FPGA Based model of 3-Phase Induction Motor using VHDL

Nomaan Sayed MSc MIET
Department of Computing, Electronics and Mathematics
Coventry University
Coventry, United Kingdom
sayednomaan18@outlook.com

***Abstract*—This paper addresses the increasing demand for advanced electric motor testing, driven by the surge in Electric Vehicles (EVs) and the need for sustainable transportation solutions. Conventional testing methods for electric motors are often costly and time-intensive. To overcome these limitations, this research introduces a novel, reprogrammable FPGA-based model of a 3-phase induction motor, focusing on real-time motor testing using the Hardware-in-the-Loop (HIL) technique. The induction motor model is coded in Very High-Level Hardware Descriptive Language (VHDL) using AMD Xilinx Vivado software. This approach improves cost-effectiveness, efficiency, and performance in hardware testing. The methodology involves the development of lookup tables, generated using Python, to represent the motor's three-phase input voltage, stator magnetomotive force (MMF), and rotor characteristics (torque, speed, current). These components are then implemented as synthesisable VHDL entities on an FPGA. Power analysis and simulation results for each module are presented. This work establishes a foundational framework for simulating induction motors on FPGAs, offering a significant step towards more advanced and cost-efficient testing environments, particularly for EV motor design and development**

***Index Terms*—Hardware-in-the-Loop, FPGA-in-the-loop, Induction Motor, VHDL, Python, VS Code, TerosHDL and AMD Xilinx Vivado.**

I. Introduction

The demand for Electric Vehicles (EVs) has surged in recent years due to increasing environmental concerns, technological advancements, and the growing need for sustainable transportation. According to the European Union, the transport sector contributes approximately 28% of total carbon dioxide emissions [1]. In the process, electric vehicles have emerged as a viable solution to mitigate these emissions while addressing the ongoing energy crisis.

Despite being considered an emerging technology today, electric vehicles have a historical legacy dating back to the late 19th century, with early models developed by pioneers such as Thomas Parker and Ferdinand Porsche. During the 1920s, Electric vehicles constituted nearly 28% of the United States automotive market [2]. However, the rapid evolution and affordability of internal combustion engine vehicles led to a significant decline in the adoption of electric vehicles.

In recent decades, the electric vehicle market has experienced a resurgence, driven by progress in battery technology, increased environmental awareness, and supportive government policies. Contemporary electric vehicles are generally classified as battery electric vehicles (BEVs), hybrid electric vehicles (HEVs), and fuel cell electric vehicles (FCEVs), each comprising key components such as an electric motor, battery, control unit and charger [3]. With the accelerated adoption of electric vehicles, the demand for comprehensive testing of diverse electric motor technologies, each exhibiting distinct characteristics in terms of efficiency, torque output, and scalability has increased significantly.

Conventional testing methods that rely on physical prototypes are costly and time-intensive. To overcome these limitations, the integration of Hardware-in-the-Loop (HIL) simulation and Field-Programmable Gate Array (FPGA)-based modeling presents a robust alternative. HIL facilitates real-time testing of motor control systems without necessitating physical hardware, while FPGAs offer high-speed, reconfigurable emulation of different motor types. This integrated approach significantly reduces R&D expenditure, accelerates development cycles, and provides a scalable framework to optimise electric motor performance in EV applications.

II. Electrical Machines.

Electric Motor is the device that converts electrical energy into mechanical energy by using electromagnetic phenomena. Most electrical machines develop their mechanical torque by interacting with current carrying conductors in a direction at right angles to a magnetic field [4]. The first motor was discovered in 1740, which was Electrostatic device which worked by using Electrostatic force, by a Scottish Monk Andrew Gordon and an American experimenter Benjamin Franklin [5]. Electric drive technologies have rapidly evolved in last few decades. The three main machines are DC motor, Induction and Synchronous machine. With intense research, other variations of electric machines now existed like Brushless motor, Permanent magnet machines etc. With advancements in micro-controllers, different motor control technologies are available depending on the user's need [6].

A. DC Motor.

DC (Direct current) motor has been widely used in many applications, electric vehicles (EV's), robotics, steel rolling

mills, etc. The operation is based on interaction between magnetic field produced by stationary and rotating components. The electrical current flows through the armature winding that creates magnetic field, then interacts with the magnetic field produced by stator. This interaction generate mechanical torque that causes the rotor to rotate. They have a precise speed control. It works on DC power supply. They have a high starting torque, which makes them suitable for the application which required high starting torque. They are high maintenance because of more parts. DC motor provides excellent speed control for acceleration and de-acceleration. There are several controllers like, PI (Proportional Integral), PID (Proportional Integral Derivative), FLC (Fuzzy Logic Controller). The excitation state is responsible to energise the field winding which creates the magnetic field necessary for motor operation. There are three such state, Separately excited, Series excited and shunt excited [7].

B. Permanent Magnet DC Motor.

It is a type of DC motor that requires a permanent magnet to create the magnetic field required for the operation of the DC motor. They are commonly used as a starter motor in automobile, windshield wiper etc. The working principle of the PMDC motor is similar to that of the general DC motor, that is, when a current-carrying conductor enters the magnet field, the conductor experiences the mechanical force and the direction of this force is governed by Fleming's left-hand rule. In PMDC, the armature is placed inside the magnetic field of a permanent magnet, and the armature rotates in the direction of the force generated. Each conductor of the armature experiences a force and the integration of all these forces produces a torque that tends to rotate the armature [8].

C. Brushless DC Motor.

The brushless DC motor (BLDC) is a major advancement in electric motor technology, offering improved efficiency and reliability by eliminating brushes and commutators. It uses electronic commutation, reducing maintenance, and ensuring smoother operation. The rotor's equipped with permanent magnets that interact with the stator's magnetic field to generate rotational motion, making them highly efficient compared to traditional brushed motors. Similarly, Permanent Magnet Synchronous Motor (PMSM) utilise permanent magnets embedded in the rotor, eliminating the need for excitation current. When a three-phase power supply is applied, the stator generates a revolving magnetic field that synchronises with the rotor's magnetic field, enabling precise control of speed and torque. PMSMs are known for their high efficiency, reduced power consumption, and ability to deliver high torque across a wide range of operating speeds. Variable Frequency Drives (VFDs) allow smooth speed control by adjusting the timing and amplitude of the stator's magnetic field, optimising performance for specific applications. BLDC and PMSM motors are critical advancements in electric motor technology, offering superior efficiency and reliability. Their applications span electric vehicles and industrial automation, driving innovation and sustainability by providing precise control, reduced maintenance, and improved overall performance [9].

D. Induction Motor

The induction motor, often referred to as the asynchronous motor, is widely used in electric vehicles (EVs) and hybrid electric vehicles (HEVs) due to its reliable performance and efficient energy conversion. It operates on the principle of electromagnetic induction, where electrical energy is converted into mechanical energy without any direct electrical connection to the rotor. Designed by Nikola Tesla, the induction motor is capable of quickly reaching the desired speed while maintaining energy efficiency and ensuring safe current levels. Induction motors are typically classified into single-phase and three-phase types. Single-phase motors are more common in residential applications, and three-phase motors are predominantly used in the commercial and industrial sectors. The motor consists of two main components: the stator and the rotor. The stator, which contains coiled windings, generates a magnetic field when alternating current (AC) is applied. This magnetic field induces motion in the rotor, which is usually made from laminated iron cores to reduce eddy current losses. In three-phase motors, the stator windings are configured in star or delta arrangements, and a rotating magnetic field (RMF) is generated by a 120-degree phase-shifted three-phase AC voltage. This interaction between the magnetic field of a stator and the rotor creates torque, causing the rotor to rotate. Induction motors are also classified by their rotor design into squirrel cage and wound rotor types. The squirrel cage rotor is the most commonly used and is simple, robust, and low maintenance. The wound rotor, on the other hand, allows for external control of speed and torque. The air gap between the stator and rotor impacts motor performance, with a smaller gap enhancing efficiency but increasing friction, while a larger gap reduces friction but weakens the magnetic field. In general, the precision, durability, and efficiency of the induction motor make it an essential component in electric vehicles and HEVs [10].

III. Electric Motor Selection.

As per the above review, The Induction motor is widely used electrical machines in the industry due to their robustness, reliability, and simplicity. They operate on the principle of electromagnetic induction, where a rotating magnetic field in the stator induces currents in the rotor, generating torque and causing the rotor to rotate. The stator has three phase windings, typically supplied with three-phase AC voltage. When AC voltage is applied to the stator windings, it produces alternating currents that vary sinusoidally over time. These alternating currents create a rotating magnetic field around each stator winding, which combines to form a rotating magnetic field in the air gap between the stator and rotor [11]. Induction motors operate by creating a rotating magnetic field in the stator, which induces currents in the rotor, generating torque that drives the rotor. They are commonly used in industrial

applications like pumps, fans, compressors, and machine tools, due to their simplicity, low maintenance, and rugged construction. However, induction motors have limitations in speed control and may suffer from reduced efficiency at low speeds. DC motors, which use direct current (DC), are available in brushed and brushless configurations. Brushed DC motors rely on brushes and a commutator to reverse current direction in the rotor, while brushless DC motors use electronic commutation for better reliability. DC motors provide precise speed control and high starting torque, making them suitable for applications requiring variable speed operation, such as robotics and electric vehicles. Brushed DC motors require periodic brush maintenance, whereas brushless DC motors offer higher efficiency but demand more complex control systems. Synchronous motors rotate at a constant speed synchronized with the AC power supply frequency, ensuring precise speed control [12]. Synchronous motors, including synchronous reluctance motors and permanent magnet synchronous motors (PMSM), offer high efficiency, power factor correction, and precise speed control, making them ideal for applications that require stable operation. Synchronous reluctance motors utilize a rotor with salient poles, known for their simplicity and robustness. PMSMs, which incorporate permanent magnets in the rotor, provide higher efficiency and power density. These motors are commonly used in industrial pumps, compressors, fans, and HVAC systems. However, synchronous motors tend to be more expensive than induction motors and may require external means of starting, limiting their applicability in certain situations. Induction motors, on the other hand, are known for their simplicity, reliability, and low maintenance. While they lack the precise speed control capabilities of synchronous or DC motors, they are widely used in industrial applications due to their rugged construction and cost-effectiveness. Factors like efficiency, torque characteristics, and maintenance requirements determine the selection between these motor types [13].

IV. Hardware Verification and Integration

The evolution of hardware testing has paralleled advancements in electrical machine development, transitioning from traditional methods to modern simulation-driven approaches. Historically, testing relied on manufacturing physical prototypes, such as induction machines, which involved substantial costs and the use of actual materials. Failures or design modifications required significant time and resources, making the process inefficient and labour-intensive. Modern hardware testing technology mitigates these challenges through advanced simulation and virtual testing. This approach enables engineers to simulate and analyse hardware designs in a virtual environment, facilitating rapid iteration and refinement before physical implementation. By identifying and addressing potential issues during the design phase, engineers can reduce errors, delays, and costs associated with physical prototyping. Hardware testing technology enhances flexibility in the development process. Unlike traditional methods that require extensive rework for design changes, virtual simulations allow for quick and seamless adjustments. Engineers can efficiently explore alternative configurations, iterate on designs, and optimize performance parameters without the constraints of physical prototypes. This innovation also enables engineers to test a broader range of design possibilities and real-world scenarios. Advanced simulation tools and modelling techniques facilitate assessments of performance under varying operational conditions, such as different loads and environmental factors. By analysing the impact of design choices on system behaviour, engineers can optimize designs for maximum efficiency and reliability. Hardware testing technology empowers engineers to evaluate diverse configurations, streamline the design process, and improve product quality. With comprehensive simulation tools, they can explore real-world conditions and optimize designs to achieve superior performance, reducing development time and cost significantly while ensuring enhanced reliability and system efficiency [14].

A. Hardware-in-the-Loop (HIL).

Hardware-in-the-loop (HIL) simulation is a vital technique which is utilised in the development of complex systems across various industries, including aerospace, maritime, and electrical engineering. This method involves the integration of real hardware components, such as controllers, sensors, and actuators, with a computer-based simulated environment. The simulated environment replicates real-world conditions, allowing engineers to visualise and analyse system behaviour in real-time. In HIL simulation, the actual hardware under test is connected to the simulation environment via interfaces such as analogue and digital I/O or other protocols. This connection forms a closed loop, enabling seamless interaction between the hardware components and the simulated environment. By incorporating real hardware into the simulation, engineers can assess system performance under realistic operating conditions and validate the functionality of complex systems [15]. The significance of HIL simulation lies in its ability to mitigate risks associated with testing real hardware in a live environment. Given the high stakes involved, particularly in industries where hardware components are valued at millions or even billions of dollars, the potential for errors during testing poses significant challenges. HIL simulation offers a controlled and safe environment for testing, allowing engineers to identify and rectify potential issues before deploying hardware in real-world applications. By applying HIL simulation, engineers can optimise system performance, ensure reliability, and reduce development time and costs. This approach enhance efficiency and minimises the likelihood of costly errors or failures during system deployment. Ultimately, HIL simulation plays a crucial role in ensuring the success of complex engineering projects by enabling thorough testing and validation of hardware systems in a controlled environment [16].

B. FPGA-in-the-Loop (FIL).

Like Hardware-in-the-loop (HIL) testing, Field-Programmable Gate Array-in-the-loop (FPGA-in-the-loop) testing involves integrating Field-Programmable Gate Arrays (FPGAs) into the testing and validation process. FPGAs

are semiconductor devices that can be re-programmed after implementation, offering flexibility and versatility in hardware design. In FPGA-in-the-loop testing, a simulation environment is established, comprising mathematical models, software, and other components. The hardware functions, such as controllers and signal processing, are implemented on the FPGA within this simulation environment. This approach is particularly advantageous in applications requiring real-time performance, high-speed data processing, or specialised logic. The FPGA-in-the-loop concept enables engineers to rapidly iterate FPGA designs, validate their functionality, and identify any issues early in the development process. By simulating the behaviour of the FPGA within a controlled environment, engineers can assess its performance under various conditions and refine the design accordingly. By applying simulation environments and FPGA technology, engineers can increase performance, reliability, and streamline the development process [17].

C. Controller-in-the-Loop (CIL).

Control-in-the-loop (CIL) validation is a technique used to assess the functionality and performance of control systems by integrating real controller hardware, such as microcontrollers or PLCs, into a simulated environment. This approach involves setting up a computer-based simulation that emulates the behaviour of the system being controlled. The controller is connected to the simulation, receiving input from the simulated system and sending control signals back to it, mimicking real-world interaction. The primary focus of CIL validation is to evaluate the performance of the controller itself, unlike hardware-in-the-loop (HIL) testing, which emphasises testing sensors, actuators, and electronic components. Through this process, engineers can assess how well the controller responds to various conditions and scenarios, providing valuable insights into its behaviour and effectiveness in controlling the system [18].

V. Experiment

In the traditional old way, where the entire Induction Machine (selected for this research) is simulated on the simulation software like MATLAB/ Simulink. The controller itself is designed using the simulation software which works on certain calculations that doesn't give the real-time output, and no other parallel processing is possible. This method does not replicate the real hardware where the real Induction machine is implemented, and it is controlled by real controller which gives the output in oscilloscope or its application. Due to lower efficiency, lower response time, lower accuracy, it was then replaced by the hardware-in-the-loop, FPGA-in-the-loop, etc concepts. The actual machine still simulates on the software. This problem is addressed with the help of FPGA. In this research, the Induction Machine is coded with the help of VHDL (Very High-Speed Integrated Circuit Hardware Descriptive language) and then it is implemented on FPGA board. This machine can be programmed as per the user requirement.

A. Why FPGA?

An FPGA board, also known as an FPGA development or evaluation board, is a hardware platform that integrates an FPGA (Field-Programmable Gate Array) alongside various essential components like I/O ports, memory, and power management systems. The FPGA is the core of the board, a reprogrammable semiconductor device that can be configured to perform a wide range of tasks. The board typically includes various interfaces for external devices such as sensors, actuators, and displays, including GPIO pins, serial interfaces (UART, SPI, I2C), Ethernet, and USB ports. In addition, FPGA boards contain different types of memory, including on-chip memory (block RAM), external memory (DDR SDRAM), and non-volatile memory (Flash). They also include clocking resources like crystal oscillators, clock generators, and phase-locked loops (PLLs) to provide stable clock signals for time-sensitive applications. Power management features, such as voltage regulators and power monitoring, ensure clean and stable power. FPGAs are increasingly used in data centres, with FPGA-based Infrastructure as a Service (IaaS) being a growing trend, offering faster processing and superior power efficiency compared to traditional GPUs. Companies like Amazon and Intel have already adopted FPGAs in their data centres. This flexibility and efficiency make FPGAs an ideal choice for designing induction machines in electronic hardware, where customizability, speed, and power management are critical [19].

B. Why not DSP Board?

The DSP board or Digital signal processing board specifically designed for processing digital signals in real-time. The centre of the DSP board is the DSP chip which is a specialised microprocessor optimised for processing digital signals efficiently. DSP board typically involves using high-level programming languages like C/C++ or specialised development algorithms provided by DSP manufacturer. DSP board has features like FPGA, but the main difference lie in their flexibility and customisation. DSP is implemented using programming language while the FPGA on the other hand when implemented using VHDL or Verilog it will work like a custom build processor, makes it much faster and efficient than DSP board. With all the pros and cons, FPGA board is selected. DSP boards are more suitable for acoustical, audio engineering where the work is directly linked with signals [20].

C. VHDL over Verilog

VHDL (Very High-Speed Integrated Circuit Hardware Description Language) is an HDL language used primarily in electronic design automation. VHDL designs are composed of entities, that is the building blocks of digital circuits. Each entity describes a component such as logic gates, register or subsystem. They have input and output ports. Each entity in VHDL is associated with one or more architecture, that describe the behaviour and functionality of the entity. This can be described using concurrent or sequential statement. VHDL supports various data types like integers, float-point

types, pre-defined libraries and packages for processing and other common tasks. VHDL allows the designer to describe the behaviour of the digital circuits using techniques such as process statements, signal assignments, conditional statements, and loops. It supports structural modelling, which allows engineers to specify the hierarchical structure of a circuit by instantiating and connecting lower-level components. VHDL designs can also be synthesised into hardware configuration for implementation on FPGA (Field Programmable Gate Array) or ASIC (Application-Specific Integrated Circuit). Verilog syntax is more like C-language, it is often bottom-up approach where the engineers first start by specifying behaviour of individual component. Both the language different syntax, design philosophies and communities users. In this research, we had selected VHDL language [21].

D. Three Phase Input Voltage

Fig. 1. Basic Block of Induction Motor.

The three-phase input voltage is a sinusoidal voltage signal. The reference voltage is taken as 230 voltage. VHDL code works with binary data but can be feed with integer data to get the maximum amplitude. The logic is built by using the look up table. The look up table works on the logic that a sine wave is generated between 0 to 2π. The logic is built to generate a sinewave. If the voltage is taken as 230, then the maximum amplitude of the look up table will be 230, the number of bits in the sine wave will decide how visual the sine wave would be. It is generally taken as 256 bits to generate a full sine wave without any distortion. The look up table is generated using the Python language by using the NumPy library.

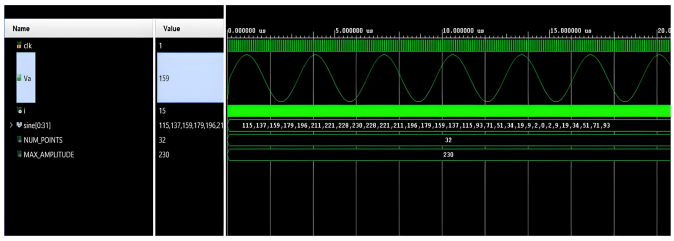

Fig. 2. Input (V) Sinewave.

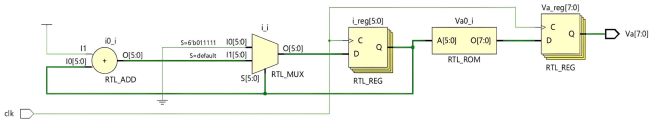

Fig. 3. RTL diagram of input voltage

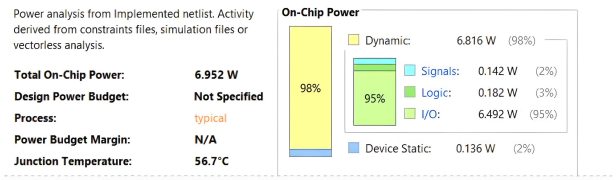

Fig. 4. Power analysis of input voltage.

E. Stator

The stator parameters are calculated manually using the motor characteristics:

$$V_L = \sqrt{3} \times 230 = 398.4\,\text{V}$$
$$\text{Number of poles} = 4 \Rightarrow \text{Number of pole pairs} = 2$$
$$f = 50\,\text{Hz}, \quad I_L = 2.33\,\text{A}$$
$$\text{Nominal Torque} = 3.69\,\text{Nm}$$
$$N_s = 3000\,\text{rpm}, \quad pf = 0.85$$
$$Z_s = 0.001\,201\,\Omega, \quad L_s = 0.012\,41\,\text{H}$$
$$N = 500\ \text{turns}$$

Assuming all mechanical parts and friction losses are negligible, the magnetomotive force (MMF) is calculated as:

$$\text{MMF} = NI = 500 \times 2.33 = 1165\,\text{A turns}$$

Assuming Y Connection The stator will be coded by using these calculations, the logic is build using python which will give a phase shift of 120 degree and -120 degree. The lookup table will then be created with a phase shift, the lookup table is stored inside the memory, which is called by giving a clock pulse. The logic is built then run the simulation by giving the clock and change the option on waveform from digital to analogue to see the sine wave.

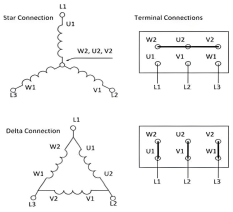

Fig. 5. Y and Delta connection.

The stator will give the MMF, which will be the input to the rotor. The stator will have a three- phase shift output which will create the rotating magnetic field. The result of the MMF is then coded again using python language with a phase shift of 120 degrees.

The output can be seen as three phase, each having a phase shift of 120 degrees sine wave, which is the MMF.

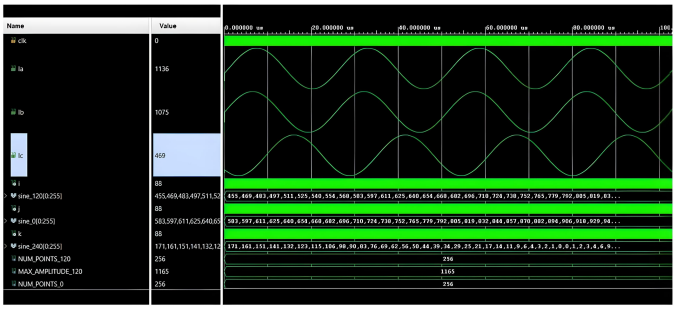

Fig. 6. Stator SineWave output.

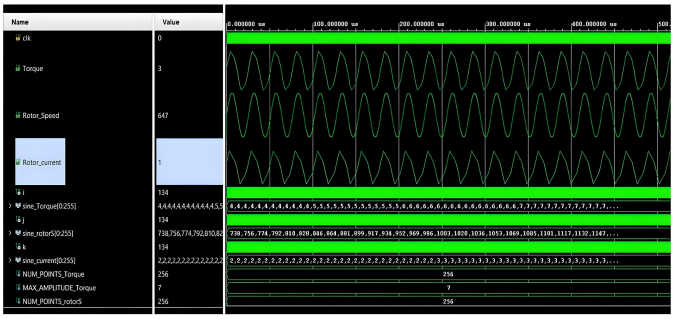

Fig. 9. Rotor current, Rotor speed, Rotor torque output.

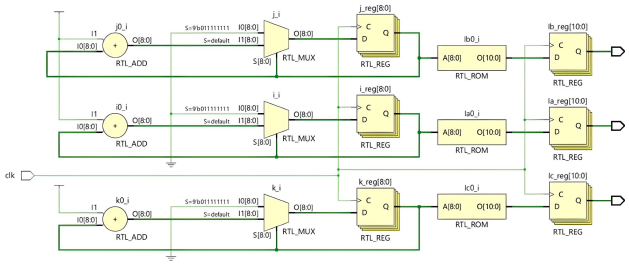

Fig. 7. RTL diagram of Stator.

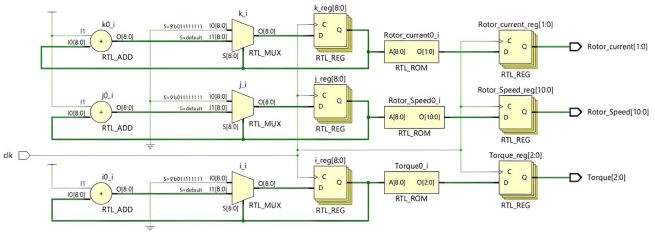

Fig. 10. RTL diagram of Rotor.

F. Rotor

Assuming all mechanical components and friction losses are equal to zero:

$$P_{\text{in}} = \sqrt{3} \times 230 \times 2.33 = 928.2\,\text{W}$$
$$P_{\text{mech}} = P_{\text{fr}} + P_{\text{em}} = 928.2\,\text{W} \quad (\text{since } P_{\text{fr}} = 0)$$

Mechanical torque is given by:

$$T_{\text{em}} = \frac{P_{\text{mech}}}{\omega_r} = \frac{60 \cdot P_{\text{em}}}{2\pi N_r}$$
$$= \frac{60 \times 928.2}{2\pi \times 1475} = 6.009\,\text{Nm}$$

Rotor current:

$$I_r = 191.507\,\text{mA}$$

Thus, we calculated rotor current, rotor torque and rotor speed (which was given). The output waveform gives the idea of Motor operating characteristic. The calculations are thus feed into the logic of look-up table which is created with the help of python. The look-up table is used to produce an output sine wave. The output is generated with the help of simulation option inside vivado, the rtl design which represents the circuit in terms of register and logical operations between them. Power consumption is helpful in designing as the user will keep an eye on the chip power consumption and design as per the requirement.

VI. How to take output?

There are several methods to observe the output. First, using Xilinx Vivado, we ran the simulation and triggered the output waveform using the clock edge. Second, Chipscope, a debugging and virtual oscilloscope software, can be used by connecting the external FPGA board via a JTAG cable to observe the output [22]. Third, after generating the bitstream and programming the device, connect the FPGA board to an

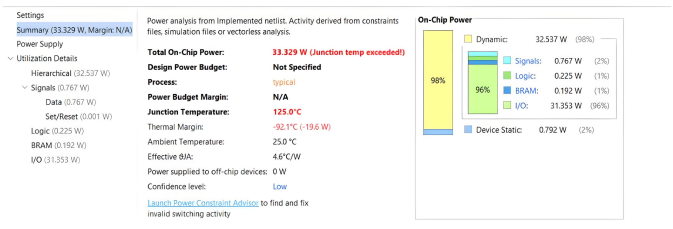

Fig. 8. Power analysis of Stator.

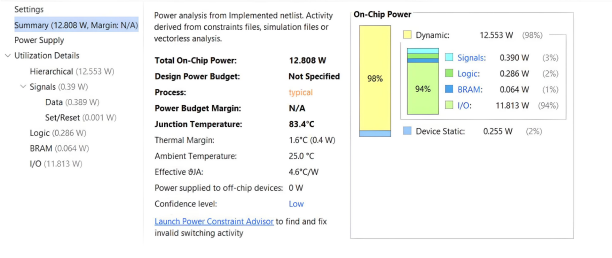

Fig. 11. Power analysis of Rotor.

oscilloscope using a USB cable, ensuring the use of a PMOD interface for external components such as I/O, VGA, or USB

VII. Conclusion

This research establishes a strong foundational framework for simulating an Induction Motor on FPGA. Future work will aim to replicate the behaviour of an actual induction motor more precisely. This approach has the potential to significantly enhance electric vehicle (EV) motor design, offering a more advanced and cost-efficient testing environment.

In future development, the system will incorporate real-time analogue input from a three-phase AC voltage source. This analogue signal will be converted into digital form using an Analogue-to-Digital Converter (ADC), which will serve as input to the digital signal processing (DSP) algorithm. This algorithm can be implemented either using C language within Xilinx Vivado or by interfacing with a dedicated DSP board. Communication with the FPGA board will be managed via interfaces such as JTAG or UART.

If a DSP board is not used, the DSP logic must be implemented directly in the FPGA by designing a signal processing algorithm to generate digital integer values. These values will be stored in ROM and processed using divider and multiplier circuits. The divider logic will be based on a 32x16-bit architecture, incorporating full adders, multiplexers, comparators, and subtractors. Similarly, a 16x16-bit multiplier will be developed using full adders, half adders, multiplexers, demultiplexers, shifters, and appropriate control logic. These logic blocks will execute inside loop structures, generating MagnetoMotive Force (MMF) as output to the rotor.

The rotor will follow a similar process, converting the input into a lookup table output, which can then be visualised on an oscilloscope. If a lookup table is implemented, there will be no need for a digital-to-analogue converter (DAC).

The controller for the system can be implemented in two ways. The first involves using an external microcontroller (e.g. Arduino or Raspberry Pi) connected to the FPGA via an appropriate interface. In this configuration, a Voltage Source Converter (VSC) must be implemented to handle switching characteristics, as seen in control methods like Direct Torque Control (DTC). The second method uses the FPGA as a controller and processor, eliminating the need for external microcontrollers. The available I/O ports on the FPGA board can be assigned to simulate the controller's switching behaviour. This research can increase the potential of motor technology reducing faults and errors significantly. This can further be executed with machine learning algorithm to detect early fault in the FPGA based motor which can save time and cost.

References

[1] V.T.-S. ,. P. G. ,. F. J. M. ,. J. M. M.-B. Julio A. Sanguesa, "A review on electric vehicles: Technologies and challenges," 15 March 2021.https://www.mdpi.com/2624-6511/4/1/22.

[2] M. Guarnieri, "Looking back to electric cars," 2012. https://www.researchgate.net/publication/261118933.

[3] Z. L. X. W. C. L. Xiaoli Sun, "Technology Development of Electric Vehicles: A Review," 23 December 2019. https://www.mdpi.com/1996-1073/13/1/90#B12-energies-13-00090.

[4] G. R. Slemom, "Electric motor," 2024. https://www.britannica.com/technology/electric-motor.

[5] Wikipedia, "The history of the electric motor," https://en.wikipedia.org/wiki/Electric_motor#cite_ref-3.

[6] P. C. SEN., "Electric motor drives and control-past, present, and future.," 1990. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=103462.

[7] Nassim Messaadi, Abdelkader Amroun, "Speed Control of DC Motor Using Fuzzy PID Controller," 2021. https://www.researchgate.net/publication/353863156.

[8] S. M. H. R. Hafiz M. Usman, " Permanent magnet DC motor parameters estimation via universal adaptive stabilization," September 2019. https://rb.gy/d8gbo5

[9] A. &. D. B. K. R. P. Ms. Loganayaki, "Permanent magnet synchronous motor for electric vehicle applications," 2019. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8728442.

[10] Y. M. A. B. S. K. V. B. &. Z. R. Valerii Tytiuk, "Exploring the MMF of a three-phase induction motor with twelve-zone stator windings.," 2021.https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9598786.

[11] Steve Drzymala, & Mark Fanslow, "Considerations in the selection and application of AC and DC motors for cement plants," 2009. https://ieeexplore.ieee.org/document/5116168.

[12] A. M. A. D. &. M. Y. Kanber Sedef, " A comparative study of the performance of DC permanent magnet and AC induction motors in urban electric cars.," 2012. https://ieeexplore.ieee.org/document/6294077.

[13] M. P. &. H. K. Merve Yildirim, "A survey on comparison of electric motor types and drives used for electric vehicles," 2014. https://ieeexplore.ieee.org/document/6980715.

[14] I. S. 1.-2. (. o. I. S. 1012-2004)., "IEEE standard for system, software, and hardware verification and validation," 2016. https://ieeexplore.ieee.org/document/8055462.

[15] Y. C. S. Z. &. N. Z. Shitao Chen, "A novel integrated simulation and testing platform for self-driving cars with hardware in the loop," 28 May 2019. https://ieeexplore.ieee.org/abstract/document/8723561

[16] M. T. A. H. Franc Mihalič, "Hardware-in-the- loop simulations: A historical overview of engineering challenges," 8 August 2022. https://www.mdpi.com/2079-9292/11/15/2462

[17] B. B. ,. M. ,. H. A. M. M. M. K. ,. T. ,. B. ,. L. a. E. M. Houda El Alami, "FPGA in the loop implementation for observer sliding mode control of DFIG-generators for wind turbines," 30 December 2021. https://www.mdpi.com/2079-9292/11/1/116.

[18] M. T. A. H. Franc Mihalič, "Hardware-in-the- loop simulations: A historical overview of engineering challenges," 8 August 2022. https://www.mdpi.com/2079-9292/11/15/2462.

[19] Y. N. F. L. &. Z. X. Xiuxiu Wang, "When FPGA meets cloud: A first look at performance.," 04 May 2020. https://ieeexplore.ieee.org/abstract/document/9086121.

[20] A. J. T. M. P. N. P. M. N. Amir HajiRassouliha, " Suitability of recent hardware accelerators (DSPs, FPGAs, and GPUs) for computer vision and image processing algorithms," 2018. https://www.sciencedirect.com/science/article/pii/S0923596518303606#sec2.

[21] O. Gazi, A tutorial introduction to VHDL programming, Springer, 2019. https://link.springer.com/chapter/10.1007/978-981-13-2309-6_6

[22] S. R. &. R. B. S. Suhas B. Shirol, ".Design and implementation of adders and multiplier in FPGA using ChipScope: A performance improvement," 31 August 2018. https://link.springer.com/chapter/10.1007/978-981-13-0586-3_2.


```python
import numpy as np

# Parameters
amplitude = 1165            # Peak amplitude
num_points = 256            # Resolution of
    the lookup table

# Generate 0 degree base sine wave
sine_0 = [ int(amplitude * (np.sin(2 * np.pi *
     i / num_points) + 1) / 2)
    for i in range(num_points)
]

# Function to generate shifted sine wave
```

```python
def generate_shifted(data, degrees):
    shift_points = int((degrees / 360.0) * len(data))
    indices = np.arange(len(data))
    shifted_indices = (indices - shift_points) % len(data)
    return np.interp(indices, shifted_indices, data)

# Generate phase-shifted waves
sine_minus120 = generate_shifted(sine_0, -120).astype(int).tolist()  # Phase B
sine_plus120 = generate_shifted(sine_0, 120).astype(int).tolist()    # Phase C

# Output the lookup tables
print("Phase A (0 degree):\n", sine_0)
print("\nPhase B (-120 degree):\n", sine_minus120)
print("\nPhase C (+120 degree):\n", sine_plus120)
```

Listing 1. Python code for generating 90°, 120°, and 240° phase-shifted lookup tables

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all; --try to use this library as much as possible.

entity sine_lut is
  generic(
    NUM_POINTS : integer := 32;
    MAX_AMPLITUDE : integer := 230
  );
  port (
    clk : in std_logic;
    Va  : out integer range 0 to MAX_AMPLITUDE
  );
end sine_lut;

architecture Behavioral of sine_lut is
  signal i : integer range 0 to NUM_POINTS := 0;

  type memory_type is array (0 to NUM_POINTS -1) of integer range 0 to MAX_AMPLITUDE;
  -- ROM for storing the sine values generated by MATLAB.
  signal sine : memory_type := (
    115, 137, 159, 179, 196, 211, 221, 228, -- 1st quarter
    230, 228, 221, 211, 196, 179, 159, 137, -- 2nd quarter
    115, 93, 71, 51, 34, 19, 9, 2,          -- 3rd quarter
    0, 2, 9, 19, 34, 51, 71, 93             -- 4th quarter
  );

begin
  process(clk)
  begin
    if rising_edge(clk) then
      Va <= sine(i);
      i <= i + 1;
      if i = NUM_POINTS - 1 then
        i <= 0;
      end if;
    end if;
  end process;
end Behavioral;
```

Listing 2. VHDL Code for Sine Look-Up Table (sine_lut)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity stator is
  generic(
    NUM_POINTS_120 : integer := 256;
    MAX_AMPLITUDE_120 : integer := 1165;
    NUM_POINTS_0 : integer :=256;
    MAX_AMPLITUDE_0 : integer := 1165;
    NUM_POINTS_240 : integer := 256;
    MAX_AMPLITUDE_240 : integer := 1165
  );
  port (
    clk :in std_logic;
    Ia : out integer range 0 to MAX_AMPLITUDE_120;
    Ib : out integer range 0 to MAX_AMPLITUDE_0;
    Ic : out integer range 0 to MAX_AMPLITUDE_240
  );
end stator;

architecture Behavioral of stator is
signal i : integer range 0 to NUM_POINTS_120 := 0;
type memory_type_120 is array (0 to NUM_POINTS_120-1) of integer range 0 to MAX_AMPLITUDE_120;
--ROM for storing the sine values generated by MATLAB.
signal sine_120 : memory_type_120 :=
(455, 469, 483, 497, 511, 525, 540, 554, 568,
    583, 597, 611, 625, 640, 654, 668, 682,
    696, 710, 724, 738, 752, 765, 779, 792,
    805, 819, 832, 844, 857, 870, 882, 894,
    906, 918, 929, 941, 952, 963, 974, 984,
    994, 1004, 1014, 1024, 1033, 1042, 1050,
    1059, 1067, 1075, 1082, 1089, 1096, 1103,
    1109, 1115, 1121, 1126, 1131, 1136, 1140,
    1144, 1148, 1151, 1154, 1156, 1159, 1161,
    1162, 1163, 1164, 1165, 1165, 1165, 1164,
    1163, 1162, 1161, 1159, 1156, 1154, 1151,
    1148, 1144, 1144, 1140, 1136, 1131, 1126,
    1121, 1115, 1109, 1103, 1096, 1089, 1082,
    1075, 1067, 1059, 1050, 1042, 1033, 1024,
    1014, 1004, 994, 984, 974, 963, 952, 941,
    929, 918, 906, 894, 882, 870, 857, 844,
    832, 819, 805, 792, 779, 765, 752, 738,
    724, 710, 696, 682, 668, 654, 640, 625,
    611, 597, 583, 568, 554, 540, 525, 511,
    497, 483, 469, 455, 441, 427, 413, 400,
    386, 373, 360, 346, 333, 321, 308, 295,
    283, 271, 259, 247, 236, 224, 213, 202,
    191, 181, 171, 161, 151, 141, 132, 123,
    115, 106, 98, 90, 83, 76, 69, 62, 56, 50,
    44, 39, 34, 29, 25, 21, 17, 14, 11, 9, 6,
    4, 3, 2, 1, 0, 0, 1, 2, 3, 4, 6, 9, 11,
    14, 17, 21,25, 29, 34, 39, 44, 50, 56, 62,
```

```
     69, 76, 83, 90, 98, 106, 115, 123, 132,
    141, 151, 161, 171, 181, 191, 202, 213,
    224, 236, 247, 259, 271, 283, 295, 308,
    321, 333, 346, 360, 373, 386, 400, 413,
    427, 441);

signal j : integer range 0 to NUM_POINTS_0 :=
    0;
type memory_type_0 is array ( 0 to
    NUM_POINTS_0 - 1) of integer range 0 to
MAX_AMPLITUDE_0;
signal sine_0 : memory_type_0 :=
(583, 597, 611, 625, 640, 654, 668, 682, 696,
    710, 724, 738, 752, 765, 779,
792, 805, 819, 832, 844, 857, 870, 882, 894,
    906, 918, 929, 941, 952, 963,
974, 984, 994, 1004, 1014, 1024, 1033, 1042,
    1050, 1059, 1067, 1075, 1082, 1089, 1096,
    1103, 1109, 1115, 1121, 1126, 1131, 1136,
    1140, 1144, 1148, 1151, 1154, 1156, 1159,
    1161, 1162, 1163, 1164, 1165, 1165, 1165,
    1164, 1163, 1162, 1161, 1159, 1156, 1154,
    1151, 1148, 1144, 1140, 1136, 1131, 1126,
    1121, 1115, 1109, 1103, 1096, 1089, 1082,
    1075, 1067, 1059, 1050, 1042, 1033, 1024,
    1014, 1004, 994, 984, 974, 963, 952, 941,
    929, 918, 906,
894, 882, 870, 857, 844, 832, 819, 805, 792,
    779, 765, 752, 738, 724, 710,
696, 682, 668, 654, 640, 625, 611, 597, 583,
    568, 554, 540, 525, 511, 497,
483, 469, 455, 441, 427, 413, 400, 386, 373,
    360, 346, 333, 321, 308, 295,
283, 271, 259, 247, 236, 224, 213, 202, 191,
    181, 171, 161, 151, 141, 132,
123, 115, 106, 98, 90, 83, 76, 69, 62, 56, 50,
     44, 39, 34, 29, 25, 21, 17, 14, 11, 9, 6,
     4, 3, 2, 1, 0, 0, 0, 1, 2, 3, 4, 6, 9,
    11, 14, 17, 21, 25, 29, 34, 39, 44, 50,
    56, 62, 69, 76, 83, 90, 98, 106, 115, 123,
     132, 141, 151, 161, 171, 181, 191, 202,
    213, 224, 236, 247, 259, 271, 283, 295,
    308, 321, 333, 346, 360, 373, 386, 400,
    413, 427, 441, 455, 469, 483, 497, 511,
    525, 540, 554, 568);

signal k : integer range 0 to NUM_POINTS_240
    := 0;
type memory_type_240 is array ( 0 to
    NUM_POINTS_240 - 1) of integer range 0 to
MAX_AMPLITUDE_240;
signal sine_240 : memory_type_240 :=
( 171, 161, 151, 141, 132, 123, 115, 106, 98,
    90, 83, 76, 69, 62, 56, 50, 44, 39, 34,
    29, 25, 21, 17, 14, 11, 9, 6, 4, 3, 2, 1,
    0, 0, 1, 2, 3, 4, 6, 9, 11, 14, 17, 21,
    25, 29, 34, 39, 44, 50, 56, 62, 69, 76,
    83, 90, 98, 106, 115, 123, 132, 141, 151,
    161, 171, 181, 191, 202, 213, 224, 236,
    247, 259, 271, 283, 295, 308, 321, 333,
    346, 360, 373, 386, 400, 413, 427, 441,
    455, 469, 483, 497, 511, 525, 540, 554,
    568, 583, 597, 611, 625, 640, 654, 668,
    682, 696, 710, 724, 738, 752, 765, 779,
    792, 805, 819, 832, 844, 857, 870, 882,
    894, 906, 918, 929, 941, 952, 963, 974,
    984, 994, 1004, 1014, 1024, 1033, 1042,
    1050, 1059, 1067, 1075, 1082, 1089, 1096,
    1103, 1109, 1115, 1121, 1126, 1131, 1136,
    1140, 1144, 1148, 1151, 1154, 1156, 1159,
    1161, 1162, 1163, 1164, 1165, 1165, 1165,
    1164, 1163, 1162, 1161, 1159, 1156, 1154,
    1151, 1148, 1144, 1144, 1140, 1136, 1131,
    1126, 1121, 1115, 1109, 1103, 1096, 1089,
    1082, 1075, 1067, 1059, 1050, 1042, 1033,
    1024, 1014, 1004, 994, 984, 974, 963,
    952,941, 929, 918, 906, 894, 882, 870,
    857, 844, 832, 819, 805, 792, 779, 765,
    752, 738, 724, 710, 696, 682, 668, 654,
    640, 625, 611, 597, 583, 568, 554, 540,
    525, 511, 497, 483, 469, 455, 441, 427,
    413, 400, 386, 373, 360, 346, 333, 321,
    308, 295, 283, 271, 259, 247, 236, 224,
    213, 202, 191, 181);
begin
process(clk)
begin
    --to check the rising edge of the clock
        signal
    if(rising_edge(clk)) then
        --one by one output the sine
            amplitudes in each clock cycle.
        Ia <= sine_120(i);
        i <= i+ 1; --increment the index.
        if(i = NUM_POINTS_120-1) then
        --reset the index to zero, once we
            have output all values in ROM
            i <= 0;
        end if;
        Ib <= sine_0(j);
        j <= j+1;
        if ( j = NUM_POINTS_0 - 1) then
            j <=0;
        end if;
        Ic <= sine_240(k);
        k <= k+1;
        if (k = NUM_POINTS_240 - 1) then
            k <= 0;
        end if;
    end if;
end process;
end Behavioral;
```

Listing 3. VHDL Code for Stator Module

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity rotor is
  generic(
    NUM_POINTS_Torque : integer := 256;
    MAX_AMPLITUDE_Torque : integer := 7;
    NUM_POINTS_rotorS : integer := 256;
    MAX_AMPLITUDE_rotorS : integer := 1475;
    NUM_POINTS_current : integer := 256;
    MAX_AMPLITUDE_current : integer := 3
  );
  port(
    clk : in std_logic;
    Torque : out integer range 0 to
        MAX_AMPLITUDE_Torque;
    Rotor_Speed : out integer range 0 to
        MAX_AMPLITUDE_rotorS;
```

```
    Rotor_current : out integer range 0 to
        MAX_AMPLITUDE_current
  );
end rotor;

architecture Behavioral of rotor is
  signal i : integer range 0 to
      NUM_POINTS_Torque := 0;
  signal j : integer range 0 to
      NUM_POINTS_rotorS := 0;
  signal k : integer range 0 to
      NUM_POINTS_current := 0;

  type memory_type_Torque is array (0 to
      NUM_POINTS_Torque-1) of integer range 0
      to MAX_AMPLITUDE_Torque;
  type memory_type_rotorS is array (0 to
      NUM_POINTS_rotorS-1) of integer range 0
      to MAX_AMPLITUDE_rotorS;
  type memory_type_current is array (0 to
      NUM_POINTS_current-1) of integer range 0
       to MAX_AMPLITUDE_current;

  signal sine_Torque : memory_type_Torque := (
    4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 6, 6,
        6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
       6, 6, 6, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,
        7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,
       7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,
        7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 6, 6,
       6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
        6, 6, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
        4, 4, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
        2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3,
        3, 3, 3, 3);

  signal sine_rotorS : memory_type_rotorS := (
   738, 756, 774, 792, 810, 828, 846, 864,
       881, 899, 917, 934, 952, 969, 986,
       1003, 1020, 1036, 1053, 1069, 1085,
       1101, 1117, 1132, 1147, 1162, 1177,
       1191, 1205, 1219, 1233, 1246, 1259,
       1272, 1284, 1296, 1308, 1319, 1330,
       1340, 1351, 1361, 1370, 1379, 1388,
       1396, 1404, 1412, 1419, 1426, 1432,
       1438, 1443, 1448, 1453, 1457, 1461,
       1464, 1467, 1469, 1471, 1473,
       1474,1475, 1475, 1475, 1474, 1473,
       1471, 1469, 1467, 1464, 1461, 1457,
       1453, 1448, 1443, 1438, 1432, 1426,
       1419, 1412, 1404, 1396, 1388, 1379,
       1370, 1361, 1351, 1340, 1330, 1319,
       1308, 1296, 1284, 1272, 1259, 1246,
       1233, 1219, 1205, 1191, 1177, 1162,
       1147, 1132, 1117, 1101, 1085, 1069,
       1053, 1036, 1020, 1003, 986, 969, 952,
       934, 917, 899, 881, 864, 846, 828, 810,
        792, 774, 756, 738, 719, 701, 683,
       665, 647, 629, 611, 594, 576, 558, 541,
        523, 506, 489, 472, 455, 439, 422,
       406, 390, 374, 358, 343, 328, 313, 298,
        284, 270, 256, 242, 229, 216, 203,
       191, 179, 167, 156, 145, 135, 124, 114,
        105, 96, 87, 79, 71, 63, 56, 49, 43,
       37, 32, 27, 22, 18, 14, 11, 8, 6, 4, 2,
        1, 0, 0, 0, 1, 2, 4, 6, 8, 11, 14, 18,
        22, 27, 32, 37, 43, 49, 56, 63, 71,
       79, 87, 96, 105, 114, 124, 135, 145,
       156, 167, 179, 191, 203, 216, 229, 242,
        256, 270, 284, 298, 313, 328, 343,
       358, 374, 390, 406, 422, 439, 455, 472,
        489, 506, 523, 541, 558, 576, 594,
       611, 629, 647, 665, 683, 701, 719);

  signal sine_current : memory_type_current :=
       (
    2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1);

begin
  process(clk)
  begin
    if rising_edge(clk) then
      Torque <= sine_Torque(i);
      i <= i + 1;
      if i = NUM_POINTS_Torque - 1 then i <=
          0; end if;

      Rotor_Speed <= sine_rotorS(j);
      j <= j + 1;
      if j = NUM_POINTS_rotorS - 1 then j <=
          0; end if;

      Rotor_current <= sine_current(k);
      k <= k + 1;
      if k = NUM_POINTS_current - 1 then k <=
          0; end if;
    end if;
  end process;
end Behavioral;
```

Listing 4. VHDL Code for Rotor Simulation Module