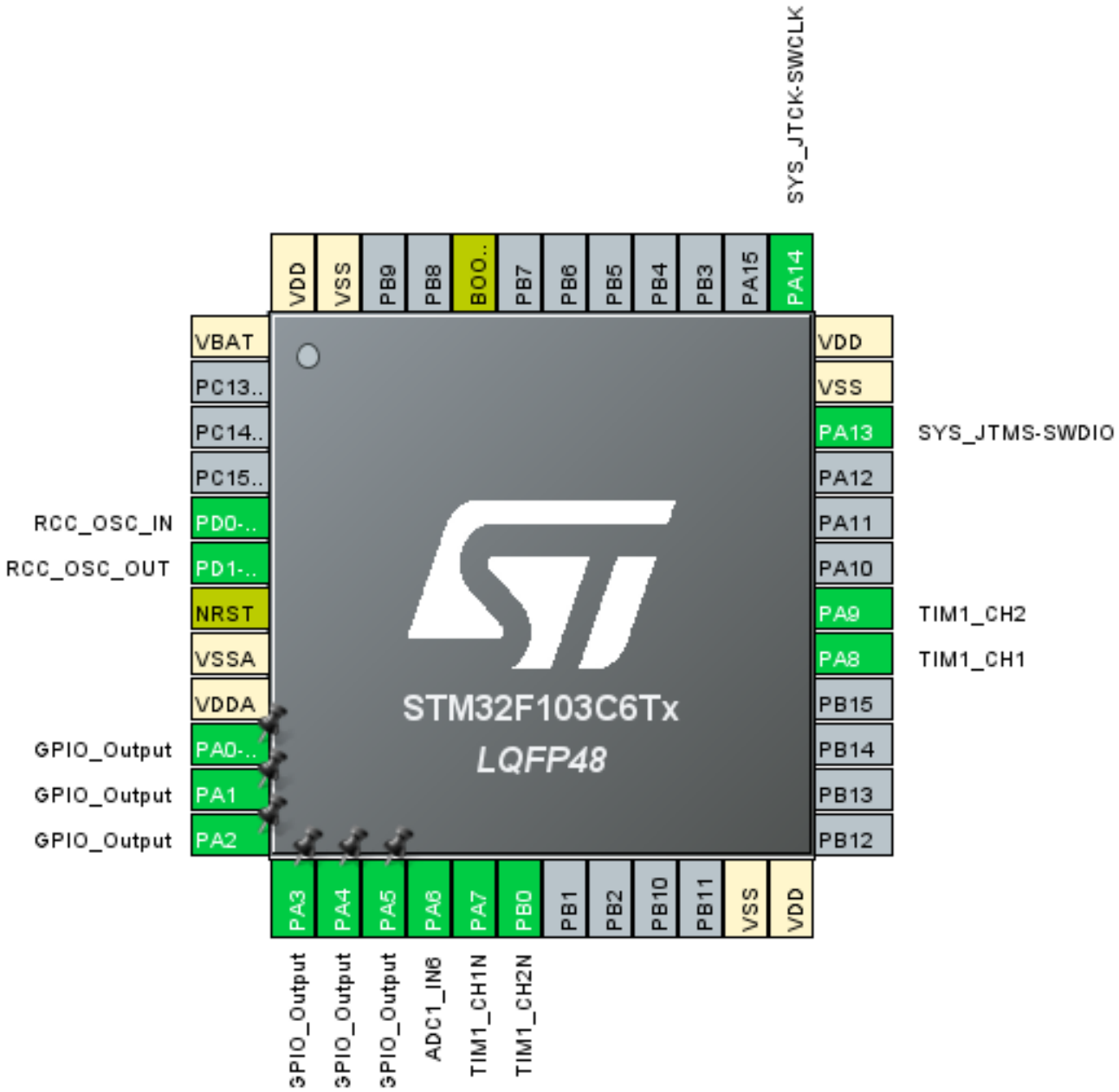


Figure 7: STM32 pinout

6 Schematics

Having selected the class D amplifier as the base of our power amplifier, we started making the overall complete circuitry for our prototype. As we have seen so far, our circuit majorly consists of 3 parts, viz

1. Microcontroller - For generating PWM signals.
2. Gate Drivers - For interfacing microcontroller with MOSFETs.
3. Full-bridge MOSFET

In our circuit, we have directly used Blue-Pill, which is a miniature version of NUCLEO-F103RB STM32 evaluation board. The reason for going with blue-pill is simply because it contains all of the features of NUCLEO board we required and moreover, it is smaller in size. However, one main issue while using blue-pill is that it does not have emulator part, which is present in NUCLEO board, making it difficult to dump code onto it. We had to use the ST-Link debugger everytime we wanted to dump our code onto blue-pill.

Moving onto the next part of our circuit, we have Gate Drivers. We chose the 2ED020I12-FI IC manufactured by Infineon. The IC can sustain high voltage up to 1.2kV, supplying currents up to 1A or -2A. The PWM signals generated by the blue-pill were then fed to the gate drivers through the RC filter. This reduces the spike levels due to switching in the microcontroller. A major part of the gate driver circuitry is the **Bootstrap circuit** at the high-side output of the gate driver IC. The bootstrap circuitry is used to supply bias to high-side FET. When the low-side FET is ON, the HS pin and source of the high-side FET are pulled to the ground. As a result, the bootstrap capacitor is charged to Vdd. When the high-side FET is ON in the next cycle, the bootstrap capacitor discharges some of its charge to the gate of FET, thereby creating some positive Vgs and turning on the FET. Schottky diode is used as the bootstrap diode because of its ability to switch at high frequencies. Calculation of bootstrap capacitance is an important step while designing the bootstrap circuit. The value of bootstrap capacitor is calculated according to the given formula -

$$C_g = \frac{Q_g}{V_{Q_{1g}}}$$

where Q_g: gate charge $\left(\text{MOSFET's datasheet}\right)$

$$V_{Q_{1g}} = V_{DD} - V_{BootDiode}$$

where $V_{BootDiode}$: forward voltage drop across the boot diode.

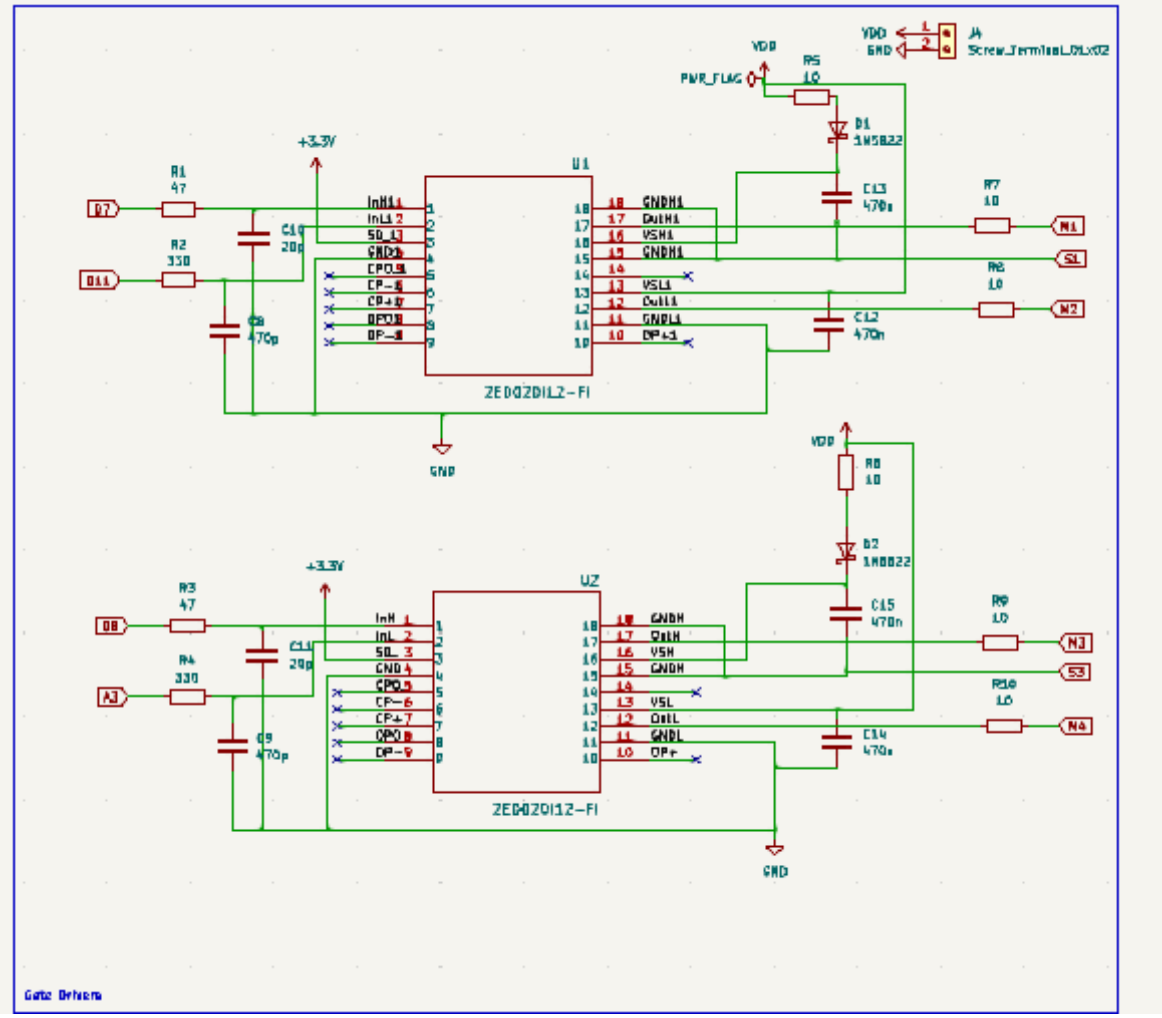


Figure 8: Gate Driver Circuit

For IRF540N, the value of gate charge is 66nC, and for 1N5822 Schottky diode, the forward voltage drop is 0.525 V, and Vdd = 15V. Plugging the values in above formula gives, Cg = 4.56 nF The minimum value of bootstrap capacitance should be chosen such that, Cboot = 10xCg Therefore, the value of bootstrap capacitor should be atleast 45nF. Apart from that, we also need to add a bypass capacitor value to Vdd, and the value of bypass capacitor should be atleast 10x the bootstrap capacitance. Therefore, Cbypass should be atleast 450nF.

Now, finally, once we have the pulses at the output of gate drivers, they are passed onto the respective gates of the FETs. The pulses are passed onto the gates of FETs through a resistor. The output is observed between the source of the two high-side FETs. An electrolytic capacitor is connected between the Vdd and GND and ceramic capacitors need to be connected betweent the drain and ground of each FET. A simple low pass filtering is performed at the output to

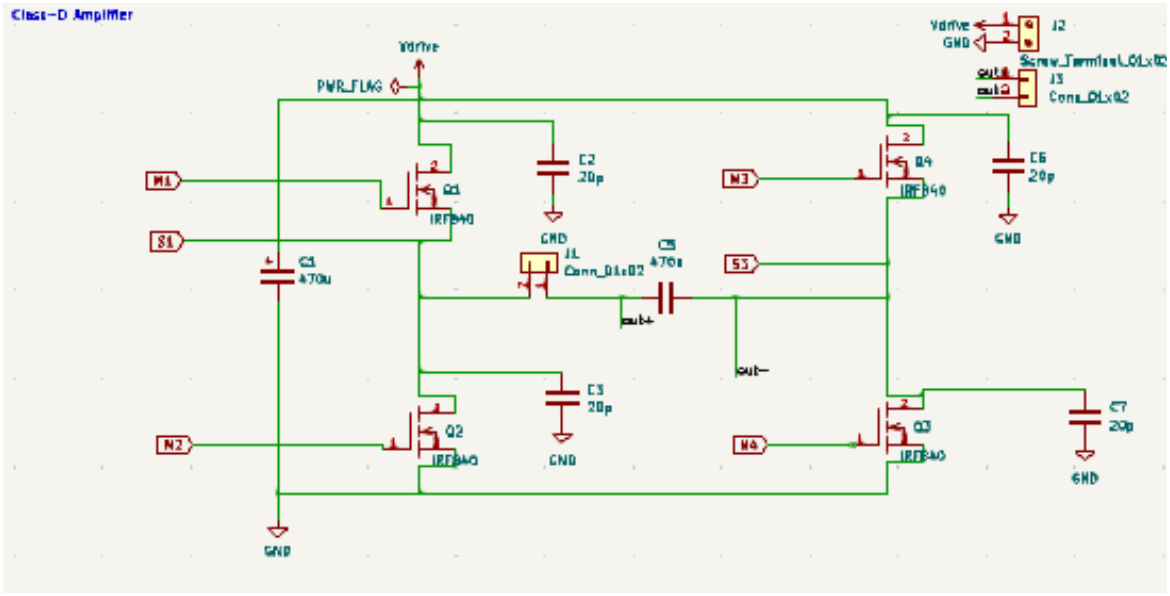


Figure 9: Class-D amplifier

remove spikes due to switching.

We have also added a provision in our circuit to control the frequency using a knob. The knob is connected to a 10kΩ potentiometer. Based on the voltage at the middle terminal of the the voltage is sensed by the ADC in the blue-pill and it is mapped to the corresponding frequency value. The current frequency value is displayed using an LCD. Finally, putting all parts together, we get our final circuit schematics.

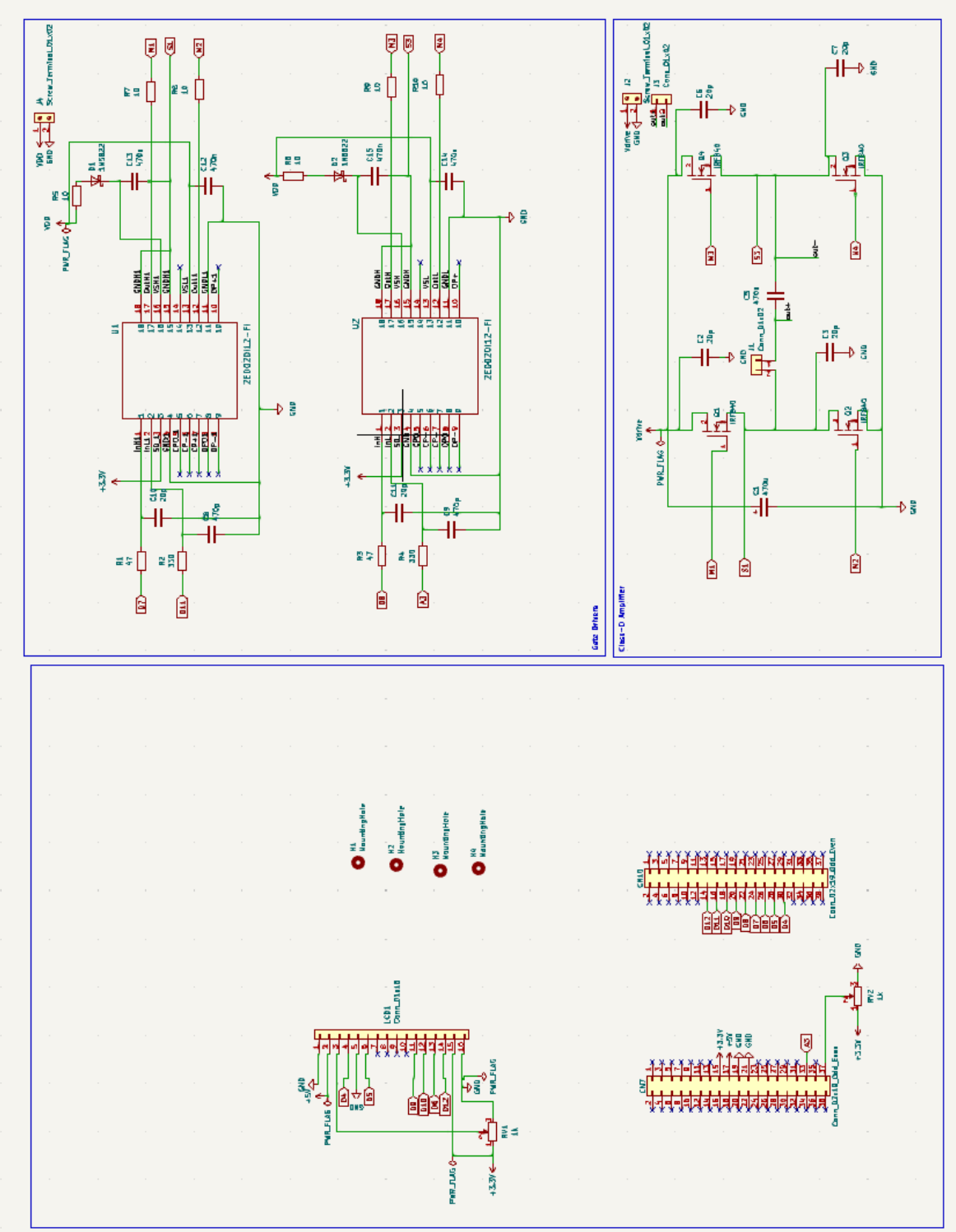


Figure 10: Full circuit schematics

7 Intermediate Testing

7.1 PWM Generation

As descibed in 5.1, we generated the PWM signals using STM32 and tested. The output is shown in the image below. Entire video can be accessed using this link.

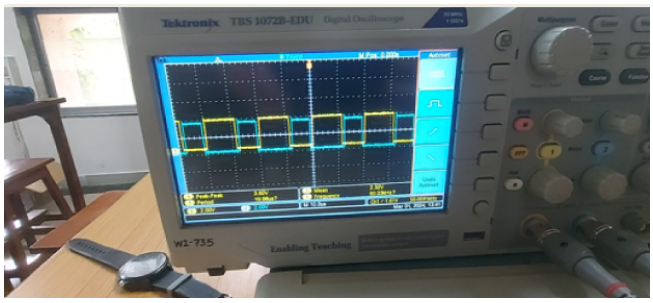


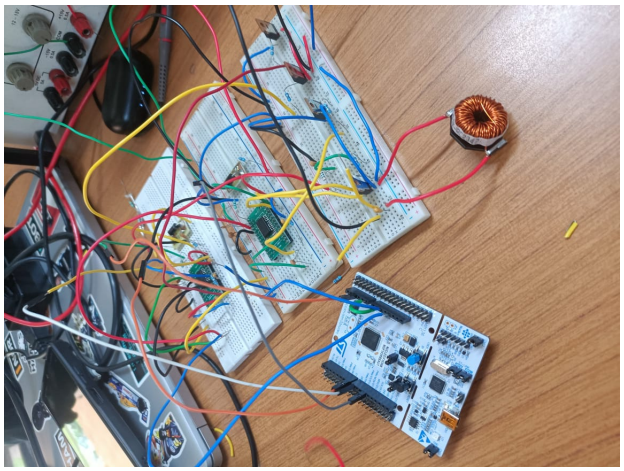
Figure 11: PWM testing

7.2 ADC interfacing

We were able to interface a potentiometer with the microcontroller using an ADC successfully. The link to demonstration can be found here.

7.3 Breadboard Testing

Link to breadboard testing video.



8 PCB Design

We designed the PCB using the KiCAD software. Various layout rules such as obtuse angles between tracks, decoupling capacitors, and correct placement of components were taken care of. We have used both SMD and THT components in our design. We have majorly used SM components for decoupling capacitors and the gate driver IC. We have used screw terminals for connecting the inputs to the PCB. The blue pill is mounted on the PCB using a breakout board.

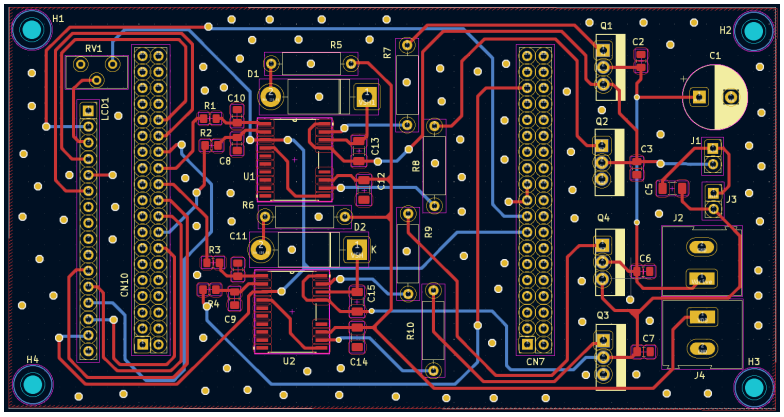


Figure 12: PCB Layout

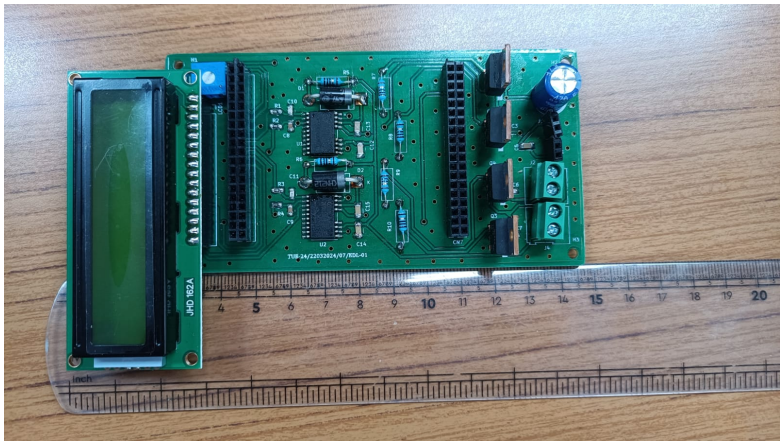


Figure 13: PCB with all components

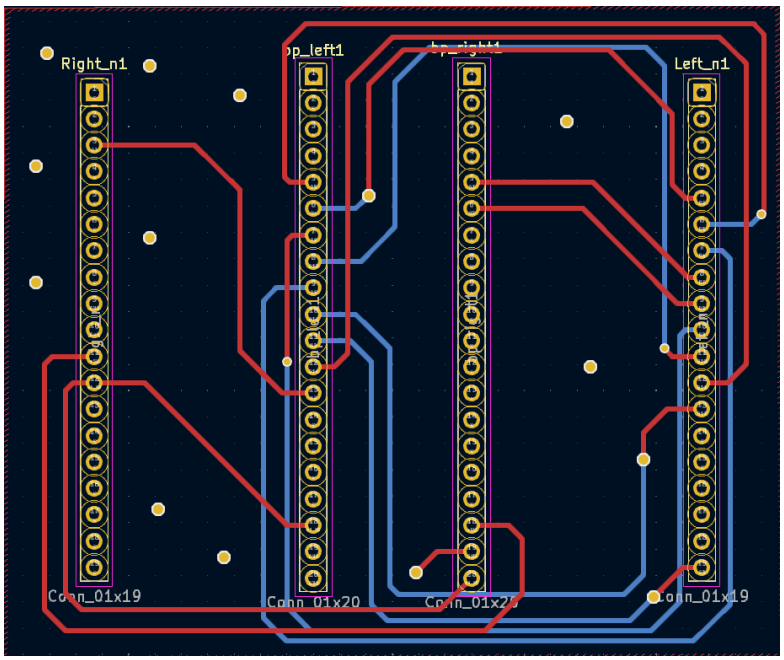


Figure 14: PCB for breakout board

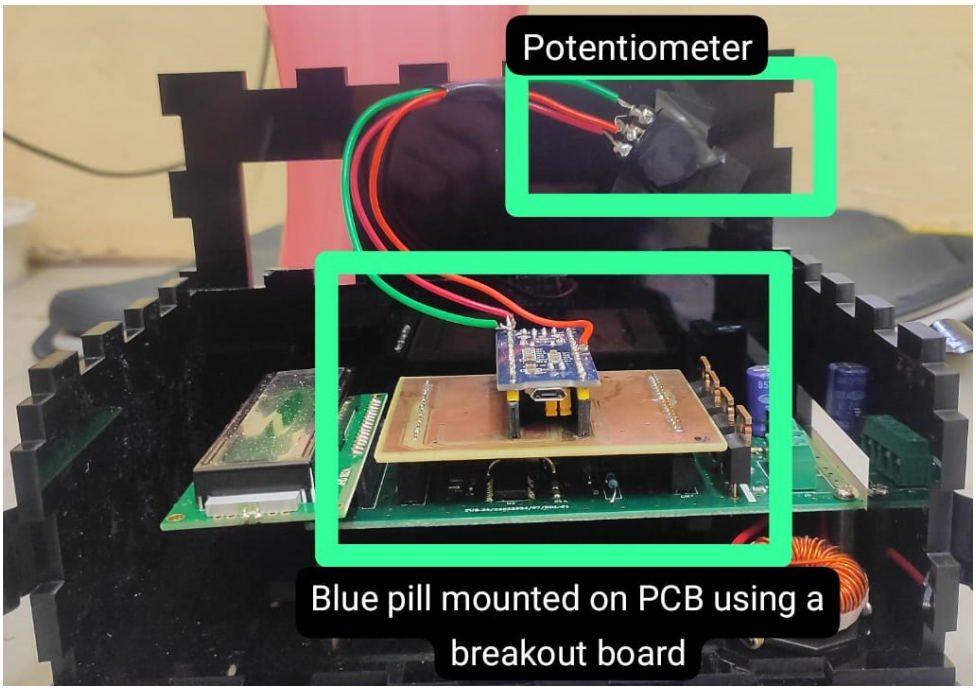


Figure 15: Final PCB

9 CAD Design

For our CAD design, we initially planned a 3D box structure using Fusion 360 software. The design process involved careful planning to ensure accessibility to certain components from the outside world while the prototype remains inside the box.

9.1 Design Requirements

1. **LCD Display:** The design incorporates an open area for the LCD display to allow users to observe the frequency without obstruction.
2. **Input and Outputs:**
   - **Power Supply:** A 5V supply is provided through a USB cable, which is also utilized for burning code onto the STM32 microcontroller.
   - **External Supplies:** V_{drive} and V_{dd} are supplied externally.
   - **Output Access:** The output across the capacitor is also accessible through an opening on the 3D box.
   - **Potentiometer:** A potentiometer is mounted on the box which will be used for controlling the frequency of the prototype. Thus, certain areas must be opened up on the box for the same.

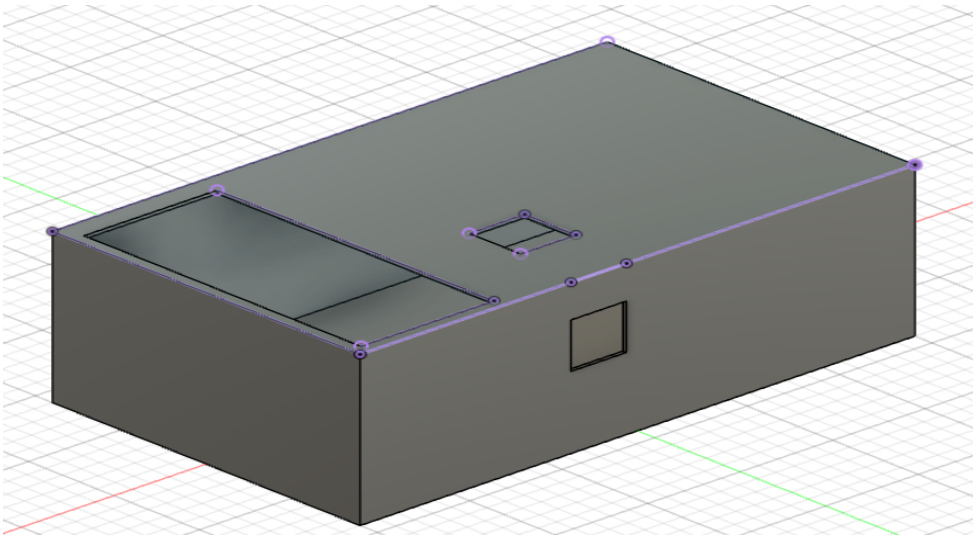


Figure 16: 3D box CAD design

However, after discussing the design with the faculty we concluded that 3D printing the box is not possible, and hence we would have to use acrylic sheets laser cutting to make our box.

Now, makercase was used for making the dxf file which has the thickness of the sheet, length, and width for each side of the box. After importing this dxf file into the Fusion 360 software, we cut out the same regions as in the 3D box design.

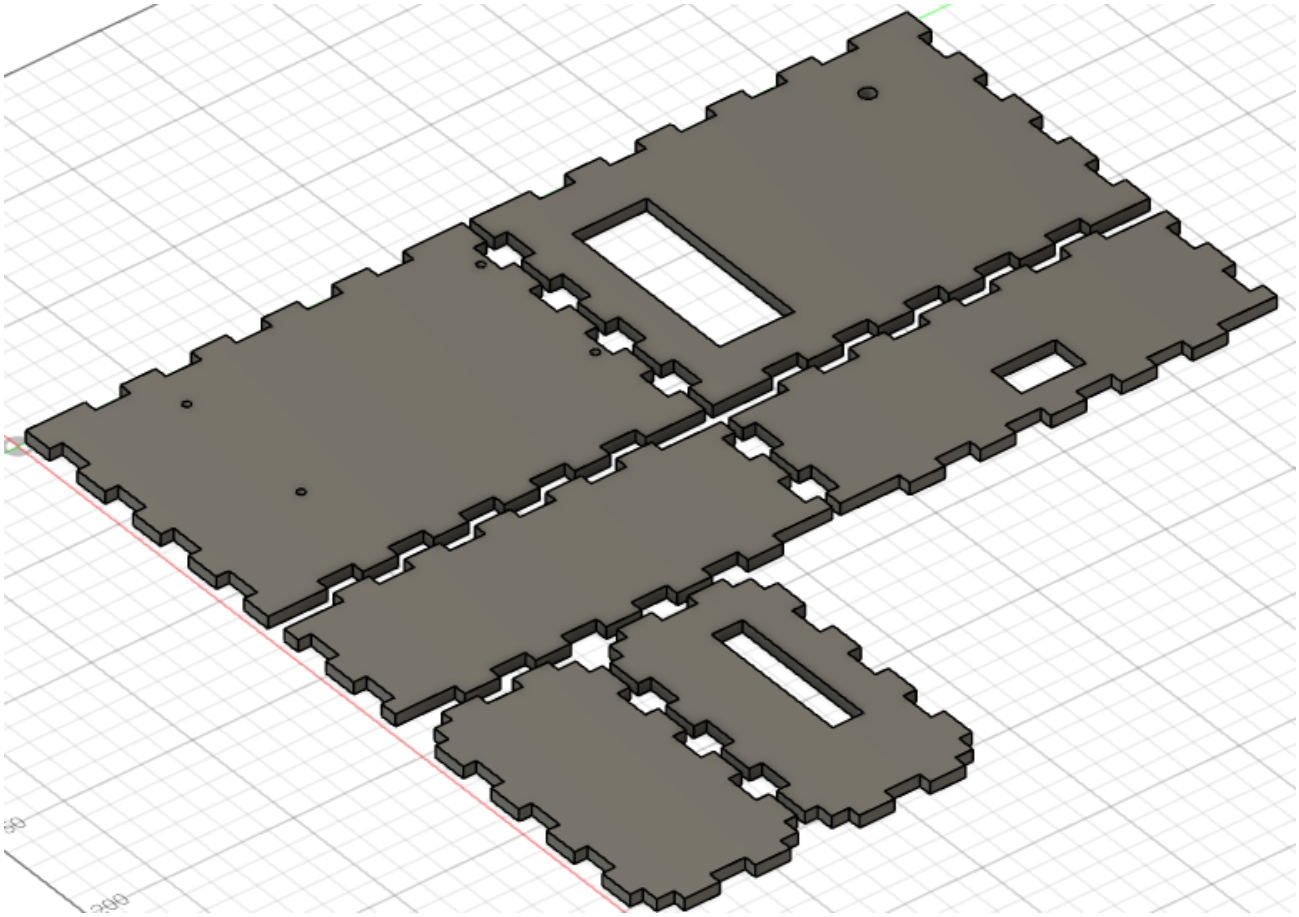
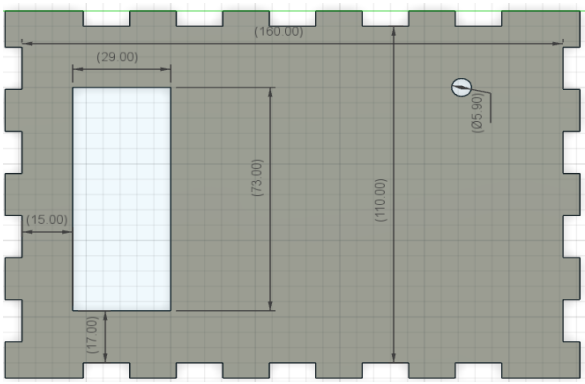
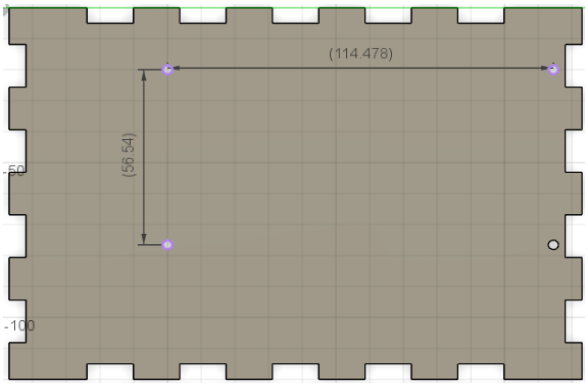


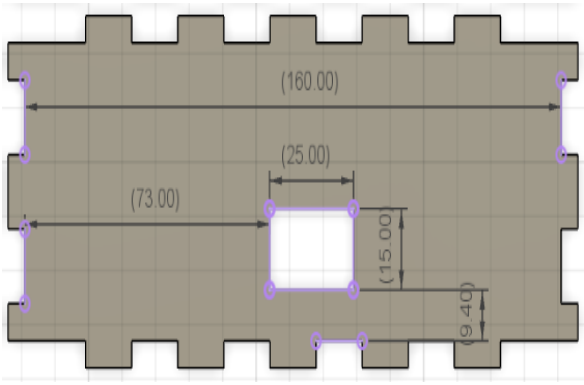
Figure 17: Acrylic sheets design for the box



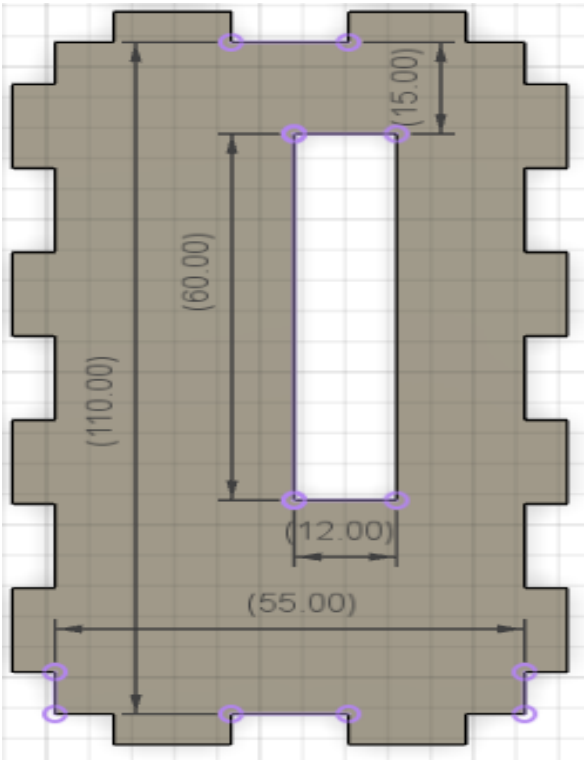
(a) Top side



(b) Bottom side



(c) Front view side



(d) Side view

Figure 18: Dimensions and open areas in the design.

10 Final Prototype

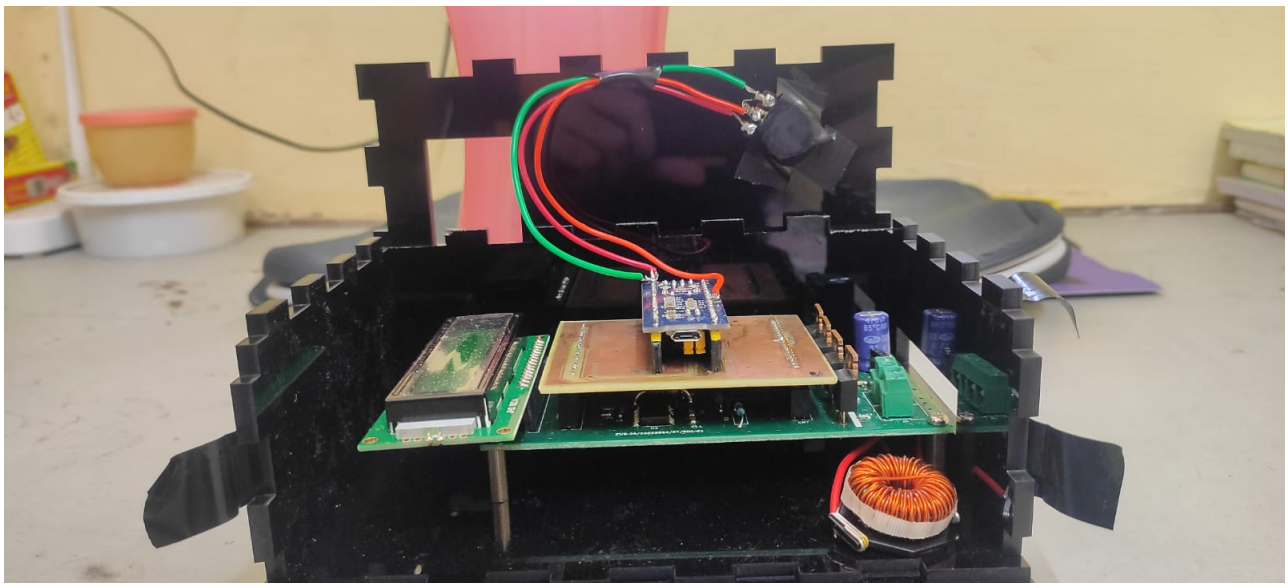


Figure 19: Final Prototype

11 Results

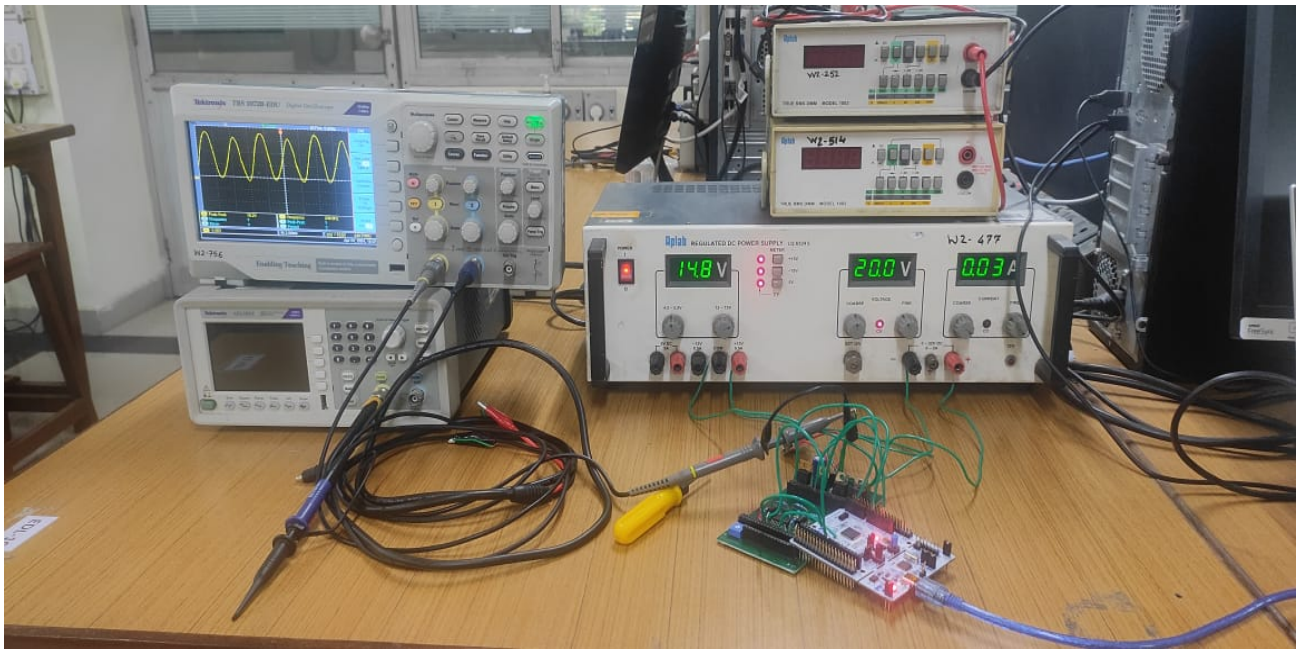


Figure 20: Final output Sine waveform

Our final output waveform is a sine wave meticulously designed for the characterization of magnetic materials. The waveform is derived from three power sources: V_{drive}, Vdd, and a 5V supply via USB to the Nucleo board. The output signal is measured across a 470nF capacitor. Our setup achieves a current rating of 1.4 A, with the peak-to-peak voltage (Vpp) reaching up to 30 V at a frequency of 500Hz.

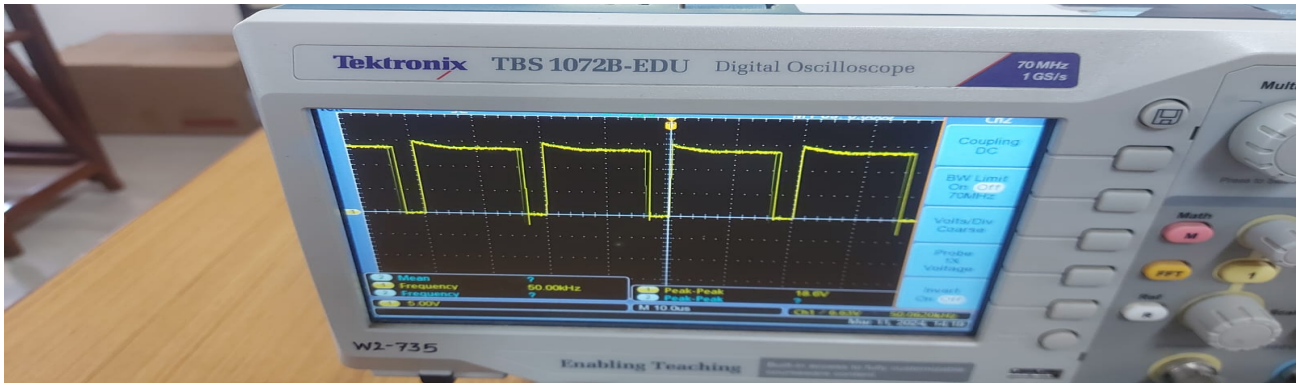


Figure 21: Gate Driver output

The output from the gate driver pertains to the high side. As anticipated, it effectively amplifies the PWM signal from the microcontroller. We are able to achieve 18.6 V of peak-to-peak voltage output from the gate driver.

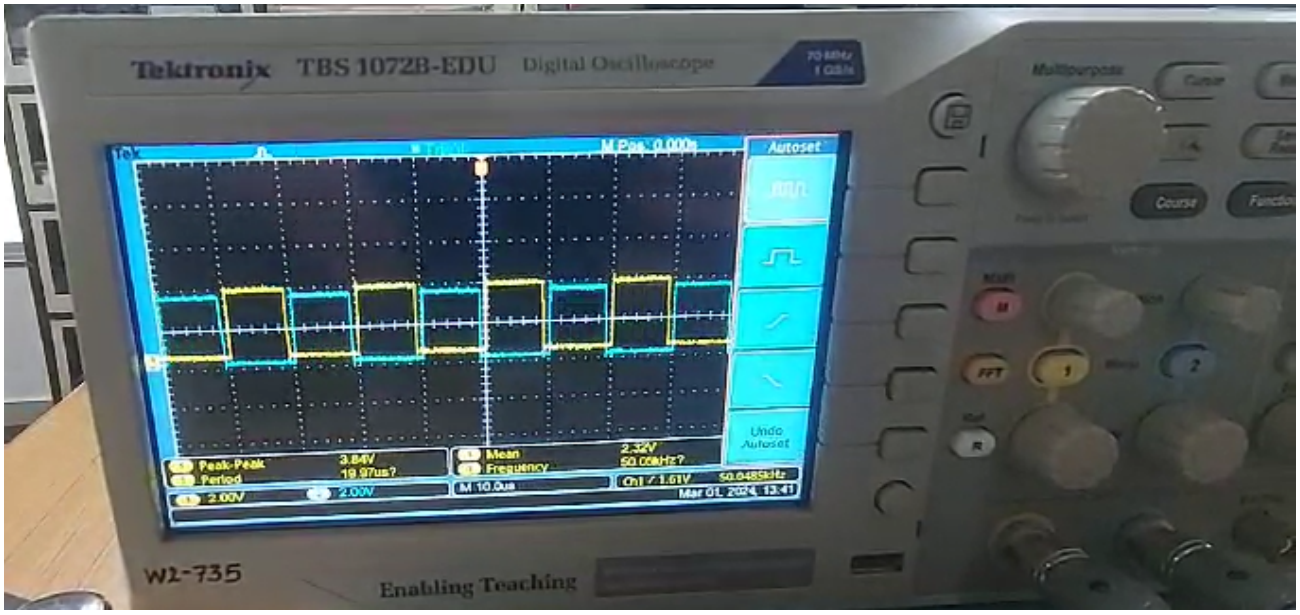


Figure 22: Microcontroller output

Above is the microcontroller PWM output. This goes as an input to our gate drivers. This PWM peak-to-peak voltage is 3.84 V. This is then amplified to 18.6 V by the gate drivers.