BILKENT UNIVERSITY

ENGINEERING FACULTY

DEPARTMENT

OF

ELECTRICAL and ELECTRONICS ENGINEERING

EEE 299

SUMMER TRAINING REPORT

Mehmet Emre Uncu

22003884

Performed at

Roketsan A.S

15.08.2023 – 12.09.2023

Table of Contents

1 Introduction

In this report, I will give information about my summer internship and the project I did. I completed my summer internship at Roketsan. In this company, which accepts approximately 100-150 interns every semester, I had the opportunity to work with interns from different universities and departments and many engineers who are experts in their fields. At the end of my internship, I had the opportunity to adapt the knowledge from my education life to real-life systems and gain much experience.

Motors are electromechanical devices that convert electrical energy into mechanical energy. To achieve this conversion, the electrical input to the motor must be controlled. It is possible to create this input logic using a controller. The task given to me during the internship was to create a Very High Speed Integrated Circuit Hardware Description Language (VHDL) code to drive a Brushless Direct Current (BLDC) motor using Field Programmable Gate Array (FPGA) and then implement it and verify that it works.

In this report, I will briefly explain Roketsan company, its products, and production systems. Afterward, I will give the necessary information about my department and supervisor, who helped me during my internship. Then, I will provide detailed step-by-step information about the project I worked on and completed throughout the internship. Finally, under the title of performance and outcomes, I will talk about how I worked during the internship, how I coped with complex engineering problems, my ethical and professional responsibilities, how I gained knowledge, how I translated my existing knowledge into practice, and the terms Diversity, Equity, and Inclusion (DEI) that I observed during the internship. I will finish the report with a summary of my internship.

2 Company Information

2.1 About the company

Roketsan was established on June 14, 1988, with the decision of the Defense Industry Executive Committee with the aim of "meeting the rocket and missile needs of the Türk Silahlı Kuvvetleri (TSK) and having a leading institution in the design, development, and production of rockets and missiles in our country" [1].

2.2 About the products and production systems of the company

Roketsan provides services on the design and production of rockets, missiles, and munitions in many areas, such as land, naval, and air defense systems, precision-guided systems, as well as space systems, ballistic protection, navigation, and subsystems [2]. It also provides services on ballistic protection testing, environmental condition testing, Electromagnetic Interference (EMI) / Electromagnetic Compatibility (EMC) testing, and flight testing [3]. With more than 4000 employees, it completes the design and production of the necessary mechanical parts of all products. Then, it works on producing and integrating the hardware and software parts. It aims to ensure the supply of products by testing the products step by step in all processes during production without allowing the slightest error.

2.3 About your department

I completed my internship at the Control Systems Development Department. In this department, where electrical and electronics, mechanical, computer, and control engineers are located, control systems of missiles are studied. I worked on embedded software development in this department, where software, circuit design, mechanical design, and testing processes are carried out by engineers who are experts in their fields.

2.4 About your supervisor

Name and Last Name:	Olgun Can Pektaş
Job Title:	Senior Engineer
University of B.S. Graduation:	Hacettepe University
Department of B.S. Degree:	Electrical and Electronics Engineering
Year of B.S. Graduation:	2017
Email:	can.pektas@roketsan.com.tr
Phone Number:	+90 (555) 486 21 01

3 Work Done

The task given to me was to drive a BLDC motor using FPGA. I was asked to create this project with VHDL on Vivado. Afterward, I would complete the project by embedding the code I created with a demo FPGA and driver card, the specifications of which I was not allowed to share, and driving a BLDC motor so that its speed was adjusted at the desired level.

In this part of the report, I will tell you about the steps I took about my project after talking about the research I did on the differences of motors before starting the project.

3.1 Differences of Motor Types

Engines are machines that convert a type of energy into mechanical energy. Engines that use electrical energy to obtain motion energy are called electric motors. Electric motors are divided into two categories according to the current type: alternating current (AC) and direct current (DC) motors. AC and DC motors differ according to their operation and application. Therefore, which one to use varies depending on the purpose and place of use.

An AC motor is an electric motor driven using alternating current. AC motors have lower maintenance costs and longer lifespans due to fewer moving parts and less friction. Speed controls and structures are more complex than DC motors. AC motors are generally used in situations requiring constant high speed and large power [4].

A DC motor is an electric motor driven using direct current. They generally require maintenance due to friction and wear, and they have shorter lifespans. The structure of DC motors is generally simpler than AC motors, and their speed can be easily adjusted by the supply voltage and field current. This is ideal for applications requiring precise speed and torque control. They are also used to lift or move heavy loads because they can produce high torque at startup [4].

3.2 Differences of DC Motor Types

DC motors are divided into two: Brushed DC motor and Brushless DC motor. Differences in their structures lead to changes in their intended use and places of use in industry.

Brushed DC motors contain fewer components and have a simpler structure. This reduces the production cost. The brush and the commutator where it contacts are prone to wear due to friction, therefore it requires maintenance, and the motor life is short. In addition, heat and energy loss occurs due to this friction, which causes the efficiency to decrease. They are preferred in toys, small household appliances, and low-power applications [5].

Brushless DC motors have higher efficiency, consume less energy, and have fewer heating problems. Since there are no brushes, there is no energy or heat loss due to friction, they do not require maintenance and have a longer life. Their ability to provide precise speed and torque control is better, so they are used in more complex applications. Production costs are high. Electric vehicles, robots, drones, and gimbal systems can be given as examples of places of use [5].

3.3 Basics of Three-Phase BLDC Motor

A Brushless DC motor consists of two main parts: the stator, which has a certain number of coils on the outside, and the rotor, which has a fixed double-pole magnet inside. BLDC motor is controlled by the electromagnetic force that occurs when current passes through the coils on the stator. With the appropriate current, the pole of a coil can be determined, and it interacts with the magnet on the rotor thanks to the push and pull force created.

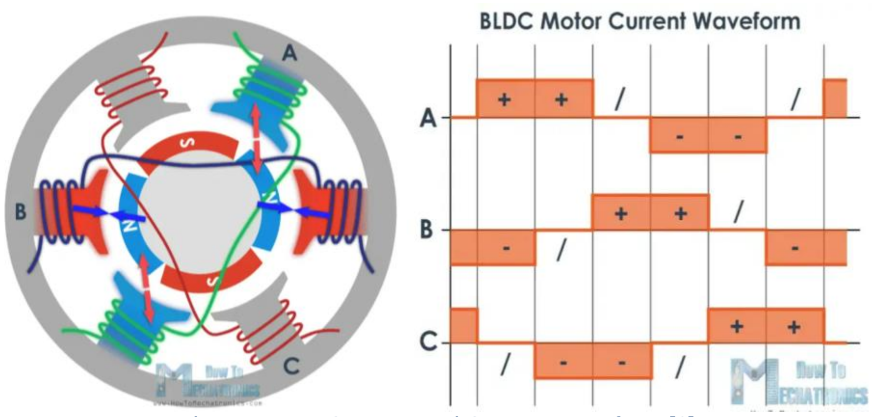


Figure 1: BLDC Motor and Current Waveform [6]

As shown in Figure 1, when successive coils are activated sequentially, rotation action occurs due to the force interaction between the rotor and the electromagnet. By applying current to the coils in the direction of rotation, the opposite pole is obtained, and a pulling force is applied to the rotor. Additionally, reverse current is applied to the passed coil, thus achieving the same polarity, which creates a push force on the rotor. To increase the efficiency of the motor, opposite coils are connected to each other, thus the same forces are applied to the opposite side of the rotor. Thanks to these operations, a continuous rotation movement is achieved.

3.4 Three-Phase BLDC Motor Driver

A driver circuit is required to activate the coils of the three-phase BLDC. In this circuit, which basically consists of 6 MOSFETs, the behavior of the coils on the rotor is regulated by sending the Pulse Width Modulation (PWM) coming from the controller to the appropriate MOSFETs.

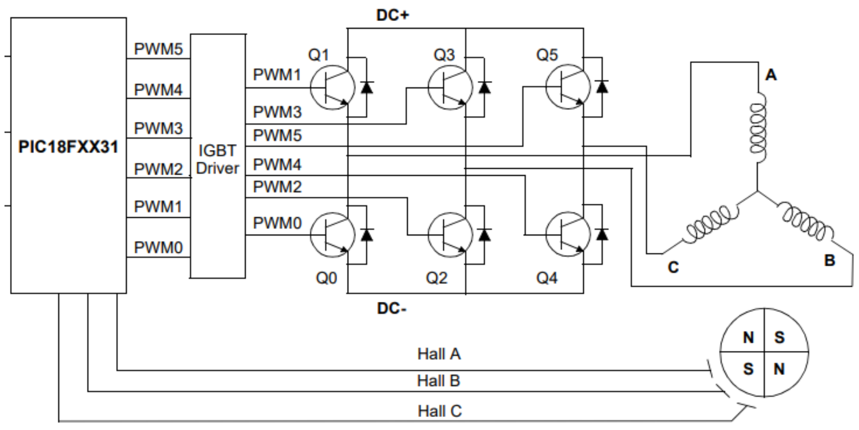


Figure 2: BLDC Motor Driver Diagram [7]

Position information of the rotor is required to determine which MOSFETs should be turned on and when. The Hall effect sensors placed inside the stator are generally used to obtain this position information. In this system consisting of 3 sensors, the position is determined according to the information coming from the sensors, and the controller uses this information to create the appropriate output to go to the MOSFETs.

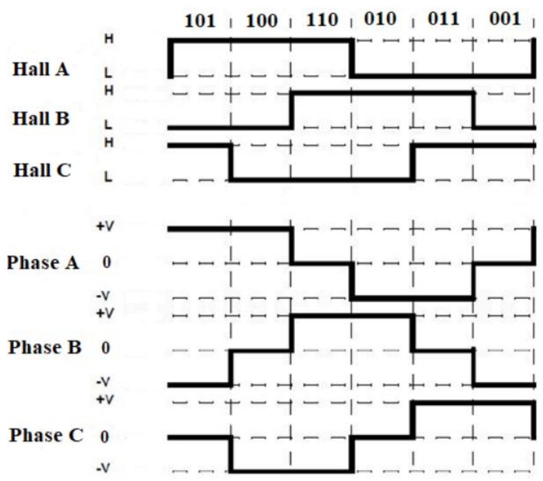


Figure 3: Symbolic Hall and Phase waveforms [8]

Figure 3 shows the information coming from the Hall effect sensors and the phase voltages obtained. After the six outputs from the controller are matched with the appropriate MOSFETs, the phases of the motor are controlled using the information from the Hall effect sensors.

3.5 Pulse Width Modulation (PWM)

It is a technique in which a series of pulses is used to turn a signal on or off for a specified period of time. This is used to control the intensity or average of the signal.

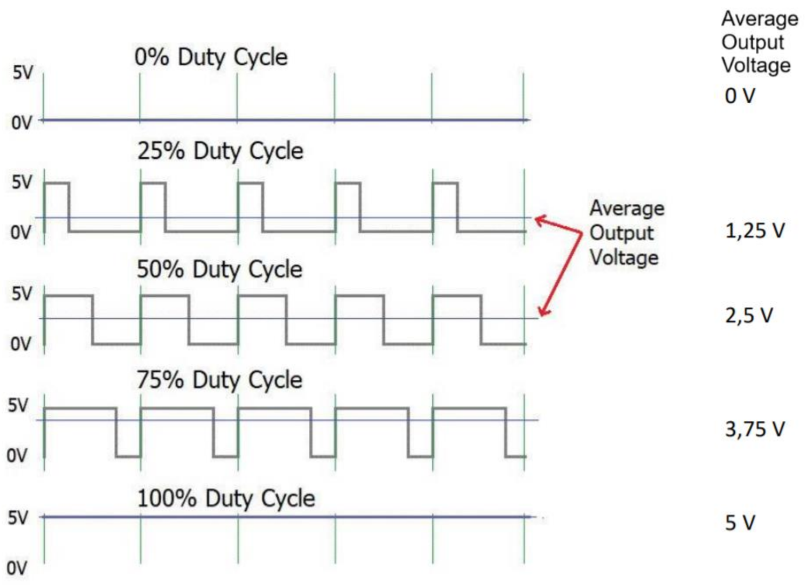


Figure 4: Pulse Width Modulation [9]

The PWM signal creates a voltage level that is used to adjust the speed of the motor. As the pulse width increases, the motor speed increases, and as the pulse width decreases, the motor speed decreases.

3.6 Coding Part

I started the coding part by writing a module that creates a PWM. After completing this part, I created a module that will process the information coming from the Hall effect sensors. Finally, I created the top-level module by bringing these modules together. Finally, I simulated it on Vivado and verified its accuracy.

3.6.1 PWM Generator

The PWM module takes the FPGA clock frequency, PWM frequency, duty value, and clock as input and gives the generated PWM signal as output.

```
entity PWM is
    generic   (  I_FREQ_CLK:    in integer;
                 I_FREQ_PWM:    in integer);
    port      (  I_DUTY:        in integer;
                 CLK:           in std_logic;
                 O_PWM_OUT:    out std_logic);
end PWM;
```

Figure 5: Inputs and Outputs of PWM module

After researching and learning what PWM is, I thought about how I could create PWM. First, I defined a 'limit' variable using the clock frequency of the FPGA given to me and the desired PWM frequency. The 'timer' variable, which will increase by one at each clock rising edge, will be reset, and a new pulse will be started when it reaches this 'limit'. I also defined the 'hightime' variable with a simple percentage calculation to make duty adjustments. The output will be '1' until the timer variable reaches this value, and '0' after this value.

```
limit <= I_FREQ_CLK/I_FREQ_PWM;
hightime <= (limit/100)*I_DUTY;
if rising_edge(CLK) then
    if timer = limit-1 then
        timer <= 0;
    elsif I_DUTY < 10 then
        O_PWM_OUT <= '0';
        timer <= timer +1;
    elsif I_DUTY > 90 then
        O_PWM_OUT <= '1';
        timer <= timer +1;
    elsif timer < hightime then
        O_PWM_OUT <= '1';
        timer <= timer +1;
    else
        O_PWM_OUT <= '0';
        timer <= timer +1;
    end if;
end if;
```

Figure 6: PWM Generator

I was also asked to set the duty to 100 when it is over 90 and 0 when it is below 10, so I added two elsif conditions to ensure it works appropriately.

3.6.2 Processing Information from Hall Effect Sensors

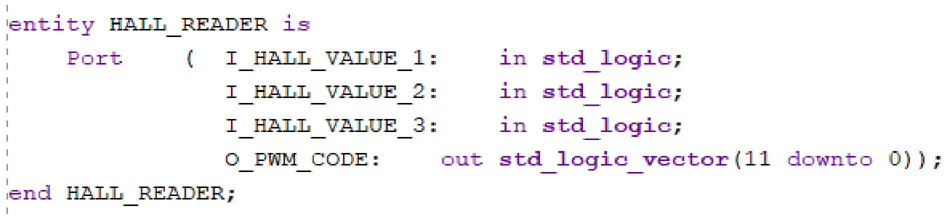
To process the information coming from the Hall effect sensors, I first created a table using the Hall table of the motor I will use.

H_A	H_B	H_C	A	B	C	Q0	Q1	Q2	Q3	Q4	Q5	PWM
0	0	0	0	0	0	00	00	00	00	00	00	000000000000
0	0	1	0	-	+	00	00	11	00	01	10	000010000110
0	1	0	-	+	0	11	00	01	10	00	00	100001100000
0	1	1	-	0	+	11	00	00	00	01	10	100000000110
1	0	0	+	0	-	01	10	00	00	11	00	011000001000
1	0	1	+	-	0	01	10	11	00	00	00	011010000000
1	1	0	0	+	-	00	00	01	10	11	00	000001101000
1	1	1	0	0	0	00	00	00	00	00	00	000000000000

Table 1: Hall to PWM Table

This table includes the state of the phases according to the Hall values, the value of the signal to be given to the MOSFETs, and finally, a 12-bit PWM code that indicates the MOSFET states together for me to use in the code. When the value under the MOSFETs is ‘00’, a ‘0’ signal will be sent, when it is ‘01’, PWM, when it is ‘10’, PWM's complement, and when it is ‘11’, a ‘1’ signal will be sent to the MOSFET.

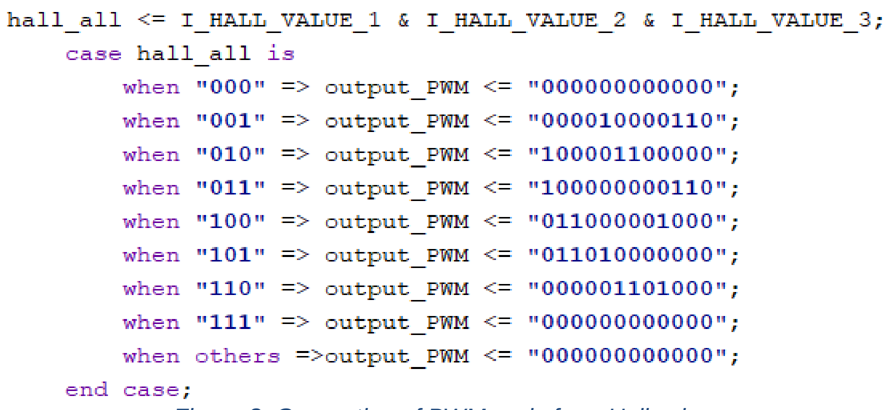
HALL_READER module takes the three Hall effect sensor values as input and gives the generated 12-bit PWM-code as output.



```
entity HALL_READER is
    Port     (  I_HALL_VALUE_1:    in std_logic;
                I_HALL_VALUE_2:    in std_logic;
                I_HALL_VALUE_3:    in std_logic;
                O_PWM_CODE:     out std_logic_vector(11 downto 0));
end HALL_READER;
```

Figure 7: Inputs and Outputs of HALL_READER module

The module presented in Figure 8 determines the appropriate PWM-code value using the case statement according to the 3 Hall effect sensor values as follows:

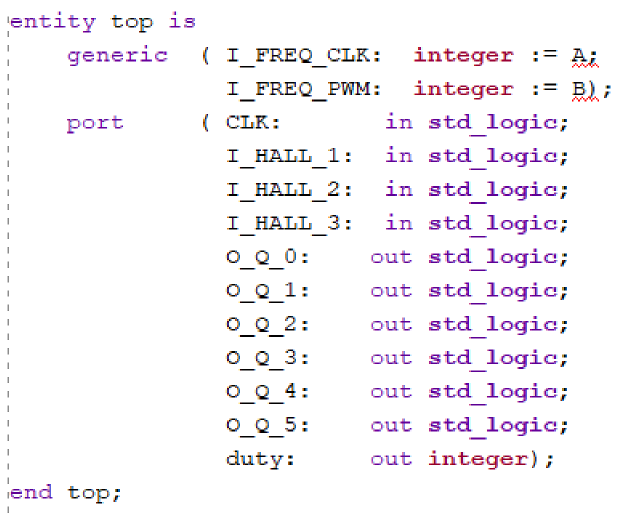


```
hall_all <= I_HALL_VALUE_1 & I_HALL_VALUE_2 & I_HALL_VALUE_3;
    case hall_all is
        when "000" => output_PWM <= "000000000000";
        when "001" => output_PWM <= "000010000110";
        when "010" => output_PWM <= "100001100000";
        when "011" => output_PWM <= "100000000110";
        when "100" => output_PWM <= "011000001000";
        when "101" => output_PWM <= "011010000000";
        when "110" => output_PWM <= "000001101000";
        when "111" => output_PWM <= "000000000000";
        when others =>output_PWM <= "000000000000";
    end case;
```

Figure 8: Generation of PWM-code from Hall values

3.6.3 Top Module

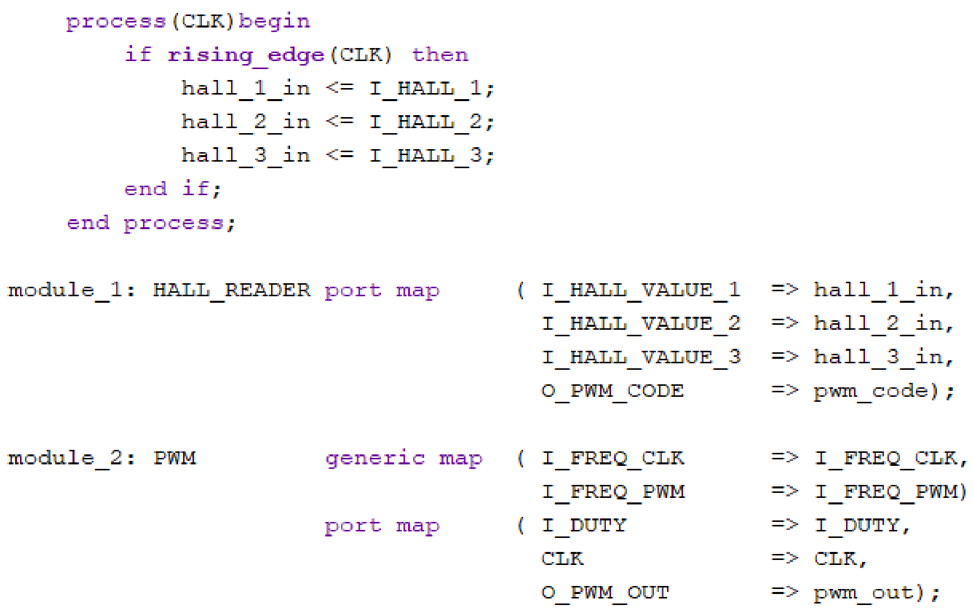
The ‘top’ module containing the PWM module and HALL_READER module takes the FPGA clock frequency, PWM frequency, three Hall sensor values, and clock as input and gives the duty value and generates six MOSFET signals as output.



```
entity top is
    generic   ( I_FREQ_CLK:  integer := A;
                I_FREQ_PWM:  integer := B);
    port      ( CLK:        in std_logic;
                I_HALL_1:   in std_logic;
                I_HALL_2:   in std_logic;
                I_HALL_3:   in std_logic;
                O_Q_0:     out std_logic;
                O_Q_1:     out std_logic;
                O_Q_2:     out std_logic;
                O_Q_3:     out std_logic;
                O_Q_4:     out std_logic;
                O_Q_5:     out std_logic;
                duty:      out integer);
end top;
```

Figure 9: Inputs and Outputs of TOP module

In Figure 9, I specified the FPGA and PWM frequencies as A and B. Because these values were requested to remain secret.



```
        process(CLK)begin
            if rising_edge(CLK) then
                hall_1_in <= I_HALL_1;
                hall_2_in <= I_HALL_2;
                hall_3_in <= I_HALL_3;
            end if;
        end process;

    module_1: HALL_READER port map      ( I_HALL_VALUE_1  => hall_1_in,
                                          I_HALL_VALUE_2  => hall_2_in,
                                          I_HALL_VALUE_3  => hall_3_in,
                                          O_PWM_CODE      => pwm_code);

    module_2: PWM         generic map   ( I_FREQ_CLK      => I_FREQ_CLK,
                                          I_FREQ_PWM      => I_FREQ_PWM)
                          port map      ( I_DUTY          => I_DUTY,
                                          CLK             => CLK,
                                          O_PWM_OUT       => pwm_out);
```

Figure 10: Obtaining hall values and input/output assignments of modules

The module presented in Figure 10 first gives the necessary values as input to the PWM module and obtains the PWM signal. Then, it takes the Hall values at each rising edge of the clock and gives it as an input to the HALL_READER module and obtains the generated PWM-code.

Then, with the obtained PWM-code, it makes appropriate assignments to the 6-bit PWMS signal created to represent the MOSFETs using a case statement.

```
case pwm_code is
    when "000000000000" => pwms <= "000000";
    when "000010000110" => pwms <= '0' & '0' & '1' & '0' & not pwm_out & pwm_out;
    when "100001100000" => pwms <= '1' & '0' & not pwm_out & pwm_out & '0' & '0';
    when "100000000110" => pwms <= '1' & '0' & '0' & '0' & not pwm_out & pwm_out;
    when "011000001000" => pwms <= not pwm_out & pwm_out & '0' & '0' & '1' & '0';
    when "011010000000" => pwms <= not pwm_out & pwm_out & '1' & '0' & '0' & '0';
    when "000001101000" => pwms <= '0' & '0' & not pwm_out & pwm_out & '1' & '0';
    when     others     => pwms <= "000000";
end case;
```

Figure 11: Obtaining 'pwms' values

Finally, it obtains the output signal that must go to the MOSFETs from the created 'pwms' signal.

```
o_Q_0 <= pwms(0);
o_Q_1 <= pwms(1);
o_Q_2 <= pwms(2);
o_Q_3 <= pwms(3);
o_Q_4 <= pwms(4);
o_Q_5 <= pwms(5);
```

Figure 12: Obtaining output signals for MOSFETs

3.6.4 Test

After the code was completed, I entered 3 different duty values and the results I obtained in the simulations I performed with Vivado are as follows.

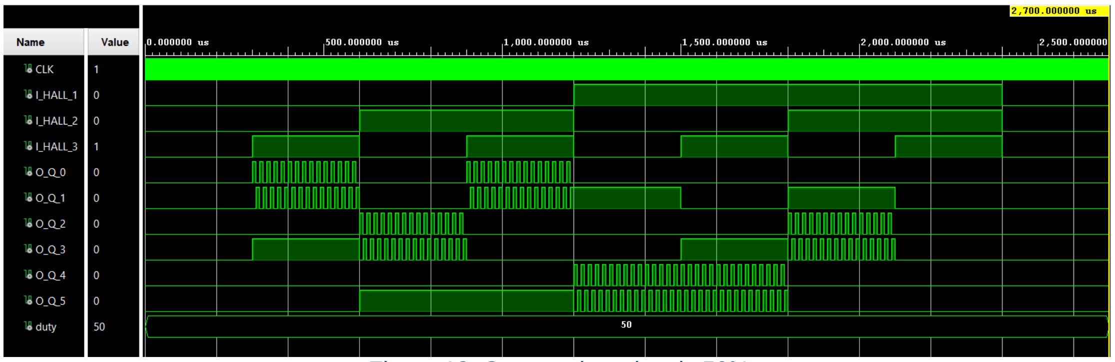


Figure 13: Output when duty is 50%

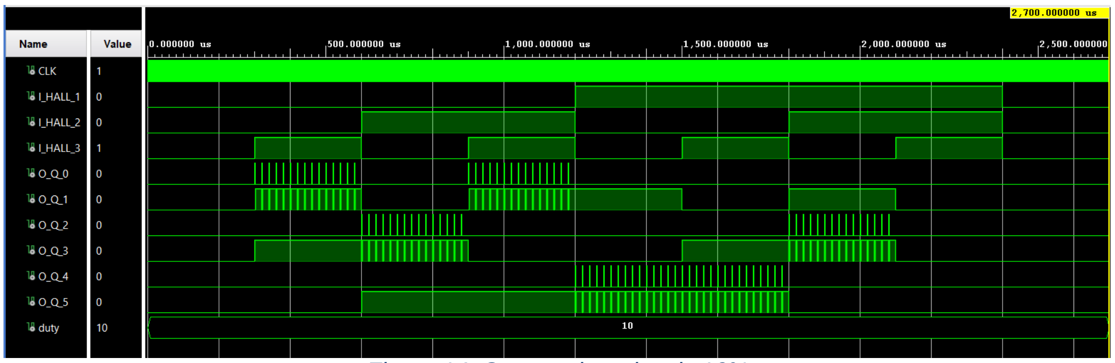


Figure 14: Output when duty is 10%

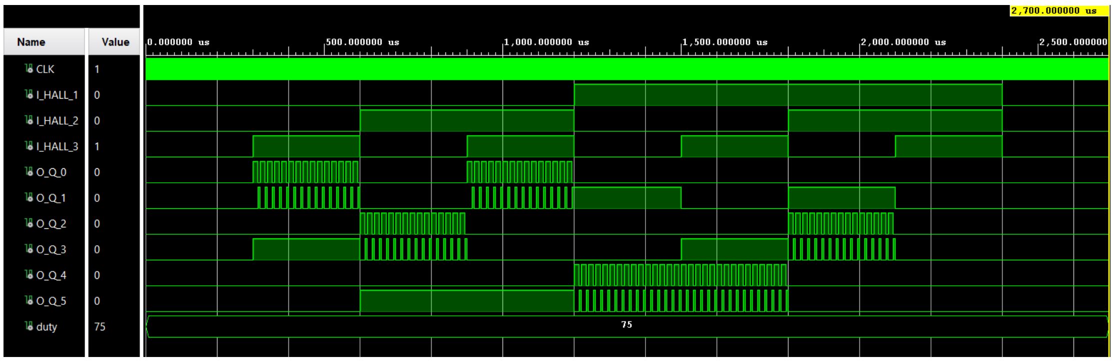


Figure 15: Output when duty is 75%

As can be seen in these results, the code fulfills its function. When different duty values were entered, the PWM signal that would control the MOSFETs was obtained by using the information from the Hall effect sensors.

After verifying that the code worked, I embedded the code into the demo FPGA in the laboratory environment, made the necessary connections and then tried to drive the BLDC motor. Although I cannot share more detailed information or images from this test section due to confidentiality reasons, the code worked successfully, I tried it and managed to drive the BLDC motor at different speeds according to 7 different duty values.

4 Performance and Outcomes

4.1 Solving Complex Engineering Problems

The project completed during the internship required me to merge my hardware and software knowledge. I had to understand the feedback logic, which I encountered for the first time, and use it in the project. Because I had to use the information coming from the sensors as input and give it back as output after the necessary operations and repeat this constantly. I created the code using feedback logic together with the structures I learned from the Digital Circuit Design course.

4.2 Recognizing Ethical and Professional Responsibilities

Since I was doing my internship in an environment with explosive and dangerous filters, I attended occupational health and safety training with all the interns on the first day. In this training, we were asked to understand the priority and seriousness of this issue very well and to pay attention to the rules explained throughout the internship.

In addition, with the information security awareness training we attended with all the interns, I was asked to know that confidentiality is a priority, as I expected from being in the defense industry, and that this issue is one of the most important things I should pay attention to during the internship.

While performing the duties assigned to me under these conditions, I understood the importance of worker safety and information confidentiality for companies and that paying attention to these issues in my future experiences was one of my primary responsibilities.

4.3 Making Informed Judgments

The first problem I faced was deciding how to start the project. In other words, it was not easy to figure out how to transfer the theoretical knowledge I gained from the sources given to me into the code or to think about where to start in the code. This process became even more difficult because I did not have internet access in the first days. To deal with this problem, I decided to put my ideas into practice with short codes in module form rather than writing my code from start to finish in a single piece. This decision was very appropriate because I had the chance to instantly verify the correctness of the code by using Vivado's simulation feature. Thus, I completed the TOP module, like building a Lego, testing, and ensuring I had attached the correct part.

4.4 Acquiring New Knowledge by Using Appropriate Learning Strategies

While I did not even know what the concept of driving was in the expression "driving a BLDC Motor" used in my job description at the beginning of my internship, I gained information on this subject by doing research from the resources given to me and the catalog of the engine I would use. In the first week of my internship, I just tried to research and obtain the necessary information and concepts to start the project. After the coding phase, I had to study and learn in detail the data sheet and pinout of the driver card and FPGA that I would use to implement the code in the laboratory. I had to understand this step in detail to avoid damaging the components.

4.5 Applying New Knowledge as Needed

I used the Vivado tool from the Digital Circuit Design course to apply the new knowledge. Working with a program I was familiar with allowed me to save time during adaptation and to put the information I learned into practice quickly. In other words, since I learned to use the Vivado program in great detail while doing the course labs and projects, it allowed me to do my job more easily during my internship.

4.6 Diversity, Equity, and Inclusion (DEI)

Roketsan gives priority to talented engineers, regardless of their gender, religion, or cultural origin, and provides an environment for them to collaborate on several projects. During my internship, I observed that engineers of different ages, genders, and experiences worked together effectively by exchanging information with each other, and everyone's opinion was considered regardless of their experience. Additionally, I got the chance to work in a really diverse workplace in terms of both personality and culture because the interns I worked with came from different universities and cities in Turkey.

5 Conclusions

My main goal during my summer internship at Roketsan was to drive a BLDC motor using FPGA. I worked on the project with VHDL via Vivado. After researching the concepts I would work on, I created and tested the VHDL code in modules. Then, I made the TOP module, which brings all the modules together, tested it, and confirmed that it all worked. Finally, I embedded the code I created into the FPGA in the test lab and confirmed that I could drive the BLDC motor at the desired speed.

Throughout my internship, I completed operating a system used in real life with the FPGA and VHDL knowledge I acquired in the 'EEE 102 Introduction to Digital Circuit Design' course. Additionally, thanks to my intern friends from other engineering departments, I gained experience in different fields by observing and asking questions about their tasks. Overall, this internship was an excellent opportunity for me to use my knowledge and gain new experiences for my career.

References

[1] “Hakkımızda”. https://www.roketsan.com.tr/tr/biz-kimiz/hakkimizda. [Accessed 24 Oct. 2023].

[2] “Ürünlerimiz”. https://www.roketsan.com.tr/tr/urunler. [Accessed 24 Oct. 2023].

[3] “Hizmetlerimiz”. https://www.roketsan.com.tr/tr/cozumlerimiz/hizmetlerimiz. [Accessed 24 Oct. 2023].

[4] “AC Motors and DC Motors”. https://www.powerelectric.com/motor-resources/motors101/ac-motors-vs-dc-motors. [Accessed: 24 Oct. 2023].

[5] “What is the Difference Between Brushed and Brushless DC Motors”. https://www.parvalux.com/what-is-the-difference-between-brushed-and-brushless-dc-motors. [Accessed: 24 Oct. 2023].

[6] “How Brushless DC Motor Works?”. https:// https://howtomechatronics.com/how-it-works/how-brushless-motor-and-esc-work/. [Accessed: 24 Oct. 2023].

[7] “AN885, Brushless DC (BLDC) Motor Fundamentals”. https://ww1.microchip.com/downloads/en/appnotes/00885a.pdf. [Accessed: 24 Oct. 2023].

[8] “Hall Effect sensor vs Phase waveform of BLDC motor”. https://www.researchgate.net/figure/Hall-Effect-sensor-vs-Phase-waveform-of-BLDC-motor_fig3_338582249. [Accessed: 24 Oct. 2023].

[9] “PWM (Darbe Genişlik Modülasyonu) – ARM Programlama”. https://kaanyagan.com/arm-programlama/pwm-darbe-genislik-modulasyonu-arm-programlama/. [Accessed: 24 Oct. 2023].

Appendices

Appendix A

VHDL code for “top” module

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity top is
   generic  ( I_FREQ_CLK:  integer := A;
            I_FREQ_PWM:  integer := B);
   port     ( CLK:       in std_logic;
            I_HALL_1:  in std_logic;
            I_HALL_2:  in std_logic;
            I_HALL_3:  in std_logic;
            O_Q_0:    out std_logic;
            O_Q_1:    out std_logic;
            O_Q_2:    out std_logic;
            O_Q_3:    out std_logic;
            O_Q_4:    out std_logic;
            O_Q_5:    out std_logic;
            duty:     out integer);
end top;

architecture Behavioral of top is

   signal I_DUTY: integer := 75;
   signal hall_1_in:  std_logic;
   signal hall_2_in:  std_logic;
   signal hall_3_in:  std_logic;
   signal pwm_out:    std_logic;
   signal pwm_code:   std_logic_vector(11 downto 0);
   signal pwms:       std_logic_vector(5  downto 0);
   signal i:          integer := 0;

   component HALL_READER    port    (  I_HALL_VALUE_1:         in std_logic;
                              I_HALL_VALUE_2:         in std_logic;
                              I_HALL_VALUE_3:         in std_logic;
                              O_PWM_CODE:            out std_logic_vector(11 downto 0));
   end component;

   component PWM            generic (  I_FREQ_CLK:   in integer;
                              I_FREQ_PWM:   in integer);
                      port    (  I_DUTY:       in integer;
                              CLK:          in std_logic;
                              O_PWM_OUT:   out std_logic);
   end component;

   begin

      process(CLK)begin
         if rising_edge(CLK) then
            hall_1_in <= I_HALL_1;
            hall_2_in <= I_HALL_2;
            hall_3_in <= I_HALL_3;
         end if;
      end process;

   module_1: HALL_READER port map     ( I_HALL_VALUE_1  => hall_1_in,
                              I_HALL_VALUE_2  => hall_2_in,
                              I_HALL_VALUE_3  => hall_3_in,
                              O_PWM_CODE      => pwm_code);
```

```
    module_2: PWM         generic map  ( I_FREQ_CLK      => I_FREQ_CLK,
                                         I_FREQ_PWM      => I_FREQ_PWM)
                          port map     ( I_DUTY          => I_DUTY,
                                         CLK             => CLK,
                                         O_PWM_OUT       => pwm_out);


    process(CLK)
    begin
      if I_DUTY < 10 then
        pwms <= "000000";
      else
        case pwm_code is
          when "000000000000" => pwms <= "000000";
          when "000010000110" => pwms <= '0' & '0' & '1' & '0' & not pwm_out & pwm_out;
          when "100001100000" => pwms <= '1' & '0' & not pwm_out & pwm_out & '0' & '0';
          when "100000000110" => pwms <= '1' & '0' & '0' & '0' & not pwm_out & pwm_out;
          when "011000001000" => pwms <= not pwm_out & pwm_out & '0' & '0' & '1' & '0';
          when "011010000000" => pwms <= not pwm_out & pwm_out & '1' & '0' & '0' & '0';
          when "000001101000" => pwms <= '0' & '0' & not pwm_out & pwm_out & '1' & '0';
          when    others      => pwms <= "000000";
        end case;
      end if;
    end process;

    O_Q_0 <= pwms(0);
    O_Q_1 <= pwms(1);
    O_Q_2 <= pwms(2);
    O_Q_3 <= pwms(3);
    O_Q_4 <= pwms(4);
    O_Q_5 <= pwms(5);
    duty <= I_DUTY;

end Behavioral;
```

Appendix B

VHDL code for “HALL_READER” module

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity HALL_READER is
    Port    ( I_HALL_VALUE_1:    in std_logic;
              I_HALL_VALUE_2:    in std_logic;
              I_HALL_VALUE_3:    in std_logic;
              O_PWM_CODE:    out std_logic_vector(11 downto 0));
end HALL_READER;

architecture Behavioral of HALL_READER is

signal hall_all:  std_logic_vector(2 downto 0);
signal output_PWM: std_logic_vector(11 downto 0);

begin
    hall_all <= I_HALL_VALUE_1 & I_HALL_VALUE_2 & I_HALL_VALUE_3;

    process(hall_all)
    begin
      case hall_all is
        when "000" => output_PWM <= "000000000000";
        when "001" => output_PWM <= "000010000110";
        when "010" => output_PWM <= "100001100000";
        when "011" => output_PWM <= "100000000110";
        when "100" => output_PWM <= "011000001000";
```

```
         when "101" => output_PWM <= "011010000000";
         when "110" => output_PWM <= "000001101000";
         when "111" => output_PWM <= "000000000000";
         when others =>output_PWM <= "000000000000";
      end case;
   end process;
   O_PWM_CODE <= output_PWM;
end Behavioral;
```

Appendix C

VHDL code for “PWM” module

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity PWM is
   generic  (  I_FREQ_CLK:   in integer;
               I_FREQ_PWM:   in integer);
   port     (  I_DUTY:       in integer;
               CLK:          in std_logic;
               O_PWM_OUT:    out std_logic);
end PWM;

architecture Behavioral of PWM is

   signal limit, hightime:   integer;
   signal timer:    integer := 0;

begin
   limit <= I_FREQ_CLK/I_FREQ_PWM;
   hightime <= (limit/100)*I_DUTY;

   process(CLK)
   begin
      if rising_edge(CLK) then
         if timer = limit-1 then
            timer <= 0;
         elsif I_DUTY < 10 then
            O_PWM_OUT <= '0';
            timer <= timer +1;
         elsif I_DUTY > 90 then
            O_PWM_OUT <= '1';
            timer <= timer +1;
         elsif timer < hightime then
            O_PWM_OUT <= '1';
            timer <= timer +1;
         else
            O_PWM_OUT <= '0';
            timer <= timer +1;
         end if;
      end if;
   end process;
end Behavioral;
```

Appendix D

VHDL code for “TEST_TOP” module

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
entity topxxx is
end topxxx;

architecture Behavioral of topxxx is

component top is
    Port    (  CLK:        in std_logic;
               I_HALL_1:  in std_logic;
               I_HALL_2:  in std_logic;
               I_HALL_3:  in std_logic;
               O_Q_0:    out std_logic;
               O_Q_1:    out std_logic;
               O_Q_2:    out std_logic;
               O_Q_3:    out std_logic;
               O_Q_4:    out std_logic;
               O_Q_5:    out std_logic;
               duty:     out integer);

end component;

signal CLK, I_HALL_1, I_HALL_2, I_HALL_3: std_logic := '0';
signal O_Q_0, O_Q_1, O_Q_2, O_Q_3, O_Q_4, O_Q_5:    std_logic;
signal duty: integer;

begin
f1: top port map (CLK, I_HALL_1, I_HALL_2, I_HALL_3, O_Q_0, O_Q_1, O_Q_2, O_Q_3, O_Q_4,
O_Q_5, duty);

clock: process
begin
    CLK <= '1';
    wait for 2.5 ns;
    CLK <= '0';
    wait for 2.5 ns;
end process;



--p1: process
--begin
--    wait for 140us;
--    reset <= '1';
--    wait for 10 us;
--    reset <= '0';
--    wait for 10000 us;
--end process;

p2: process
begin
    I_HALL_1 <= '0';
    I_HALL_2 <= '0';
    I_HALL_3 <= '0';
    wait for 300 us;
    I_HALL_1 <= '0';
    I_HALL_2 <= '0';
    I_HALL_3 <= '1';
    wait for 300 us;
    I_HALL_1 <= '0';
    I_HALL_2 <= '1';
    I_HALL_3 <= '0';
    wait for 300 us;
    I_HALL_1 <= '0';
    I_HALL_2 <= '1';
    I_HALL_3 <= '1';
    wait for 300 us;
    I_HALL_1 <= '1';
    I_HALL_2 <= '0';
    I_HALL_3 <= '0';
    wait for 300 us;
    I_HALL_1 <= '1';
    I_HALL_2 <= '0';
    I_HALL_3 <= '1';
    wait for 300 us;
    I_HALL_1 <= '1';
    I_HALL_2 <= '1';
    I_HALL_3 <= '0';
    wait for 300 us;
    I_HALL_1 <= '1';
    I_HALL_2 <= '1';
    I_HALL_3 <= '1';
    wait for 300 us;
end process;


end Behavioral;
```

Self-Checklist for Your Report

Please check the items here before submitting your report. This signed checklist should be the final page of your report.

- ☐ Did you provide detailed information about the work you did?
- ☐ Is supervisor information included?
- ☐ Did you use the Report Template to prepare your report, so that it has a cover page, has all sections and subsections specified in the Table of Contents, and uses the required section names?
- ☐ Did you follow the style guidelines?
- ☐ Does you report look professionally written?
- ☐ Does your report include all necessary References, and proper citations to them in the body?
- ☐ Did you remove all explanations from the Report Template, which are marked with yellow color? Did you modify all text marked with green according to your case?

Signature: ________________________

While writing your summer internship reports, you should follow the rules of ethical writing. You can find an extensive guide on ethical writing at:

https://ori.hhs.gov/content/avoiding-plagiarism-self-plagiarism-and-other-questionable-writing-practices-guide-ethical-writing