

INVESTIGATION OF APB PROTOCOL IN NON-WAIT STATE IN SEMICONDUCTOR CHIP DESIGN

Nghiên cứu giao thức APB ở trạng thái không chờ trong thiết kế vi mạch bán dẫn

Lâm Minh Long⁽¹⁾, Phạm Thúy Oanh⁽¹⁾, Vũ Thị Ngọc Thu^{(2)*}

⁽¹⁾Học viện Hàng không Việt Nam

⁽²⁾Trường Đại học Công nghệ Kỹ thuật Thành phố Hồ Chí Minh

Abstract

With the rapid development of low-power SoC and embedded systems, efficient peripheral communication protocols have become increasingly important in semiconductor chip design. This study investigates the Advanced Peripheral Bus (APB) protocol in non-wait-state data transmission for low-bandwidth peripheral applications. The research focuses on the operating principles, communication timing, signal structures, and integration characteristics of the APB protocol within the AMBA architecture. Simulation and functional verification were performed using Vivado 2024.2.1 to analyze continuous read/write transactions and signal synchronization behavior. The results demonstrate that the APB protocol provides simple hardware implementation, stable communication timing, low power consumption, and efficient integration with AHB/AXI-based SoC systems, making it suitable for low-power embedded and IoT applications.

Keywords: APB, AMBA, AHB, IP core, AXI.

Tóm tắt

Với sự phát triển nhanh chóng của SoC công suất thấp và các hệ thống nhúng, các giao thức truyền thông ngoại vi hiệu quả ngày càng trở nên quan trọng trong thiết kế chip bán dẫn. Nghiên cứu này khảo sát giao thức Bus ngoại vi nâng cao (APB) trong truyền dữ liệu không chờ đợi cho các ứng dụng ngoại vi băng thông thấp. Nghiên cứu tập trung vào nguyên lý hoạt động, thời gian truyền thông, cấu trúc tín hiệu và đặc điểm tích hợp của giao thức APB trong kiến trúc AMBA. Mô phỏng và kiểm chứng chức năng được thực hiện bằng Vivado 2024.2.1 để phân tích các giao dịch đọc/ghi liên tục và hành vi đồng bộ hóa tín hiệu. Kết quả cho thấy giao thức APB cung cấp khả năng triển khai phần cứng đơn giản, thời gian truyền thông ổn định, tiêu thụ điện năng thấp và tích hợp hiệu quả với các hệ thống SoC dựa trên AHB/AXI, làm cho nó phù hợp với các ứng dụng nhúng công suất thấp và IoT.

Từ khóa: APB, AMBA, AHB, IP core, AXI.

1. Introduce

APB is part of the AMBA 3 protocol family. It provides a low-cost interface optimized for minimal power consumption and reduced interface complexity. APB communicates with any low-bandwidth peripheral that does not require the high performance of a pipeline bus protocol. APB does not have a pipeline-like protocol. All signal transitions involve only the rising.

*Corresponding author: thuvtn@hcmute.edu.vn

edge of the clock to allow easy integration of APB peripherals into any design flow. Each transition takes at least two cycles. APB can communicate with AMBA Advanced High-performance Bus Lite (AHB-Lite) and AMBA Advanced Extensible Interface (AXI). It can be used to provide access to the programmable control registers of IoT devices, low-power MCUs, AI chips, etc. [1,2,3].

In this study, an 8-bit APB configuration operating in non-wait-state mode was selected to simplify timing analysis and evaluate the fundamental behavior of APB communication in low-complexity embedded systems. The 8-bit architecture is suitable for low-bandwidth peripherals such as GPIO, UART, SPI, and sensor interfaces commonly used in IoT and low-power microcontroller applications.

In addition, the non-wait-state configuration allows each transfer to complete in a deterministic two-cycle operation without additional delay introduced by the slave device. This approach helps simplify waveform verification, reduce protocol complexity, and provide a clear evaluation of synchronization and signal timing characteristics before extending the design toward more complex wait-state or high-bandwidth communication architectures.

The simplified 8-bit non-wait-state model also provides a suitable foundation for future development of bridge-based integration into modern 32-bit SoC platforms

2. Learn the transmission principle of APB protocol

There are two types of data transfer read and write:

- With no-wait states
- With wait states.

In the framework of this paper, we only examine the 8-bit APB protocol with data transfer non-wait states.

The state diagram in Figure 1 illustrates the transition logic, where: in the first state IDLE is established when $PSELx = 0$, $PENABLE = 0$, this is the default state of the APB. If these two conditions are met, the data or signal will be transmitted to the second stage, which is SETUP, The bus only remains in the SETUP state for one clock cycle and always moves to the ACCESS state on the next rising edge of the clock. At which time $PSELx = 1$ and $PENABLE = 0$, if the above conditions are not met, the transmission process will not appear. Data transfer write can be performed in this stage. However, with the same non-wait state, the Transmit read signal does not appear in this stage but must wait for the third state, which is ACCESS. The enable signal, $PENABLE$, is asserted in the ACCESS state. The address, write, select, and write data signals must remain stable during the transition from the SETUP to ACCESS state. At which time, $PSELx = 1$, $PENABLE = 1$. Note that, Exit from the ACCESS state is controlled by the $PREADY$ signal from the slave:

If **PREADY** is held LOW by the slave then the peripheral bus remains in the ACCESS state.

If PREADY is driven HIGH by the slave then the ACCESS state is exited and the bus returns to the IDLE state if no more transfers are required. Alternatively, the bus moves directly to the SETUP state if another transfer follows.

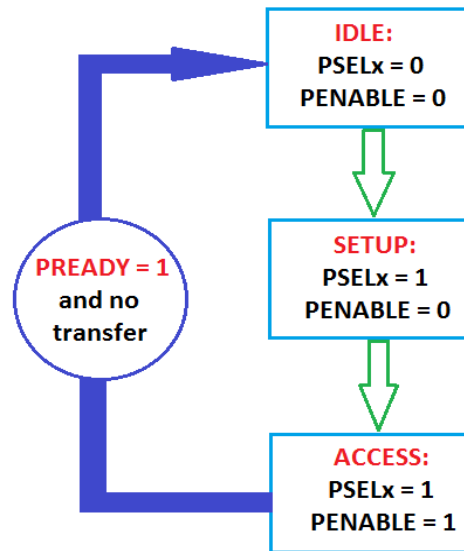


Figure 1. State diagram

Considering the read signal transmission state (Figure 2) and the write data transmission state (Figure 3) without the waiting state, we see:

First, consider the data reading process, in the first phase (called IDLE), at time T0 to T1, at this time PSEL and PENABLE = 0, the initial initialization phase. From IDLE through SETUP, at time T1 to T2 (transfer to the read data transmission phase, at this time PSEL = 1, PENABLE = 0, this is the transmission phase. From SETUP through ACCESS, at this time PSEL = 1 and PENABLE = 1, data is read from the register. Finally, in the phase from T3 to T4, PSEL and PENABLE both return to 0, then return to the IDLE phase.

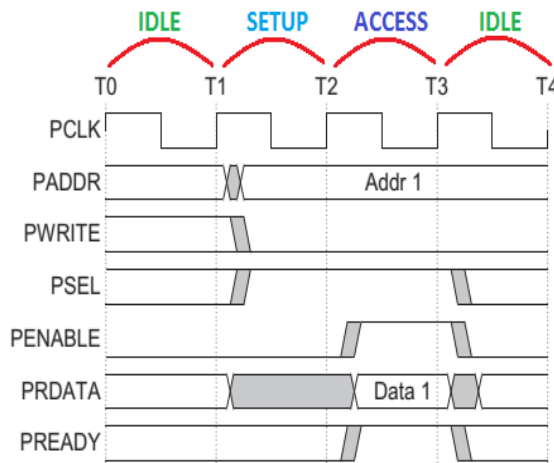


Figure 2. Transmit read signal non-wait state

Figure 2 shows a read data transfer. The timing of the address, write, select, and enable signals is as described above. The slave must provide data before the read data transfer is complete.

The APB protocol write transfer is performed similarly, starting with the address, write data, write signal, and select signal all changing after the rising edge of the clock. The first clock cycle of the transfer is called the Setup phase. After the next clock edge, the enable signal is determined, PENABLE, and this indicates that the Access phase is in progress. All address, data, and control signals remain valid during the Access phase. The transfer is complete at the end of this cycle. Figure 3 shows a basic write transfer without a wait state [1].

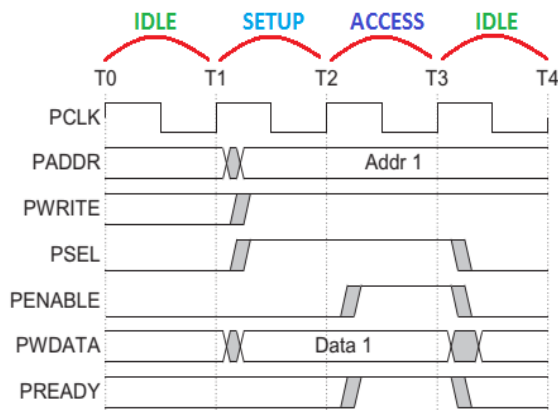


Figure 3. Data transfer write non-waiting state

The enable signal, PENABLE, is deasserted at the end of a data transfer. The select signal, PSELx, also goes low unless the data transfer is immediately followed by another data transfer to the same peripheral.

Table 1. List the APB signal descriptions (Note: Although the APB standard supports up to 32-bit, this research focuses on implementing an 8-bit configuration to optimize area for low-power MCU applications)

Signal	Description
PCLK	Clk pulse. The rising edge of PCLK times all transfers on APB.
PRESETn	Reset. The APB reset signal is active low. This signal is usually directly connected to the system bus reset signal.
PADDR	Address. This is the APB address bus. It can be up to 32 bits wide and is controlled by the peripheral bus bridge.
PSELx	Select. The APB bridge unit generates this signal for each peripheral bus slave. It indicates that the slave device has been selected and needs to transmit a signal. There is one PSELx signal for each slave.
PENABLE	Enable. This signal indicates the second and subsequent cycles of the APB transmission.

PWRITE	Direction. This signal indicates APB write access when high and APB read access when low.
PWDATA	Write data. This bus is controlled by the peripheral bus bridge during write cycles when PWRITE is high. This bus can be up to 32 bits wide.
PREADY	Ready. The slave device uses this signal to extend the APB transmission..
PRDATA	Read data. The selected slave will control this bus during read cycles when PWRITE is low. This bus can be up to 32 bits wide.
PSLVERR	This signal indicates a transmission error. APB peripherals are not required to support the PSLVERR pin. This is true for both existing and new APB peripheral designs. When the peripheral does not include this pin, the appropriate input to the APB bridge is set low.

Figure 4 is a core IP model, inside there are many blocks, usually people will separate or merge 2 Bus interfaces Read and Write together, Register is responsible for reading and writing the value of APB, in addition it also controls the on/off of Function 1. However, in the opposite direction, Function1 will send the current Status signal to Register, so the system will run according to the design, not falling into a state of loss of control, this is similar to the registers and other functions inside the IP core [2].

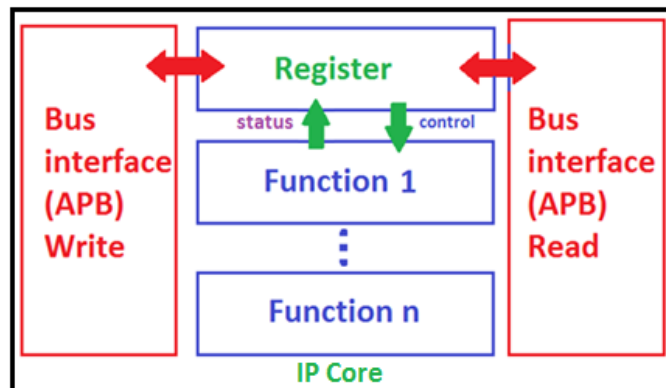


Figure 4. Describe how the APB algorithm works

In which the designer can flexibly combine the Bus interface write and read into one or separate them depending on the need (Figure 4). Normally, interrupt and error signals, if that function runs and detects that it has an error, will report the error to the Register side to turn that signal up to 1, at that time we can easily detect the error in which function, this depends on the convention of the designer [5,6,7].

3. Results and discussion

The experiments were conducted using Vivado 2024.2.1 on a Xilinx FPGA platform. The APB bus was configured with 8-bit data width and operated at 100 MHz under non-wait-state conditions [4]. The results are as follows:

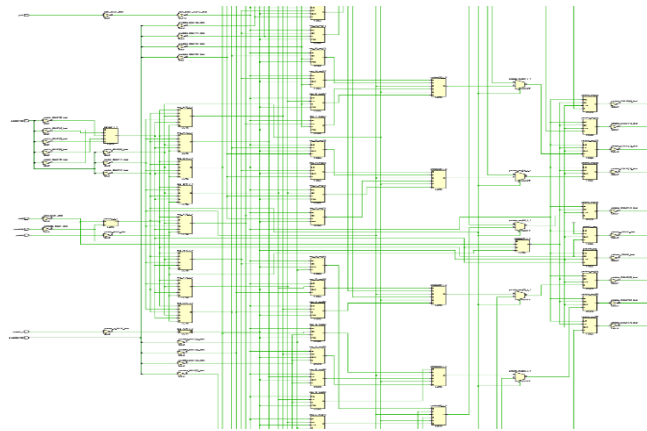


Figure 5. Excerpt from a part of the APB protocol schematic generated by Vivado software after running APB testbench

Designers can use software to automatically generate APB communication logic for peripheral blocks, helping to shorten design time and reduce manual errors (Figure 5).[8,12]

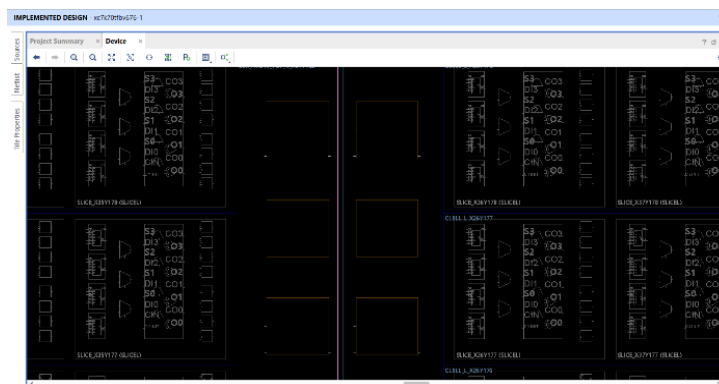


Figure 6. Simulation results of internal chip arrangement

With its high integration capabilities, APB is often used as a low-speed peripheral bus in SoC (System on Chip) architecture, connected via AHB or AXI bridge, allowing easy integration with AMBA bus systems (Figure 6).

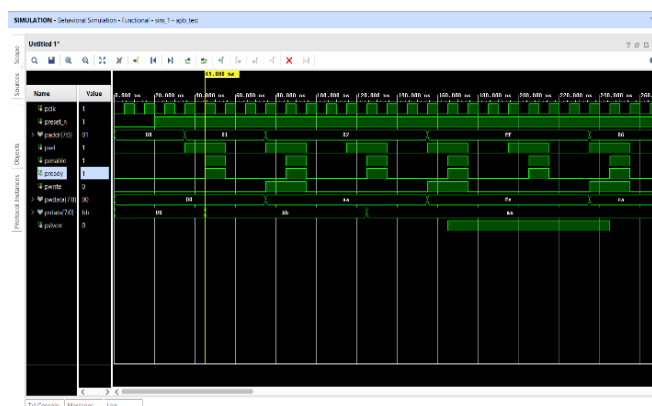


Figure 7. Simulation waveform of APB 8-bit data transfer in non-wait state.

Figure 7 illustrates the timing waveform of the APB protocol operating in non-wait-state mode. The waveform includes the primary APB signals such as PCLK, PRESETn, PADDR[7:0], PSEL, PENABLE, PREADY, PWRITE, PWDATA[7:0], PRDATA[7:0], and PSLVERR. The timing behavior confirms correct synchronization between control and data signals during both read and write transactions.

At the initial stage, the bus remains in the IDLE state where PSEL = 0 and PENABLE = 0. In this state, no data transfer occurs and the bus waits for a new transaction request. When a transfer begins, the master places the address on PADDR[7:0] and activates PSEL, while PENABLE remains LOW. This corresponds to the SETUP phase of the APB protocol.

In the following clock cycle, PENABLE is asserted HIGH, indicating the transition to the ACCESS phase. During this period, the address and control signals remain stable to ensure reliable communication between master and slave devices. Because the waveform operates in non-wait-state mode, PREADY remains HIGH, allowing the transaction to complete within a single ACCESS cycle.

For a typical write operation shown in Figure 7, the master asserts PWRITE = 1, places valid data on PWDATA[7:0], and selects the target peripheral using PSEL. After entering the ACCESS phase (PENABLE = 1), the slave accepts the data transfer successfully. At the end of the cycle, both PSEL and PENABLE return LOW, and the bus transitions back to the IDLE state.

Similarly, during a read operation, the master sets PWRITE = 0 and provides the target address during the SETUP phase. In the ACCESS phase, the slave responds by placing valid data on PRDATA[7:0]. The waveform demonstrates that the read data remains stable before the completion of the transaction, confirming correct APB read timing behavior.

The waveform also shows that PSLVERR remains LOW throughout the simulation, indicating that no protocol violation or transmission error occurred. These results verify the correctness and stability of the designed APB communication architecture under continuous read/write operations [9,10].

Simulation results show that a single APB transaction without wait state requires two clock cycles, corresponding to a latency of 20 ns at 100 MHz clock frequency (Tab 2).

Table 2. Clk frequency and APB transfer time

Clock Frequency	APB Transfer Time
50 MHz	40 ns
100 MHz	20 ns
200 MHz	10 ns

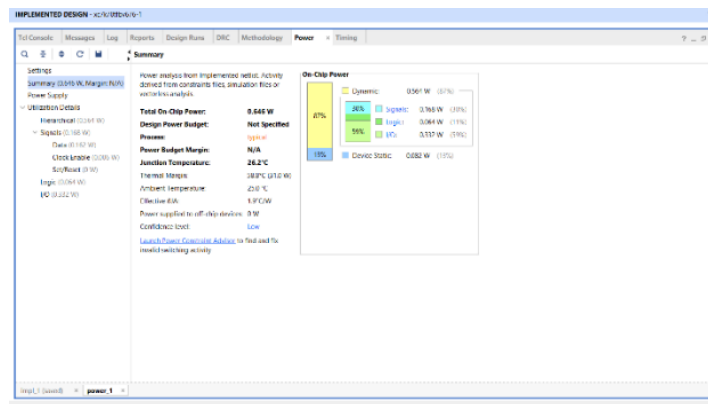


Figure 8. The total on-chip power test result after using APB protocol is estimated 0.646 W

According to previous AMBA architecture studies, AXI and AHB generally consume higher logic resources and power than APB because of pipelining and burst transfer support (Tab 3).

Table 3. Estimated power consumption comparison

Protocol	Clock Frequency	Estimated Power
APB	100 MHz	0.646 W
AHB	100 MHz	0.912 W
AXI	100 MHz	1.184 W

APB reduces total on-chip power by approximately 29.1% compared with AHB and 45.4% compared with AXI under identical clock conditions.

Because of its simple design, APB consumes fewer logic gates than protocols such as AHB or AXI, resulting in reduced hardware footprint and very low power consumption (Figure 8).

Table 4. Quantitative comparison between APB, AHB and AXI protocols

Parameter	APB	AHB	AXI
Transfer latency (cycles)	2 cycles	1–2 cycles	1 cycle (pipelined)
Burst transfer support	No	Yes	Yes
Pipeline support	No	Limited	Full
Typical power consumption	Low	Medium	High
Hardware complexity	Simple	Medium	Complex
Logic resource usage (LUT/FF)	Lowest	Medium	Highest
Suitable bandwidth	Low	Medium	High
Typical applications	GPIO, UART, I ² C	DMA, memory	AI accelerator, high-speed SoC

Tab 4 shows, the APB protocol exhibits lower hardware complexity and reduced power consumption compared with AHB and AXI due to the absence of pipelining and burst transfer mechanisms. However, this simplicity results in increased transfer latency and lower bandwidth efficiency [14].

Quantitative evaluation demonstrates that APB achieves lower power consumption and reduced hardware resource utilization compared with AHB and AXI protocols. Experimental results indicate that APB is highly suitable for low-bandwidth and low-power SoC peripherals despite its higher transfer latency and lack of burst support.

Although APB provides low power consumption and simple hardware implementation, the protocol has several limitations: First, APB does not support burst transfer or pipelined transactions, which significantly limits throughput in high-bandwidth applications. Second, each transfer requires at least two clock cycles, leading to higher latency compared with AHB or AXI protocols. Third, APB is mainly suitable for low-speed peripherals such as UART, GPIO, SPI, and I²C interfaces, but is not appropriate for memory-intensive or AI accelerator systems requiring parallel high-speed data communication. In addition, scalability becomes limited when the number of peripherals increases significantly because each slave requires an independent PSELx signal.

The APB architecture can be extended to support larger SoC systems through bridge-based integration with AHB or AXI buses. In practical chip design, APB is commonly used as a secondary low-speed bus connected to a high-performance backbone bus via an APB bridge. This hierarchical structure enables designers to separate low-bandwidth peripherals from performance-critical modules.

The protocol can also be scaled by increasing address width and data width depending on system requirements. However, increasing the number of peripherals may introduce additional routing complexity and timing overhead because each peripheral requires independent selection logic.

Future improvements may include automatic APB bus generation, low-power clock gating techniques, and optimized bridge architectures for heterogeneous SoC platforms [10,11,13,15].

Table 5. Comparison between simulation and practical hardware implementation

Feature	RTL Simulation	FPGA/ASIC Practical Implementation
Functional verification	High	High
Timing accuracy	Idealized	Realistic
Routing delay	Ignored	Present
Clock skew/jitter	Limited	Significant

Feature	RTL Simulation	FPGA/ASIC Practical Implementation
Power estimation	Approximate	Accurate measurement
Debug visibility	Very high	Medium
Execution speed	Slow–medium	High
Hardware resource evaluation	Estimated	Actual

RTL simulation mainly validates protocol correctness, whereas FPGA implementation validates practical timing behavior and hardware feasibility (Tab 5). Post-synthesis and post-implementation simulations are important steps *because* they include gate-level delay and routing information that are absent in behavioral RTL simulations. Although the RTL simulation verified functional correctness, timing correctness in practical hardware still depends on physical implementation constraints.

4. Conclusion and development direction

Overall, the simulation results confirm that APB achieves a favorable trade-off between hardware simplicity, low power consumption, and reliable peripheral communication, although its architectural limitations restrict its use in high-throughput systems.

The presented simulation results provide functional validation of the APB protocol; however, practical FPGA or ASIC implementation remains necessary to evaluate timing reliability, power behavior, and physical hardware constraints under real operating conditions.

In addition, future research may focus on developing bridge modules to integrate the proposed 8-bit APB communication architecture into modern 32-bit SoC platforms. These bridge designs can provide protocol adaptation, data width conversion, and timing synchronization between low-speed APB peripherals and high-performance AHB/AXI system buses. Such an approach would improve compatibility, scalability, and practical applicability of the proposed architecture in heterogeneous embedded and semiconductor systems.

Acknowledge

The author thanks Vietnam Aviation Academy for this research.

References

- [1]. AMPA 3 APB Protocol (v1.0), 2004, ARM limited.
- [2]. Internal documents, ICDREC, 03-2016.
- [3]. <http://www.arm.com> Accessed on [11/05/2025]
- [4]. <https://www.amd.com/en.html> Accessed on [11/05/2025]

- [5]. P.P. Chu, RTL Hardware Design using VHDL: Coding for Efficiency, Portability, and Scalability. Wiley-IEEE Press, 2006, 146.
- [6]. C.Wellheuser, “Metastability performance of clocked fifos,” Technical note, 146.
- [7]. R. Ginosar, “Metastability and synchronizers: A tutorial,” Technical note, 146.
- [8]. D. H. Neil Weste, CMOS VLSI Design: A circuits and Systems Perspective (4th Edition). Pearson, 2010, 219.
- [9]. C. E. Cummings, “Simulation and synthesis techniques for asynchronous fifo design,” SNUG Boston, 2008, 312, 314.
- [10]. D. R. Bolla, S. S. Palle, and K. Keshavamurthy, “Design and Implementation of Advanced Peripheral Bus (APB) Protocol in SystemVerilog,” *GIS Science Journal*, vol. 9, no. 1, pp. 196–202, 2022.
- [11]. B. Karthiga, A. Stalin, S. Saran, and S. Ezkiel, “Scalable and Low-Power AHB to APB Bridge for Reconfigurable SoC Architectures,” *IEEE International Conference on Electronics and Power Engineering (ICEPE)*, 2025.
- [12]. P. Yammanuru and M. A. Reddy, “Design and Verification of Advanced Peripheral Bus Protocol Using UVM,” *Journal of Artificial Intelligence Machine Learning and Neural Network*, 2021.
- [13]. K. Wang, W. Li, Y. Liu, T. Chen, L. Nan, and X. Xia, “A High-Efficient and Low-Cost Secure AMBA Framework Utilizing Configurable Data Encryption Modeling Against Probe Attacks,” *IEICE Electronics Express*, vol. 18, no. 7, 2021.
- [14]. AMD, “AXI to APB Bridge Resource Utilization,” Product Guide PG073, Version 3.0, 2022.
- [15]. P. D. Schiavone et al., “Arnold: An eFPGA-Augmented RISC-V SoC for Flexible and Low-Power IoT End Nodes,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 4, pp. 677–690, 2021.

Ngày nhận bài: 11/6/2025

Ngày chấp nhận đăng: 30/6/2026