**ELTI**
**Jurnal Elektronika, Listrik dan Teknologi Informasi Terapan**
Volume 6, No 2, Desember 2024 (x-x) https://ojs.politeknikjambi.ac.id/elti

# Design and Simulation Multichannel PWM Based on Modbus Protocol in FPGA device

Sigit Kurniawan [a, *], Mahmud Idris[a], Sepdian[b], Mazwan[c]

[a]*Program Studi Teknik Elektronika, Politeknik Jambi, Jln Lingkar Barat 2 Kota Jambi, Indonesia*

[b]*Program Studi Teknik Listrik, Politeknik Jambi, Jln Lingkar Barat 2 Kota Jambi, Indonesia*

[c]*Program Studi Teknik Mesin, Politeknik Jambi, Jln Lingkar Barat 2 Kota Jambi, Indonesia*

*E-mail: sigit@politeknikjambi.ac.id*, mahmud@politeknikjambi.ac.id,

sepdian@politeknikjambi.ac.id, mazwan@politeknikjambi.ac.id

** Corresponding Author:*

*Abstract— Pulse Width Modulation (PWM) is a widely employed technique in power converter circuits. Given the prevalence of power converters in industrial settings, there is a need for a straightforward and efficient communication method among devices, such as the Modbus protocol. This study focuses on developing and simulating a multichannel PWM architecture integrated with Modbus modules. The objective is to create a multichannel PWM architecture with a Modbus module suitable for implementation on FPGA devices. The findings indicate that the architecture successfully facilitates the transmission and reception of Modbus frame data from the Modbus master. Data transmitted by the Modbus Master is utilized for reading or writing registers within the architecture, employing standard function codes such as 0x30, 0x04, 0x06, and 0x10. One of these functions serves to modify the duty value of the four PWM channels.*

*Keywords*— ***PWM, Modbus, FPGA, Parallel Converter***

Abstrak— Modulasi Lebar Pulsa (PWM) merupakan teknik yang banyak digunakan dalam rangkaian konverter daya. Mengingat maraknya konverter daya dalam pengaturan industri, diperlukan metode komunikasi yang mudah dan efisien di antara perangkat, seperti protokol Modbus. Studi ini berfokus pada pengembangan dan simulasi arsitektur PWM multisaluran yang terintegrasi dengan modul Modbus. Tujuannya adalah untuk membuat arsitektur PWM multisaluran dengan modul Modbus yang cocok untuk diimplementasikan pada perangkat FPGA. Temuan menunjukkan bahwa arsitektur tersebut berhasil memfasilitasi transmisi dan penerimaan data bingkai Modbus dari master Modbus. Data yang dikirimkan oleh Master Modbus digunakan untuk membaca atau menulis register dalam arsitektur, menggunakan kode fungsi standar seperti 0x30, 0x04, 0x06, dan 0x10. Salah satu fungsi ini berfungsi untuk mengubah nilai tugas dari empat saluran PWM.

*Kata kunci*— **Kata Kunci 1, kata Kunci 2, Kata Kunci 3**

## I. INTRODUCTION

Numerous Pulse Width Modulation (PWM) techniques are employed for the control of analog circuits, predominantly in power converters, as noted in [1]. Given the widespread use of power converters in industrial settings, there is a requirement for a communication module that can operate concurrently to streamline tasks stemming from PWM, as highlighted in [2]. In this context, the prevalent choice for communication protocols among industries is the Modbus standard.

The design of PWM and protocol architecture for communication in Programmable Logic devices (PLD), such as CPLD or FPGA, presents a versatile model. These devices offer adaptable architectures, enhanced speed, parallel operation, and support for Single on Chip (SoC) architecture development, as referenced by a knowledgeable source. As the focus on SoC architecture increases among researchers, the integration of a communication module and PWM module using FPGA components becomes a viable option.

Numerous researchers are actively involved in the developmental design of UART communication protocols, including Modbus, on FPGA components, as evidenced by the works of Shikar [3], Byung [2], and Lingxi Kong [4]. Shikar's contribution involves developing a UART module on a FPGA tyni, equipped with Baudrate detection to facilitate flexible and efficient data delivery from a PC. In a different approach, Lingxi Kong, Qirui Niu, and Pai Yang propose a more stable UART design incorporating a signal propagation mechanism for enhanced signal reception [4]. Additionally, Mohammad Awedh and Ahmed Mueen put forth a UART Design featuring a conceptual model, which combines elements of a microprocessor, integrating Datapath and Control unit [5].

This paper focuses on the development and simulation of a multi-channel PWM architecture incorporating a Modbus module within the Altera FPGA device. The Modbus architecture adheres to the Modbus RTU protocol, comprising both transmitter and receiver modules. The multichannel PWM system features four outputs with a 10-bit resolution, constructed from comparators and counters. It has the capability to operate simultaneously based on the values stored in the holding register of the Modbus Architecture's register file.

## II. METHODS

The top module responsible for generating multichannel PWM employs Modbus communication, as depicted in Figure 1. The Rx and Tx pins facilitate serial communication with devices. The master utilizes the Modbus protocol, with a clock signal input (CLK_100 MHz), device ID as the architecture's ID for Slave, a global reset pin for module-wide reset, and PWM pins 0 to 3 as the output pins for the four-channel PWM.

Within the devised architecture, there are three primary modules: the UART module, Control Unit module, and PWM module. The architecture receives data from the master through the receiver module in byte format within each data frame from the Modbus protocol. For every byte received by the receiver module, a signal labeled "rx_done" is generated in the Control Unit module. Subsequently, the Control Unit executes calculations based on the number of bytes within one Modbus data frame. The Control Unit reads commands from the received data in the form of Modbus function codes, which are then translated to either read or write registers.

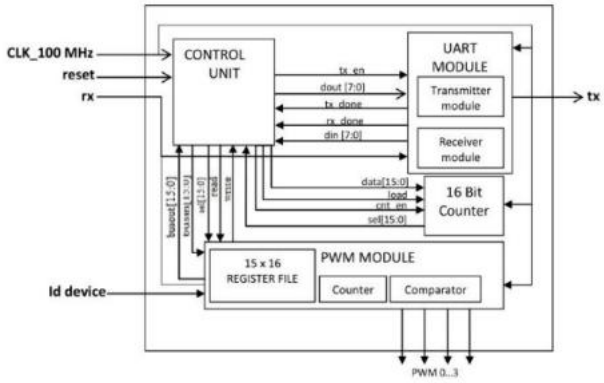


Figure 1. Blok diagram of the proposed Architecture

### 2.1 UART Modules

The communication module utilizing the Modbus protocol is implemented through the UART module, which comprises both the transmitter and receiver modules. The transmitter module follows a parallel in, serial out (PISO) register concept, whereas the receiver module operates in the reverse manner, employing a serial in, parallel out (SIPO) register. Illustrated in Figure 2, the UART module takes input pins from the clock, reset, receive (rx), data input (din [7:0]), and transmit enable (tx_en), while output pins include data out (dout [7:0]), transmit (tx), receive done (rx_done), and transmit done (tx_done).

In Modbus communication, a single Modbus data frame encompasses a series of byte-by-byte data, reaching up to 255 data bytes. Each byte within the data frame consists of 10 bits—comprising 1 start bit, 8 data bits, and 1 stop bit. The operational sequence of the receiver module is delineated in Figure 3. A transition change from '1' to '0' on the rx pin, identified by the edge detector, triggers the bit counter in the receiver module to count from 1 to 10 (Fig. 3a.). This count is based on the bit flag signal (Fig. 3b.) derived from the baud rate calculation. The received data (Fig. 3c.) is then consolidated in the parallel register within the receiver module. Once 10 bits of data are received, the rx_done pulse signal becomes '1' (Fig. 3d.), signifying the reception of one byte of

data, ready for transfer from the UART module to the buffer register in the Control Unit module. This data is conveyed through the dout [7:0] pin, and all data bytes are temporarily stored in the buffer register until a complete Modbus data frame has been received [6].

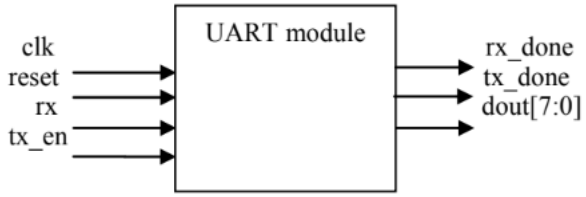


Figure 2. UART module

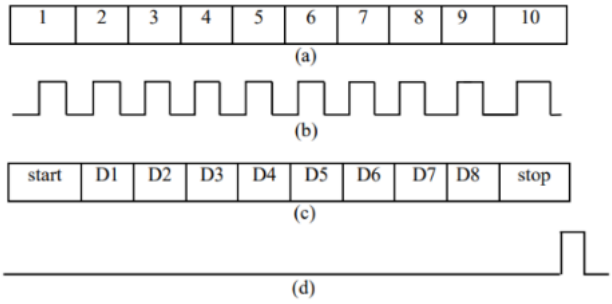


Figure 3. The Operation Diagram of Receiver Module, (a) Bit Counter Value, (b) Bit Flag Signal, (c) Frame Data of Byte, and (d) Rx_Done Signal

The rx_done signal serves as a trigger for initiating calculations to determine the number of bytes stored in the buffer. This quantity is then compared to the function code type of command from Modbus, which specifies the maximum number of bytes in a command. For instance, the command for reading both holding and input registers entails 4 bytes of data, while the command to write a single register comprises 4 bytes of data, and the command to write multiple registersinvolves twice the multiple of the number of registers written. Once it is established that the received bytes are complete, a state transition occurs in the Control Unit, progressing from the 'receiving data' state to the subsequent state.

Illustrated in Figure 4 is the operational process of the transmitter module. The module initiates data transmission when the signal on the tx_en pin (Fig. 4a.) shifts from '0' to '1'. Under this condition, the Baud Counter commences counting from 0 to 10,416 (Fig. 4b.), representing the Baudrate period of 9600 for the system clocked at 100 MHz. Upon reaching the value of 10,416, the Bit flag (Fig. 4c.) assumes the value '1', triggering the Bit counter to commence calculations (Fig. 4d.). Simultaneously, it provides an index to the parallel register of the transmitter module, initiating the output of data bit by bit via the tx pin (Fig. 4e). Upon the completion of data transfer, indicating that 10 bits of data have been transmitted, the 'tx_done' signal assumes the value '1' (Figure 4f).

Similar to the rx_done signal, the tx_done signal also serves to initiate calculations to determine the number of bytes transmitted from the transmitter module. This enables the Control Unit to transition from the 'sending data' state to the next state.

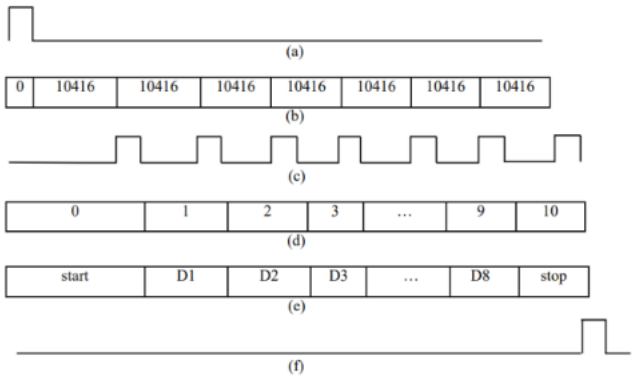


Figure 4. The operation diagram of transmitter module, a. tx_en signal, b. Baud counter value, c. Bit flag signal, d Bit counter value, e. frame data of byte, and f. tx_done signal

## 2.2 Control Unit

The Finite State Machine (FSM) is employed, as depicted in Figure 5, as the Control Unit within the designed Architecture. It comprises 11 states, with input signals including reset, rx_done, cell, and tx_done serving as signal statuses. Meanwhile, the outpu signals, namely load, cnt_en, read, write, and tx_en, are utilized to manage and control the UART module.

From s20 to s45, while for other function codes, it moves to s25 (reset 16-bit Counter) to reset the counter before writing/reading the register from 16x16 Registers. Subsequently, the machine shifts from s25 to s30 (read/write register) upon the completion of register reading/writing, indicated by the cell value matching the last address to be

written or read. From s30, it proceeds to s40 (reset 16-bit Counter) to reset the 16-bit counters. Following this, the machine enters s45 (data sending) totransmit data using a transmitter module. The control unit calculates each byte of data sent based on the received rx_done signal. If the amount of data sent matches the required number of bytes, the machine transitions to state s5, awaiting new data.

## 2.3 PWM Module

The Finite State Machine (FSM) is employed, as depicted in Figure 5, as the Control Unit within the designed Architecture. It comprises 11 states, with input signals including reset, rx_done, cell, and tx_done serving as signal statuses. Meanwhile, the output signals, namely load, cnt_en, read, write, and tx_en, are utilized to manage and control the UART module.

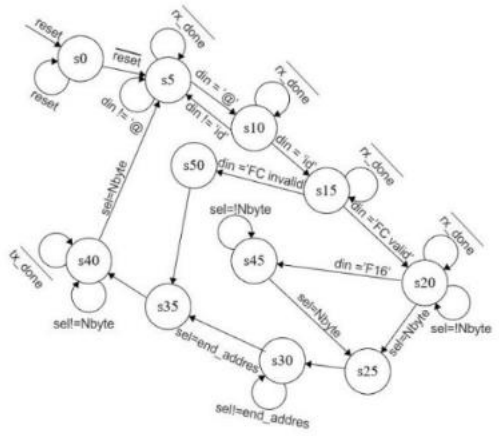


Figure 5. FSM of Control Unit

Upon receiving a reset input signal, the machine enters state s0 (reset state). The transition towards s5 occurs when the reset input is in a LOW or '0' condition. In state s1 (check similarity character), the machine awaits the HIGH condition of the rx_done signal, denoting the reception of one byte of data by the receiver module. If the received data byte aligns with the character '@', the machine shifts from s5 to s10 (check ID similarity). In s10, it verifies the next data byte triggered by the rx_done signal. If this data byte matches the device ID, the machine progresses to s15 (check function code) to validate the function code values. If the function code is valid, the machine advances to s20 (copy data to buffer); otherwise, it moves to s50 (invalid function code). In s20, distinction is made between function codes 03/04/06 and function For function code 16, the machine transitions from s20 to s45, while for other function codes, it moves to s25 (reset 16-bit Counter) to reset the counter before writing/reading the register from 16x16 Registers. Subsequently, the machine shifts from s25 to s30 (read/write register) upon the completion of register reading/writing, indicated by the cell value matching the last address to be written or read. From s30, it proceeds to s40 (reset 16-bit Counter) to reset the 16-bit counters. Following this, the machine enters s45 (data sending) to transmit data using a transmitter module. The control unit calculates each byte of data sent based on the received rx_done signal. If the amount of data sent matches the required number of bytes, the machine transitions to state s5, awaiting new data. code 16, as it relates to the length of the received data bytes.

## 2.4 PWM Module

The PWM module is constructed utilizing a Counter and comparator. The functionality of PWM involves comparing the duty values with a comparator value to generate a HIGH or LOW value for the PWM pulse. Illustrated in Figure 6 is the diagram of the Multi-channel PWM module, where the input duty value is derived from the Modbus register holding value. This input duty value is then compared with a 10-bit Counter within the PWM module. If the input duty value is less than the calculated value of the 10-bit counter, the PWM signal assumes a value of 1; otherwise, it takes on a value of 0.

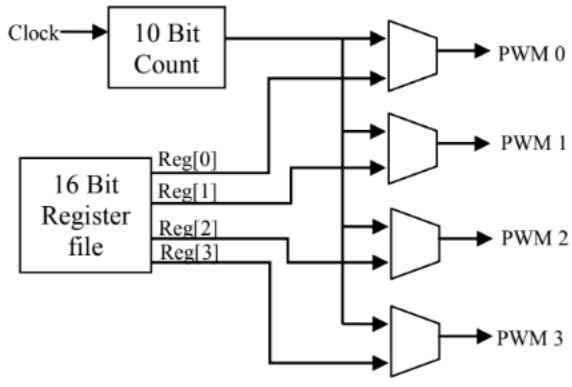


Figure 6. Module of Multi-channel PWM

## III. RESULTS AND DISCUSSIONS

### 3.1. Read Single/Multiple Register

Within the Modbus protocol, registers are categorized as either input registers or holding registers. When reading these registers, it is essential to identify the function code and the address of the register to be accessed. If there is a discrepancy in synchronization between the function code and the address of the register being read, the Slave will transmit an exception code to the Master.

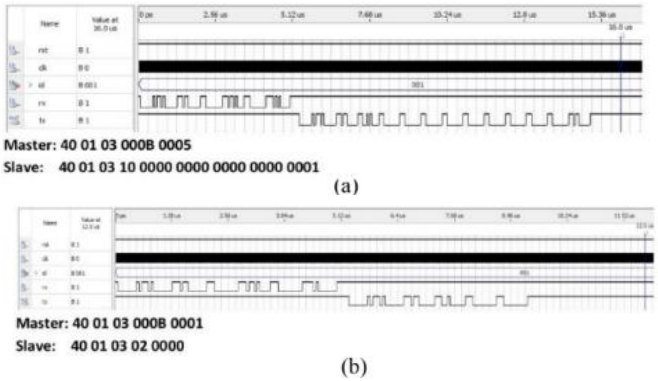


Figure 7. Testing of Communication Data Using Data Frame Modbus with 0x03 Function Code, (a) Write Single Register, (b) Read Multiple Register

Figure 7(a) illustrates the outcomes of the Modbus communication tests utilizing function codes 0x03 for single-register reading, while Figure 7(b) displays the test results for multiple-register reading. The data frame comprises the initial character '@' (0x40), ID (0x01), function code (0x03), start register (0x00, 0x0B), and length register (0x00, 0x05). This initial character serves as a separator between consecutive data frames, adhering to the ≥3.5character requirement in the Modbus standard [7][8]. Its inclusion is also necessitated by the architecture's design, where the CRC byte is disregarded [9].

The data frames for single and multiple register readings are identical, differing only in the requested address's start value and the register length. If the number of registers to be read is one, the register length is set to one; otherwise, for multi-register readings, the register length becomes N-Reg. Similarly, when reading Modbus Input Registers with function code 0x04, the data frame pattern remains the same as the one for function code 0x03. Thus, both commands 0x03 and 0x04 follow a similar sequence. In the designed architecture, exception codes are implemented to signal errors to the master. The exception codes include ILLEGAL FUNCTION and ILLEGAL DATA ADDRESS. ILLEGAL FUNCTION arises when the master transmits a function code not supported in the architecture, specifically function codes other than 0x03, 0x04, 0x06, and 0x10. ILLEGAL DATA ADDRESS occurs due to asynchronous commands between the function code and the register address to be read.

### 3.2. Write Single Register

The designed architecture incorporates five holding registers available for both reading and writing purposes. These holdings comprise four registers of 16 bits each, specifically allocated for duty cycle values of the four PWM channels. Additionally, one register is designated for modifying the Baud rate in Modbus communication.


Figure 8. Testing of Communication Data Using Data Frame Modbus with 0x06 Function Code for Writing Single Register

Figure 8 depicts the outcomes of communication testing in Modbus utilizing function code 0x06 for writing a single register at address 0x00, 0x0F. The transmitted data includes the initial character '@' (0x40), ID (0x01), the address of the register to be written, and its corresponding value. Following a successful writing process, the architecture, acting as the Slave, transmits data to the master, comprising the initial character '@' (0x40), ID (0x01), the address of the written register, and the value that was written. In case of a writing failure, the Slave sends an exception code in the form of ILLEGAL DATA ADDRESS.

### 3.3. Write Multiple Register

One of the advantages of Modbus communication is its capability to send or receive multiple bytes in a single package. The multi-register write command stands out as the lengthiest among various Modbus commands, with a message length reaching up to 256 bytes in Modbus RTU.

Regarding the data reception sequence, the architectural limitations for data length extend to 65,535 bytes. This is achievable due to the architecture's utilization of a 16-bit counter for counting iterations, avoiding the need for looping logic such as for... or while... constructs, as Verilog language restricts iterations to 255 cycles. Nevertheless, despite the potential for extensive data processing, the architecture's practical limit is constrained to a maximum of 18 bytes. This comprises 1 byte for the character, 1 byte for ID, 1 byte for the function code, 4 bytes for the register address, 1 byte for data, and 10 bytes for the data to be written in the register.

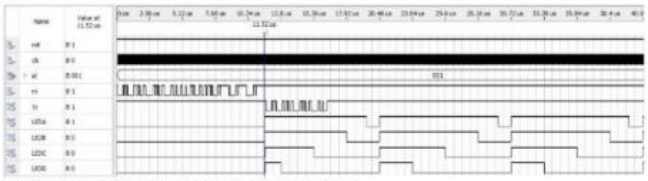


Figure 9. Testing of communication data using data frame Modbus with 0x16 function code for writing multi register

Figure 9 displays the outcomes of communication testing in Modbus, utilizing function code 0x16 for writing to multiple registers at addresses 0x00, 0x0B to 3, covering a total of 4 registers. The transmitted data includes the initial character '@' (0x40), ID (0x01), the starting address of the registers to be written, the number of registers to be written, the length of data bytes, and 8 bytes of values to be written across the 4 registers. Following a successful writing process, the architecture, operating as the Slave, transmits data to the master, including the initial characters '@' (0x40), ID (0x01), the initial register address, and the number of registers written. In the event of a writing failure, the Slave transmits an exception code in the form of ILLEGAL DATA ADDRESS [10].

The method employed for writing data to the register address in this architecture involves a counter module. During data transfer between the 8-bit buffer and a 16-bit register, the counter is simultaneously activated and commences counting based on the transferred data quantity. The counter calculation is ordered after the counter is initially activated [11], and before activation, the counter is reset with data from the initial register address. This data transfer necessitates a specific approach, where data in an 8-bit buffer is split based on the index value corresponding to the order of received data. This ensures the acquisition of data in a 16-bit format from the 8-bit data. The buffer data is concatenated based on the index order. For example, the value 000B represents an index in buffers 0 and 1, with the value 0398 at buffer indices 5 and 6. The iteration process is not based on loop calculations from 0 to 10, but instead relies on the iteration of the register addresses written. Consequently, the calculation of the register address must be converted into an index value ranging from 0 to 10.

### 3.4. Multichannel PWM

The digital PWM is constructed through a technique involving calculation and comparison. The calculated value of a counter serves as a comparison metric with the given duty value. If the counter value is less than the duty value, the PWM output will be set to 1; otherwise, it will be set to 0. Once the calculation is complete, the PWM output becomes 1, and the counter restarts its counting process. The period of the PWM is influenced by the resolution of the counter utilized—higher resolution leads to a longer PWM period [12]. In this architecture, a PWM module is designed with a resolution of 10 bits, where the counter also has a 10-bit value.

Figure 10. Waveform of 4 Chanel PWM with Duty Variation

Figure 10 illustrates the signal outcomes from PWM for each channel. The PWM signal for Channel LEDA exhibits a 90% duty cycle, as the provided duty value is 398h or 920 in decimal. This value represents 90% of the calculated maximum value of the 10-bit counter, namely 1024. Channel LEDB displays a PWM signal with a 75% duty cycle, attributed to the duty value of 2FF or 767 in decimal, equivalent to 75% of the maximum calculation. Similarly, LEDC and LED0 channels have duty cycles of 50% and 25%, respectively.

The PWM architecture in FPGA devices

offers distinct advantages, enabling the creation of parallel PWM channels, each with its own independent duty cycle. Additionally, the PWM frequency output can surpass that of other PWM generators, such as microcontrollers or embedded systems. An advantageous feature of the designed PWM module is its utilization of a single counter that is employed concurrently for all channels. This streamlined approach minimizes the utilization of logic blocks within the FPGA device.

## IV. CONCLUSION

A successful creation involved the development of a Multichannel PWM architecture featuring a Modbus communication module. This innovative design builds upon the existing UART receiver and transmitter module, along with the FSM module, for the purpose of managing the reception or transmission of data frame packets within the Modbus Protocol.

During testing, it was discovered that introducing a unique character at the start of data transmission could serve as a viable alternative to replacing the CRC byte. The minimum delay observed between bytes in the data stream is determined to be 2.5 bytes or greater. It's important to note that the limitation of this research lies in its reliance on simulation through HDL programming; thus, actual hardware implementation is necessary to confirm the absence of errors during data communication between two distinct devices. Additionally, to enhance the Modbus module architecture, the inclusion of a CRC module is suggested to achieve a design aligned with industry standards.

## V. ACKNOWLEDGEMENT

The Authors would like to thanks to Dinas Perumahan Rakyat, Kawasan Permukiman (DKKP) Kota Jambi Government and Politeknik Jambi for the funding support grant “Design and development of LED lighting driver module”. This research was fully funded by Politeknik Jambi and Kota Jambi Government.

## REFRENSI

[1] R. Srivastava and Y. K. Chauhan, “Generation of pwm using verilog in fpga,” in Int. Conf. Electr. Electron. Optim. Tech., 2016, no. March, pp. 4593–4597. doi: 10.1109/ICEEOT.2016.7755586.

[2] B. Park, S. Park, and F. Kang, “A novel communication method using pwm and capture function of dsp for parallel controlled power electronics systems,” IEEE Access, vol. 10, no. May, pp. 68266–68280, 2022, doi: 10.1109/ACCESS.2022.3186690.

[3] Shikhar, “Design and implementation of multiple pwm channels using universal asynchronous receiver transmitter,” in Fourth Int. Conf. Electron. Commun. Aerosp. Technol., 2020, pp. 485–492.

[4] L. Kong, Q. Niu, and P. Yang, “Design and implementation of uart based on verilog hdl,” Highlights Sci. Eng. Technol., vol. 38, no. 2023, pp. 949–955, 2023.

[5] M. Awedh and A. Mueen, “Design and fpga implementation of uart using microprogrammed controller,” Sch. J. Eng. Technol., vol. 3, no. August, pp. 600–608, 2015.

[6] I. A. Adamovich and Y. A. Klimov, “An fpga packet communication protocol,” Progr. Syst. Theory Appl., vol. 11, no. 1, pp. 57–78, 2020, doi: 10.25209/2079-3316-2020-11-1-57-78.

[7] I. Zagan and V. G. Gaitan, “Enhancing the modbus communication protocol to minimize acquisition times based on an stm32-embedded device,” mathematics, vol. 10, no. 4686, pp. 1–19, 2022.

[8] P. Rane, “Design of modbus controller using vhdl for remote administrations of a network of devices,” in Third Int. Conf. Emerg. Trends Eng. Technol., 2010, vol. 67, pp. 694–697. doi: 10.1109/ICETET.2010.99.

[9] J. N. Chhatrawala, N. Jasani, and V. Tilva, “FPGA based data acquisition with modbus protocol,” in Int. Conf. Commun. Signal Process., 2016, pp. 1251–1254.

[10] D. De Paula, L. Caetano, M. Pereira, A. Leivas, R. Steffens, and C. Botelho, “Welding turns digital: electronics and fpga-based design to actuate a linear welding work cell,” ICCEEg, vol. 1, no. 14, pp. 35–43, 2016.

[11] P. Pagare, S. Upadhya, J. Desai, M. Kumbhare, K. Thakur, and R. Krishnan, “Implementation of modbus on fpga,” in 3RD Int. Conf. Innov. Comput. Commun.,

2020, pp. 1–5.
[12] M. Facta, T. Sutikno, and Z. Salam, “The application of fpga in pwm controlled resonant converter for an ozone generator,” Int. J. Power Electron. Drive Syst., vol. 3, no. 3, pp. 336–343, 2013.