ISSN 2959-6157

Design and Implementation of a Synchronous AHB to APB Bridge in System-on-Chip Applications

Hongjin Chen

Glasgow College, University of Electronic Science and Technology of China, Chengdu, China
Corresponding author: 2720611@student.gla.ac.uk

Abstract:

The requirement of effectively combining key units in an integrated system is proliferating. SoC system is developed to provide a chip-level integration, which become the inevitable trend of integrated circuit development and is widely used in smartphones, industrial applications, and microcontrollers. The ARM AMBA protocol works as a universally adopted way of interaction between various parts of the system. In AMBA architecture, the AHB to APB bridge significantly contributes to combining the high-performance AHB bus and low-power APB bus in the SoC system. This project is to implement an AHB to APB bridge using Verilog, enabling stable data transfer between these two buses. The proposed AHB to APB bridge is intended to fit different read or write strategies and ensure the proper working of the peripherals on the APB bus. The bridge has been implemented via Verilog Hardware description language (HDL). A test bench was created with a virtual AHB host and an optimized SRAM as the high-speed APB peripheral. Verdi simulation shows the bridge completely meets the design intention.

Keywords: AHB to APB Bridge; System-on-chip (SoC); AMBA Protocol.

1. Introduction

As the demand for miniaturization in electronic devices increases, SoC (System on Chip) systems are employed to provide higher integration, resulting in lower power consumption, reduced costs, and enhanced reliability. In an SoC, the quality of interaction and communication between functional blocks is often fundamental to the proper operation of the system [1]. The Advanced Microcontroller Bus Architecture (AMBA) is an open standard provided by ARM to connect and manage functional blocks in an SoC system [2]. It facilitates the right-first-time development of multiprocessor designs, with large numbers of controllers and peripherals. This protocol has persisted as a universally adopted open standard for nearly 30 years. Up to AMBA5, the AMBA protocol has mainly encompassed ACE, CHI, AXI, AHB, APB, ATB, CXS, ATP, DTI, LTI, and LPI [3]. Among these, the AHB and APB stand out as two widely used and efficient buses, serving as the backbone buses for basic high-speed and low-speed operations, respectively.

The AMBA Advanced High Performance Bus (AHB) supports multiple bus masters and high bandwidths operations [4]. It is the most widely used interface protocol for ARM Cortex-M Processors and often serves as the primary bus in SoCs, which helps connect processors, caches, DMA controllers, and other high-frequency, high-performance components [3]. Another famous interface protocol that often works with AHB is APB, the AMBA Advanced Peripheral Bus (APB) is a low-bandwidth bus, which features a compact size and low power consumption [5]. It typically provides connections to low-speed communication protocols such as UART, SPI, IIC, and other low-speed, low-power IO devices.

The synergy between AHB and APB allows the system to maintain high speed and performance while reducing power consumption and complexity of peripheral devices and IOs, ultimately lowering system costs. Given this context, a channel connecting the AHB and APB buses becomes an essential component for ensuring stable and efficient system operation.

The AHB to APB bridge serves as a transfer between AHB and APB buses shown in Fig.1, realizing data exchange between the two buses and enabling connections between AHB master devices and APB slave devices. Its primary function is to expand multiple low-level ports to help optimize the overall performance and power consumption of the SoC system. The AHB to APB bridge works as a slave device in the AHB bus. However, the bridge acts as the sole master in the APB bus, responsible for activating and selecting APB sub-devices, and managing data transfers.

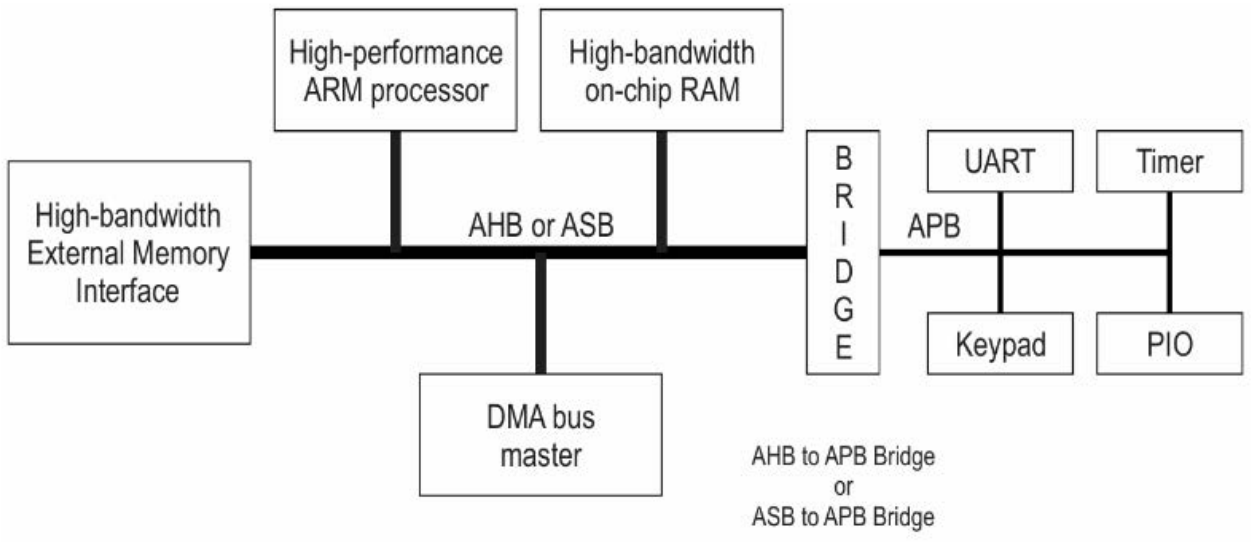

Fig. 1 AMBA Architectural Plan [3]

There are two types of bridges: synchronous and asynchronous. The APB bus clock is controlled by the PCLKEN signal, leading the synchronous bridges operating with the same clock source and aligned phase. This character allows synchronous bridges to change operating frequencies for various APB sub-devices. In asynchronous scenarios, the clock of the APB bus is not controlled by the AHB devices, which means it is easy to cause a metastable state due to phase unaligned. Therefore, the asynchronous bridges require additional synchronization across clock domains.

2. Literature Survey

The concept of the AHB to APB bridge was introduced alongside the APB bus in the AMBA2 protocol. Over the years, extensive research has been conducted in this field, including various power optimization designs, diverse verification methodologies, and designs for connecting other high-speed buses to the APB bus. Additionally, updates to the design have been made in response to iterations of the AMBA protocol. In paper [1] a design and implementation of a standard AHB to APB bridge that allows integration of different buses in a SoC. It provides a performance analysis under various clock configurations by synthesis and simulation with System Verilog and Modelsim. The author [6] realized FPGA implementation by checking operation results with the chipscope tool. Compared with software implementation, this method allows easy testing and optimization through error testing and debugging. The paper [7] discusses the way of using UVM (Universal Verification Methodology) to enhance the efficiency and effect of the AHB to APB bridge implementation. The simulation results demonstrated the effectiveness of the bridge and obtained 100% function coverage. The author [8] designed an IP (Intellectual Property) core of APB bridge, which extends the bridge support to AXI bus, and facilitates the bridge to support other AMBA high-performance buses. The paper [9] focuses on layout design to reduce the power consumption and area usage of the AHB to APB bridge in SoC. By using Cadence SoC Encounter in TSMC 180 nm technology, the designed asynchronous bridge under APB3 specification saves 6% power and 10% area over the one with APB2 specification.

The difference between synchronous and asynchronous AHB to APB bridges lies in whether the clock domains are consistent. Synchronous bridges can typically be achieved using FSM (finite state machines) and registers. They allow the APB bus to have variable frequencies for different endpoint devices, which offer high performance, flexibility, and module isolation [10]. Asynchronous bridges typically use asynchronous FIFO or handshake signalling methods to realize information transfer between different clock domains. These approaches enhance system compatibility and reduce data loss or errors during transmission [1].

3. Methodology and Technical Foundations

3.1 Interface Overview

In the design of the bridge, careful consideration must first be given to its inputs and outputs. As illustrated in Fig.2, the bridge acts as a slave to the AHB bus, necessitating the inclusion of all typical inputs expected from an AHB slave device: data, address, clock, control signals, and others. Simultaneously serving as a master to the APB bus, its inputs encompass read data from APB during read

processes, along with fundamental Reset signals.

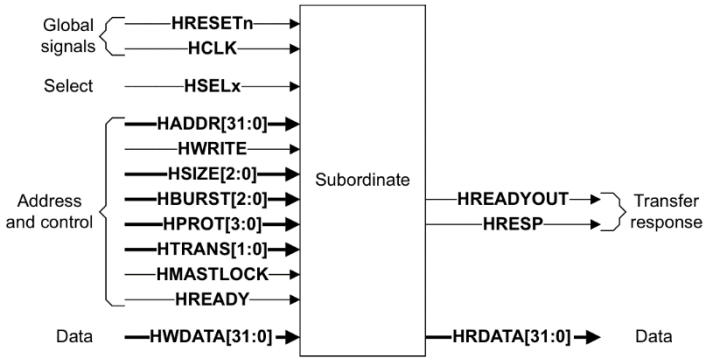

Fig. 2 AHB Signal Interface of an AHB Slave [3]

Fig.3 indicates the output interfaces of the bridge. Its outputs primarily consist of signals essential for APB bus read or write operations: chip select, data, address, control, and clock signals, all directly connected to the APB bus for slave device usage. Since the AHB to APB bridge remains an AHB slave device, it must also generate feedback signals on the AHB bus, such as HREADYOUT and HRESP, crucial for AHB bus masters to monitor the current transfer status of the bridge and handle special states like Busy or Error.

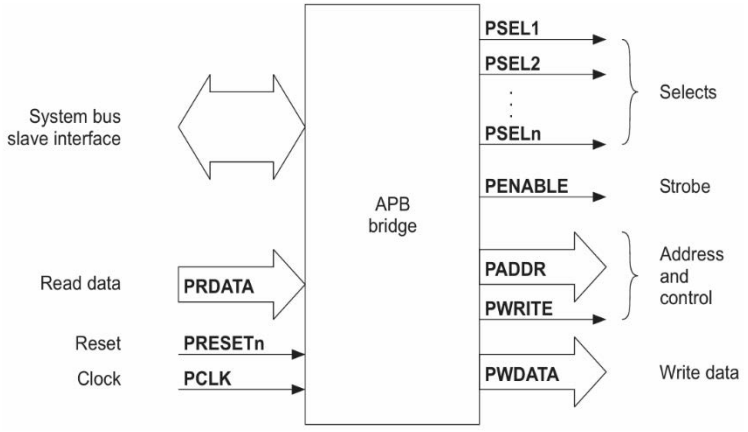

Fig. 3 APB Signal Interface of an APB Bridge [5]

3.2 State Machine

The state machine of the AHB to APB bridge consists of seven states: ST_IDLE, ST_APB_TRNF, ST_APB_TRNF2, ST_APB_WAIT, ST_APB_ENDOK, ST_APB_ERR1, and ST_APB_ERR2. As depicted in Fig.4, transitions between these states are indicated by green lines for normal transitions, yellow lines for special cases preventing deadlock, and red lines for error states.

1) ST_IDLE: it is a state where the bridge waits for signals, effectively representing an idle state. When the APB clock's rising edge occurs (PCLKEN active) and apb_select is high, meeting the conditions for read or non-buffered write operations, the state transitions to ST_APB_TRNF; otherwise, it remains in the IDLE state. If a buffered write operation is required (apb_select high), the state transitions to ST_APB_WAIT; otherwise, it continues to maintain the IDLE state.

2) ST_APB_WAIT: in this state, it waits for the next APB clock and is responsible for holding data for one clock before outputting it. Upon arrival of the next APB clock, the state transitions to ST_APB_TRNF to complete the buffering task; otherwise, it continues to remain in the WAIT state.

3) ST_APB_TRNF: This state establishes the transfer state. If an APB clock arrives, the state transitions to ST_APB_TRNF2; otherwise, it continues to maintain the TRNF state.

4) ST_APB_TRNF2: it is the transfer state that continues until the transfer is completed. When triggered by a clock (APB clock), if the PSLVEER error signal is low and PREADY transfer completion signal is high, indicating data needs buffering for one cycle, the state transitions to ST_APB_ENDOK; otherwise, it continues to maintain the

TRNF2 state. If no buffering is needed and apb_select is high, indicating subsequent transfers, the state transitions to ST_APB_WAIT. If no buffering is needed and apb_select is low, indicating no subsequent transfers, the state transitions to ST_IDLE. When triggered by a clock (APB clock), if PREADY is high and PSLVEER is high, indicating an error condition, the state transitions to ST_APB_ERR1; otherwise, it continues to maintain the TRNF2 state.

5) ST_APB_ENDOK: the end state indicating completion of the current transfer task. Upon the rising edge of the next APB clock (PCLKEN active) and apb_select is high, meeting the conditions for read or non-buffered write operations, the state transitions to ST_APB_TRNF; otherwise, it transitions to ST_IDLE. If a buffered write operation is required (apb_select high), the state transitions to ST_APB_WAIT; otherwise, it transitions to ST_IDLE. This state behaves similarly to ST_IDLE, differing only when no conditions are met upon arrival of the next APB clock, in which case it transitions back to ST_IDLE to await the same detection conditions.

6) ST_APB_ERR1: an error state allowing the AHB master to decide whether to cancel or continue the transfer. Upon the arrival of the next AHB clock, it directly transitions to ST_APB_ERR2.

7) ST_APB_ERR2: an error state where subsequent state transitions depend on the handling decision made by the AHB master (reflected in bus changes). Upon the rising edge of the next APB clock (PCLKEN active) and apb_select is high, meeting the conditions for read or non-buffered write operations, the state transitions to ST_APB_TRNF; otherwise, it transitions to ST_IDLE. If a buffered write operation is required (apb_select high), the state transitions to ST_APB_WAIT; otherwise, it transitions to ST_IDLE. The subsequent state transition behavior of this state is identical to that of ST_APB_ENDOK.

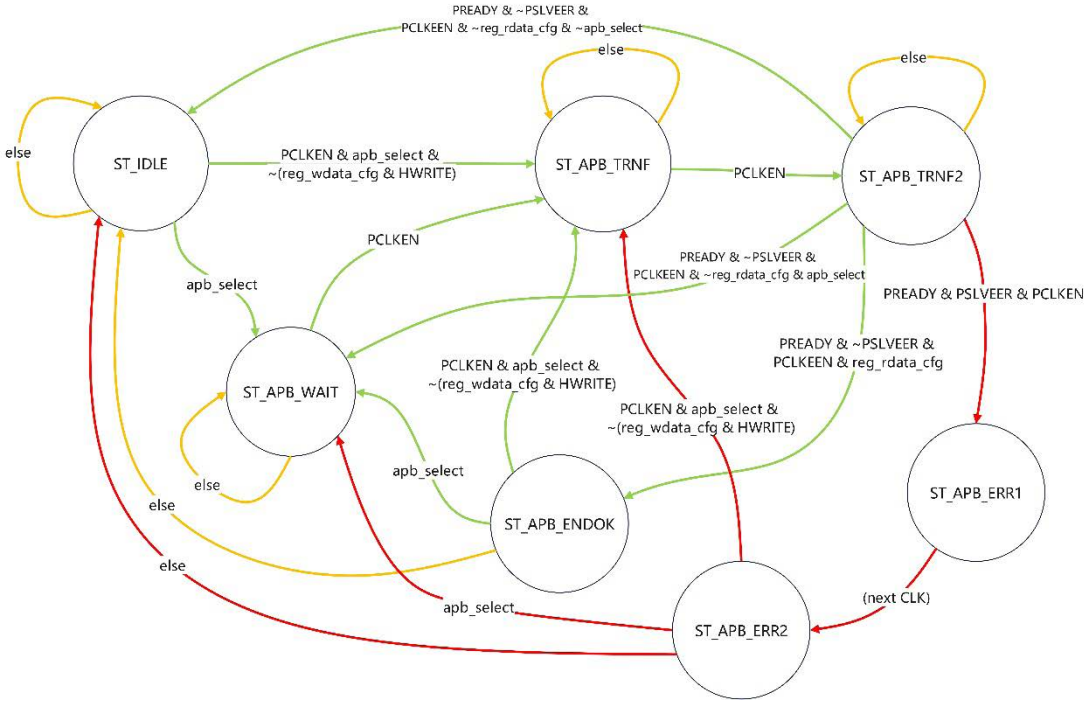

Fig. 4 State Machine of the Bridge

(Picture credit: Original)

4. Application Scenarios for AHB to APB Bridge

ST-Microelectronics is one of the top 10 semiconductor companies, which is famous for its MCU, sensors, RF & wireless chips. There 32-bit MCU STM32 microcontrollers are widely used in various embedded systems and electric devices, such as PLC (Programmable Logic Controller), smart outlets, vehicle collision warning systems, and heart rate monitors. As STM32 microcontrollers are based on ARM Cortex-M series processors, AHB buses work as the primary bus in the SoC system.

In STM32 microcontroller, it has a 32-bit bus matrix to handle access and efficient operation when high-speed peripherals work simultaneously. However, this bus matrix is a multi-layer AHB bus matrix, means we could roughly consider it as a huge AHB bus that data can be transferred to all endpoints since the AHB to APB bridge uploads data. STM32 microcontrollers usually have 2 configurable AHB to APB bridges, its frequency and endpoint IO ports are changeable due to configuration [11].

As the official diagram shown in Fig.5, the AHB to APB bridge take all the APB endpoints, responsible for analogue-digital signal conversion, serial port communication, and PWM generation. After receiving data, command signal or address signal from the AHB system bus, the AHB to APB bridge select and sends relative information to peripheral endpoints. When uploaded, the AHB to APB bridge load the information to the AHB bus. The bridge here isolates the peripheral endpoints from the high-performance AHB bus matrix, providing the MCU low-power/low-voltage operation, connectivity, and real-time capabilities, while maintaining full integration and ease of development [11]. Therefore, the success of the STM32 MCU is inseparable from the outstanding contribution of AHB to APB Bridge.

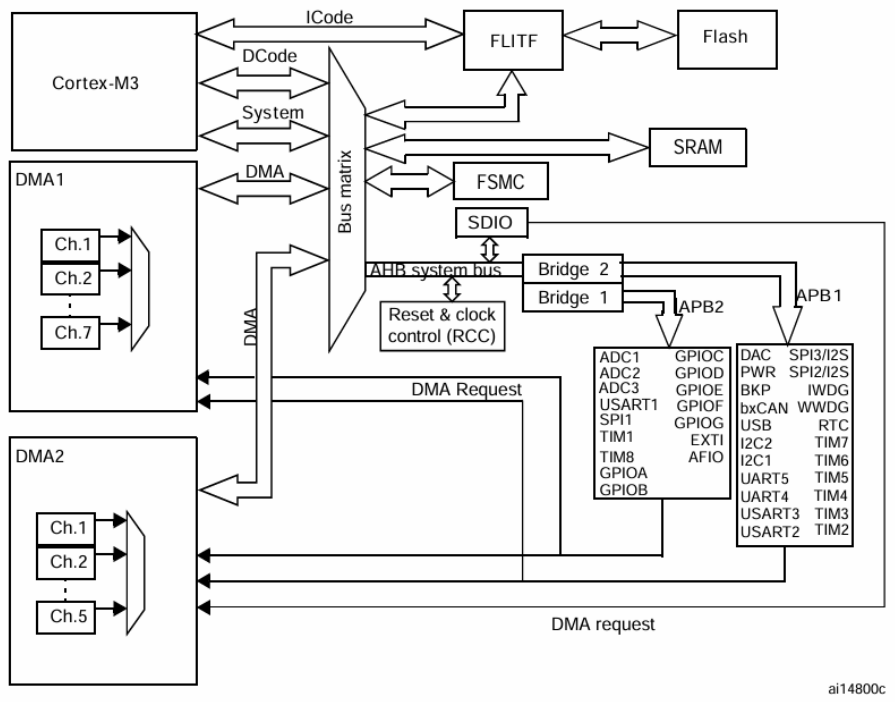

Fig. 5 System Architecture of the STM32F1 Family [11]

5. Validation Design and Simulation Results

The design and the validation are done with Verilog and System Verilog. A virtual AHB master is built to generate all correct functional signals and random transmission data. The APB slave here is an optimized SRAM used for read and write operations. SRAM was optimized to a timing sequential read/write state, and the data input and output were stored until the next clock. Consequently, the data always needs to wait for the next clock in both input and output operations. The designed AHB2APB bridge is a universal configurable bridge that supports four operational modes: direct read, direct write, buffered read, and buffered write. By configuring REGISTER_RDATA or REGISTER_WDATA, different read/write strategies can be implemented in circuits. Result Waveforms are drawn by WaveDrom editor, with simulation data by Verdi. Only the two most complicated operation modes, buffered read and buffered write results with key parameters are shown in the diagram.

5.1 Read Process in Buffered Mode

As illustrated in Fig.6, the read task occurs when HWRITE signal is low. Following the output of HADDR to the address, it passes through the bridge into the APB bus. After the state machine jumps to 3 and adding another clock, the first data is delivered, subsequently stored in PRDATA upon the arrival of the next APB clock. After another APB clock cycle, PRDATA enters the AHB to APB bridge and is held in the bridge's read/write registers. Simultaneously, the data is directly assigned to HRDATA, facilitating its entry into the AHB bus. The entire system operates read operations in a manner of address assignment, data retrieval, internal storage within the bridge, and mounting onto the AHB bus. Simulation results comprehensively demonstrate the complete process from the initiation of a read operation to the data entering the AHB bus, without errors in each data transfer cycle.

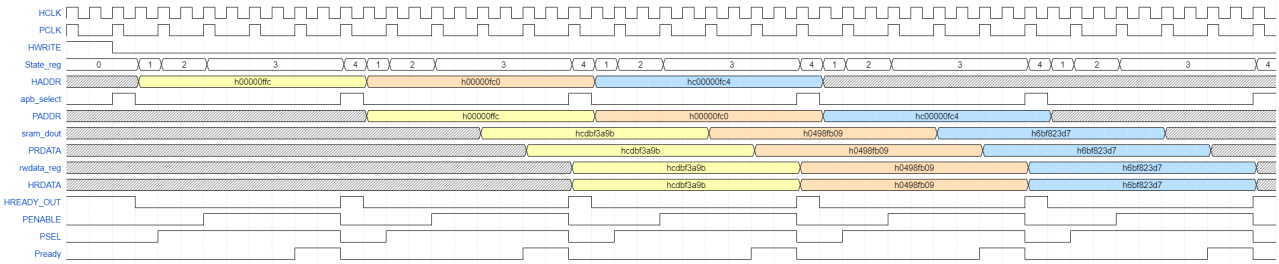

Fig. 6 Read Process Validation Waveform Diagram

(Picture credit: Original)

In addition, typically, the Pready signal should rise as PENABLE rises. However, SRAM here is optimized to wait for the clock, which leads SRAM to output data one clock after PENABLE is raised. Buffered output caused the Pready signal rise one clock after SRAM output, which means the Pready here is two clocks later than typical conditions. The optimized system ensures that the APB bus can support high speed operation of the whole system without issues.

5.2 Write Process in Buffered Mode

As depicted in the Fig.7, the bridge operates in write mode when the HWRITE signal is high. Initially, HADDR provides the target SRAM address, followed by the data at the subsequent address. The data enters the AHB2APB bridge and is stored in its read/write registers upon the arrival of the next APB clock cycle. After another APB clock arrives, the data enters the APB bus and is received by the SRAM. In SRAM, the address needs to follow 32bits alignment, which must remove the last 2 bits of the address received, leading the address of the SRAM operational is not equal to the one AHB device provided. The entire system executes read operations in manner of address assignment, data retrieval, internal storage within the bridge, and mounting onto the APB bus. Simulation results comprehensively illustrate the entire process from the initiation of a write operation to the data entering the APB bus, without errors in each data transfer cycle. This ensures accurate communication from the high-speed AHB bus to the low-speed APB bus.

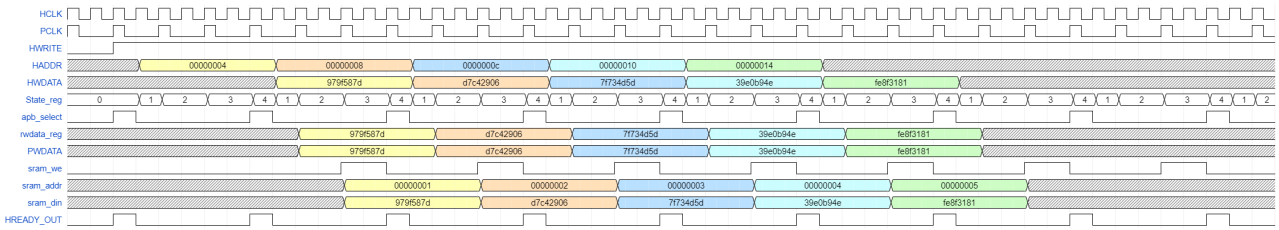

Fig. 7 Write Process Validation Waveform Diagram

(Picture credit: Original)

6. Conclusion

The AHB to APB bridge is widely used in high-performance SoC systems. In this paper, to give an architecture overview, a theoretical AHB to APB interface and a designed state machine is shown as the foundation of the design. An example of the architecture of the STM32F family is shown to explain how the bridge is utilized in the real SoC system. The designed AHB to APB bridge has been verified by using suitable test benches. Waveform has validated its function of data transmission between AHB bus and APB bus and demonstrated the operation of the AHB to APB bridge required in AMBA protocol. The bridge provides effective communication means for system stability, and it is a universal configurable bridge that supports different read/write strategies in circuits. This adaptability allows the bridge to increase pipelining structures in high-delay scenarios or perform direct read/write operations in low-delay situations, thereby enhancing the overall performance of the SoC system.

References

[1] RM, Vani and Roopa, M. Design of AMBA based AHB2APB bridge. IJCSNS, 2010, 10(11): 14.

[2] Flynn, D. AMBA: enabling reusable on-chip designs. IEEE Micro, 1997, 17(4): 20-27.

[3] Arm Ltd. AMBA Specifications. ARM IHI0011a, 2002. Available: https://developer.arm.com/documentation/ihi0011/a/.

[4] Arm Ltd. AMBA AHB Protocol Specification. ARM IHI 0033C, 2021. Available: https://developer.arm.com/documentation/ihi0033/latest/.

[5] Arm Ltd. AMBA APB Protocol Specification. ARM IHI 0024E, 2023. Available: https://developer.arm.com/documentation/ihi0024/latest/

[6] Aithal, Sowmya and Baligar, JS and Guruprasad, SP. FPGA

implementation of AHB to APB protocol. International Journal Of Engineering And Computer Science, 2016, 5(5): 16617-16619.

[7] Kiran, Ankem and Thrimurthulu, V. Verification of AMBA AHB2APB bridge using universal verification methodology (UVM). International Journal in IT \& Engineering, 2016, 4(12): 1-10.

[8] Ma, Chenghai and Liu, Zhijun and Ma, Xiaoyue. Design and implementation of APB bridge based on AMBA 4.0. 2011 International Conference on Consumer Electronics, Communications and Networks (CECNet), 2011, 193-196.

[9] Paunikar, Abhijeet and Gavankar, Rohan and Umarikar, Nachiket and Sivasankaran, K. Design and implementation of area efficient, low power AMBA-APB Bridge for SoC. 2014 International Conference on Green Computing Communication and Electrical Engineering (ICGCCEE), 2014, 1-6.

[10] Dubois, M. and Savaria, Y. and Bois, G. A generic AHB bus for implementing high-speed locally synchronous islands. Proceedings. IEEE SoutheastCon, 2005, 11-16.

[11] STMicroelectronics. STM32F101xx, STM32F102xx, STM32F103xx, STM32F105xx and STM32F107xx advanced Arm®-based 32-bit MCUs. RM0008 Reference manual, 2021. Available: https://www.st.com/resource/en/reference_manual/rm0008-stm32f101xx-stm32f102xx-stm32f103xx-stm32f105xx-and-stm32f107xx-advanced-armbased-32bit-mcus-stmicroelectronics.pdf.