Design and Verification of AMBA APB Protocol using System Verilog

Abhinandan K1, Basavaraj A S2, Danesh S A3, Likhith G K4

Dr. Praveen J5

Student, GM University1,2,3,4, HOD, Dept of ECE, GM University5

Department of Electronics and Communication, G M University, Davangere 577 001

**ABSTRACT**

The Advanced Microcontroller Bus Architecture (AMBA), Advanced Peripheral Bus (APB) protocol stands as a cornerstone in modern System-on-Chip (SoC) designs, serving as the vital link between peripheral components and the central processing unit (CPU). Crafting this interface demands meticulous attention to detail, encompassing the development of an AMBA APB interface that strictly adheres to the protocol's exacting standards. This entails designing and integrating the APB controller, the APB bus interface, and various peripheral modules. Leveraging System Verilog facilitates a modular and adaptable design approach, accommodating diverse data widths, address spaces, and peripheral requisites. AMBA APB protocol's specifications. The verification environment, constructed with System Verilog, meticulously evaluates the design's capacity for data transfers (read/write), proficient handling of bus arbitration, and adept management of wait states.

***Keywords: AMBA, APB, SOC***

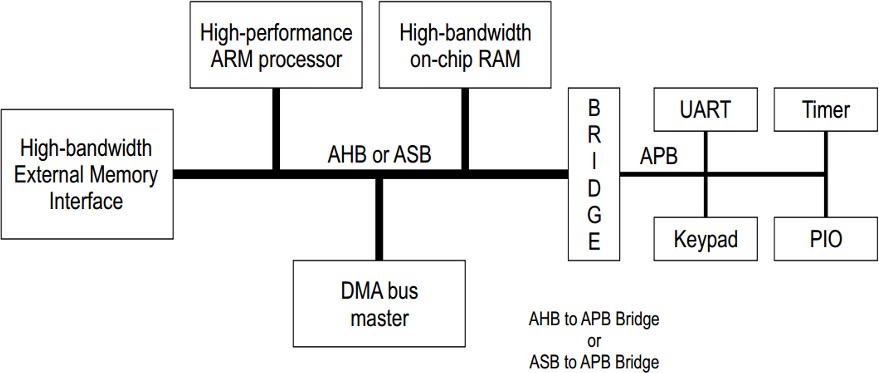
# INTRODUCTION

In the domain of digital system design, industry standards are crucial for guaranteeing the compatibility, dependability, and seamless operation of integrated circuits, microcontrollers, and System-on-Chip (SoC) designs. The advanced peripheral bus protocol, a vital element of ARM's AMBA, exemplifies the importance of these standards. AMBA APB has emerged as the preferred option for crafting interfaces between microcontroller peripherals and the central processing unit (CPU) in ARM-based systems, and its remarkable balance of simplicity and effectiveness.

The design and verification of an AMBA APB interface is a process that encompasses two pivotal phases. In the design phase, the goal is to create an AMBA APB interface that adheres to the stringent protocol requirements. This phase involves the design and integration of various components, including the APB controller, the APB bus interface, and a host of peripheral modules that are designed to serve as bridges between the digital world and the physical environment. This process requires a keen eye for detail and a deep understanding of the AMBA APB protocol to ensure that the designed interface aligns with the protocol's specifications.an industry standard for ASIC design for portable applications. Creating and applying powerful, portable and at the same time re-usable intellectual Property (IP), capable of enhancing an ARM core.

## Description Of AMBA APB Bus.

## The Advanced Microcontroller Bus Architecture (AMBA) comprises a suite of interconnect specifications developed by ARM (Advanced RISC Machines) to facilitate the creation of high performance, low-power systems-on-chip (SoC). Among its essential elements, the Advanced Peripheral Bus (APB) stands out, designed to furnish a cost-effective, energy-efficient interface for linking peripherals to the system. AMBA APB operates on a straightforward, low-frequency bus



**Fig. 1** A typical AMBA architecture.

protocol, commonly utilized for interfacing various low-bandwidth peripherals like timers, interrupt controllers, and supplementary components to the primary system bus. Specifically crafted for applications where top-tier performance isn't imperative, APB proves ideal for connecting slower peripherals.

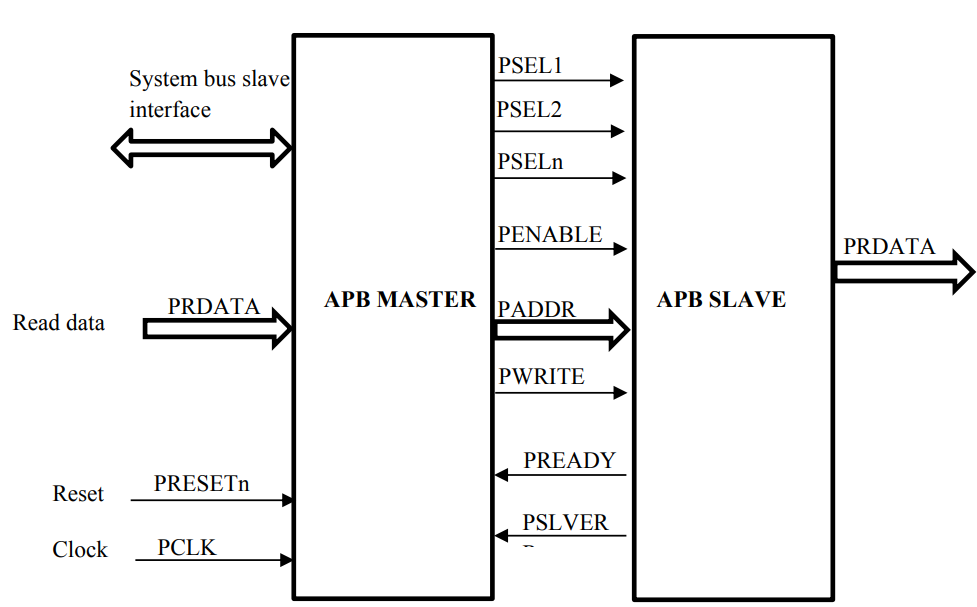
AMBA APB transfer can commence the bus, master must be granted access to the bus. This process is started by the master. AMBA encompasses a collection of interconnect protocols facilitating communication among diverse components within a computer system, including processors, peripherals, and memory. Illustrated in the diagram are an advanced ARM processor and high-bandwidth on-chip RAM, with a central bridge connecting the processor to the external memory interface. Various bridges exist, tailored to different bus types, such as the AHB to APB Bridge or ASB to APB Bridge.

The Advanced High-performance Bus is engineered for swift, high-bandwidth transfers between masters and slaves, while the Advanced System Bus serves as a high-performance bus primarily utilized for memory system interfaces. Contrarily, the Advanced Peripheral Bus caters to simpler peripherals not necessitating high-performance transfers and emphasizes low-power. asserting a request signal to the arbiter. Then the arbiter indicates when the master will be granted use of the bus.

A granted bus master starts an AMBA APB transfer by driving the address and control signals. These signals provide information on the address, direction, and width of the transfer, as well as an indication if the transfer forms part of a burst. Two different forms of burst transfers are allowed: incrementing bursts, which do not wrap at address boundaries; and wrapping bursts, which wrap at address boundaries. A write data bus is used to move data from the master to a slave, while a read data bus is used to move data from a slave to the master.

## Communication of Master to Slave

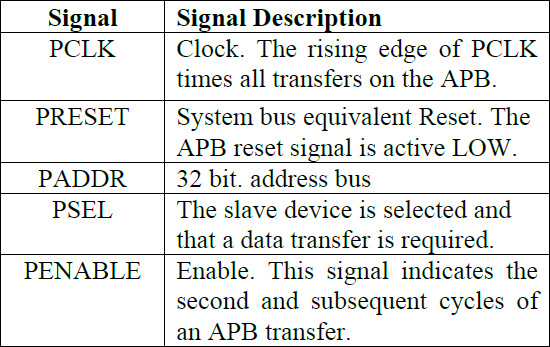
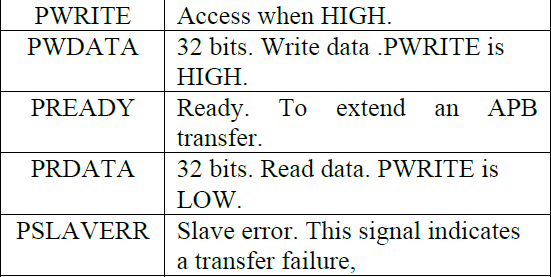
The APB slave module receives PSEL, PENABLE, PWRITE (Address and Control), PRESETn, and PCLK as input signals. PADDR and PWDATA serve as 32-bit input control signals from the bridge and furnish 32 bits of PRDATA as output. All signals within the AMBA APB architecture are denoted with a single letter P prefix. Certain signals within the APB, such as the clock, may be directly linked to the corresponding signals within the system bus. The APB Slave module boasts a straightforward yet adaptable interface, facilitating its utilization across various slave interfaces. It executes the subsequent functions effectively. The interface of APB slaves is characterized by its simplicity and flexibility. The specific configuration of this interface will vary based on the chosen design approach, allowing for numerous potential options. Two crucial signals in this interface, PSLVERR and PREADY, primarily ensure data integrity during data transfer processes.



**Fig 2:** Interfacing of APB Master & Slave.

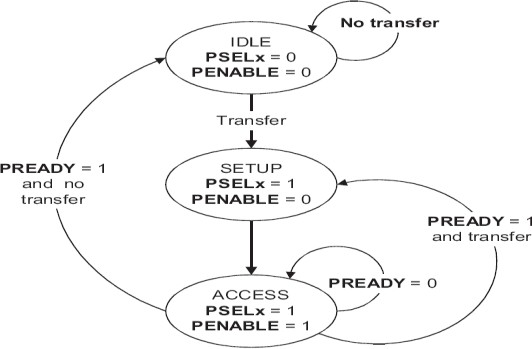
These signals facilitate communication between master and slave devices. To initiate a data transfer, the master places the address and data (for a write transfer) on the bus while asserting the necessary control signals. Upon recognizing its address on the PADDR bus, the slave device responds by asserting the PREADY signal, indicating readiness to receive data. Once the data is received, the slave asserts the PSLVER signal to confirm the validity of data on the PRDATA bus. Subsequently, the master retrieves the data from the PRDATA bus. All signals within the AMBA APB architecture are denoted with a single letter P prefix. Certain signals within the APB, such as the clock, may be directly linked to the corresponding signals within the system bus.

1. **DESCRPTION OF SIGNALS**



**Table 1.**  APB signals

Idle is the normal state of the APB. When a Transfer is necessary the bus relocates into the Setup state, where the suitable select signal, Pselx, is asserted.

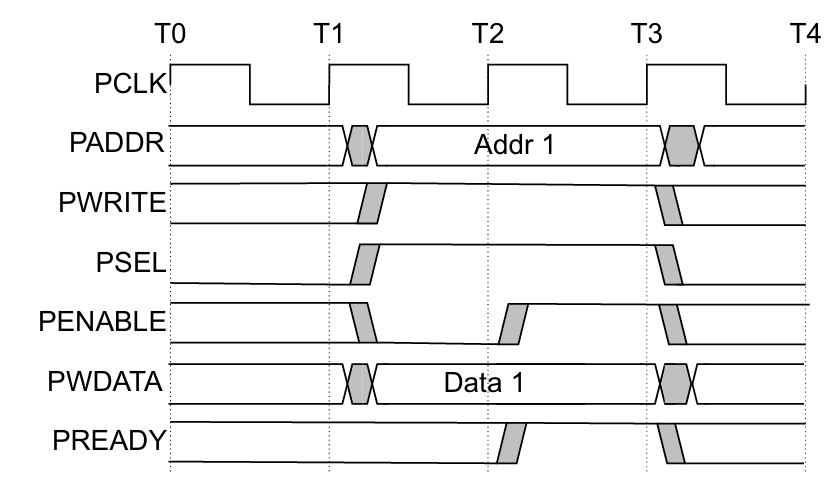


**Fig 3:** Diagram depicting States

The bus only waits in the SETUP state for one clock cycle and always moves to the ACCESS state on the next rising edge of the clock. ACCESS will enable signal, PENABLE, is asserted in the ACCESS state. The write, write data signals, select, and address must remain stable during the transition from the SETUP to ACCESS state. ACCESS state is controls when to exit by the PREADY signal from the slave. These are the conditions one is if PREADY. is held LOW by the slave then the peripheral bus remains in the ACCESS state another is 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 after that it will start the same cycle. Conversely, if the Completer sets PREADY to a HIGH state, the ACCESS state concludes. If further transfers are unnecessary, the bus reverts to the IDLE state. Alternatively, if another transfer is imminent, the bus transitions directly to the SETUP state. The ACCESS state termination is regulated by the PREADY signal originating from the Completer.

1. **TRANSFERS**

**WRITE Transfers**

****

**Fig 4:** Transfer without wait states

T1: The master asserts all the necessary signals to initiate the write transfer. These signals include:

PCLK: System clock signal.

PADDR: Address bus that specifies the memory location where the data needs to be written.

PWRITE: Write control signal (active high).

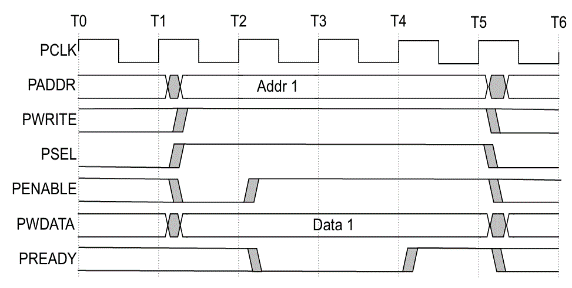
PSEL: Slave select signal to identify the specific slave device involved in the transfer.

PENABLE: Enable signal for the data bus.

PWDATA: Write data bus that carries the data to be written (Data 1).

T2: The slave asserts the PREADY signal. This indicates to the master that the slave is ready to accept the data on the PWDATA bus. Since PREADY is asserted in the same clock cycle (T2) as the other control signals by the master, there are no wait states inserted by the slave device in this transfer.

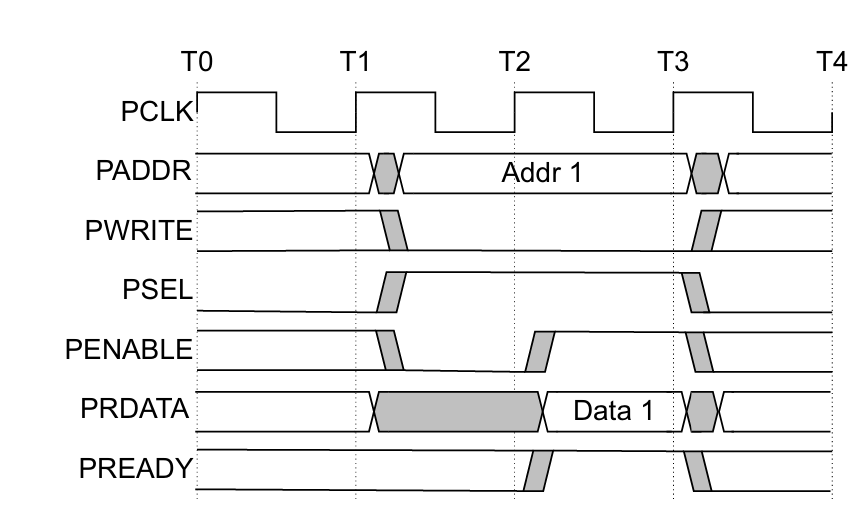
T3: The write transfer is complete. The master deasserts PENABLE and PSEL signals.



**Fig 5:** Transfer with wait states

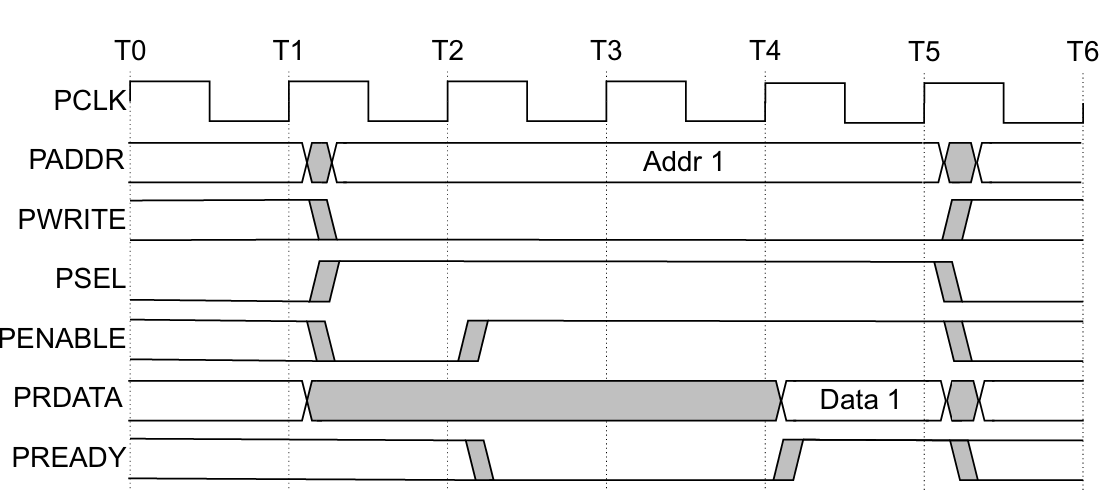
T0: The master initiates communication by activating the PCLK, PADDR, PWRITE (set to HIGH for a write), PSEL, and PENABLE signals. This signifies the intent to write Data 1 to a specified address, Addr 1. T1: The slave device sets the PREADY signal low, transitioning into a waiting state, indicating to the master its unreadiness to receive data presently. Despite this, the master maintains control over all other signals, including PCLK, PADDR, PWRITE, PSEL, and PENABLE. T2-T5: These intervals represent wait states introduced by the slave device. The slave keeps PREADY low, indicating its unreadiness. T6: The slave reasserts the PREADY signal, signifying its readiness to accept the data. During the wait states (T1 to T6), the master continues to drive the PCLK, PADDR, PWRITE, PSEL, PENABLE, and PWDATA (Data 1) signals.

**READ Transfers**

****

**Fig 6:** Transfer without any wait states

T0: The master asserts all the necessary signals to initiate the read transfer. These signals include: PCLK: System clock signal PADDR: Address bus that specifies the memory location from where the data needs to be read (Addr 1). PWRITE: Write control signal (active LOW for a read transfer). PSEL: Slave select signal to identify the specific slave device involved in the transfer. PENABLE: Enable signal for the data bus. T1: The slave activates the PREADY signal simultaneously with transmitting Data 1 on the PRDATA bus. This signals to the master that the slave is prepared to deliver the data. As PREADY is asserted concurrently with the other control signals by the master in the same clock cycle (T1), there are no wait states introduced by the slave device in this transfer. T2: Upon completion of the read transfer, the master retrieves Data 1 from the PRDATA bus and removes the assertions of the PENABLE and PSEL signals.



**Fig 7:** Transfer with wait states

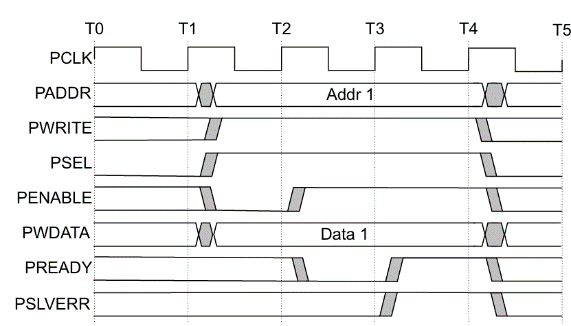
T0: The master asserts the PCLK, PADDR, PWRITE (which is LOW for a read), PSEL, and PENABLE signals. This indicates to the slave device that the master is requesting to read data from a specific address. T1: The slave device asserts the PREADY signal to indicate that it is ready to accept the request. It then deasserts PREADY to enter a wait state. T2-T3: These are wait states inserted by the slave device. The slave is not ready to provide the data yet, possibly due to some internal operations it needs to complete first. It keeps PREADY low to signal this to the master. The master continues to drive the address, PWRITE, PSEL and PENABLE signals during this time. T4: The slave asserts PREADY again to indicate that it has the data ready. T5: The master reads the data (Data 1) from the PRDATA bus. T6: The master deasserts PENABLE and PSEL. The read transfer is complete.

The transfer undergoes extension when PREADY is lowered during an Access phase. The subsequent signals maintain their state while PREADY remains in a LOW state.

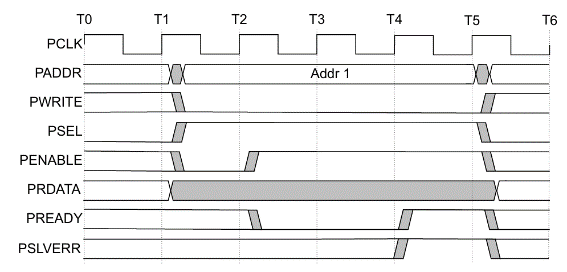
**Error response**

PSLVERR serves to signal an error condition during an APB transfer, applicable to both read and write transactions. Its validity is confined to the last cycle of an APB transfer, characterized by PSEL, PENABLE, and PREADY being simultaneously asserted. While it is recommended that PSLVERR be set to LOW when PSEL, PENABLE, or PREADY are not asserted, it is not mandatory.

In case of an error, it is permissible for the peripheral state to remain unchanged, as this behavior is peripheral-specific. An error in a write transaction does not imply that the peripheral register has not been updated. In the case of read transactions encountering an error, the returned data may be invalid. Despite receiving an error response to a read transfer, a Requester may still utilize the data. It should be noted that a Completer is not obliged to support PSLVERR. In instances where PSLVERR is not supported by a Completer, the appropriate input to the Requester is connected to LOW.



**Fig 8:** Unsuccessful write transfer.

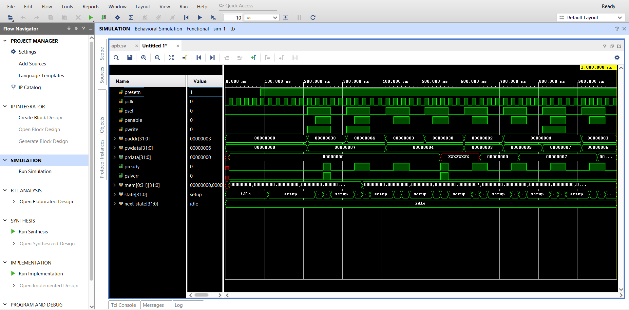


**Fig 9:** Unsuccessful read transfer.

A read transfer can conclude with an error response, signifying the absence of valid read data. This error condition is denoted by PSLVERR in an APB transfer. Such errors can manifest in both read and write transactions. PSLVERR is deemed valid solely during the final cycle of an APB transfer, specifically when PSEL, PENABLE, and PREADY are all asserted. While it is advisable to set PSLVERR to LOW when it is not being sampled, i.e., when any of PSEL, PENABLE, or PREADY are LOW, it is not obligatory.

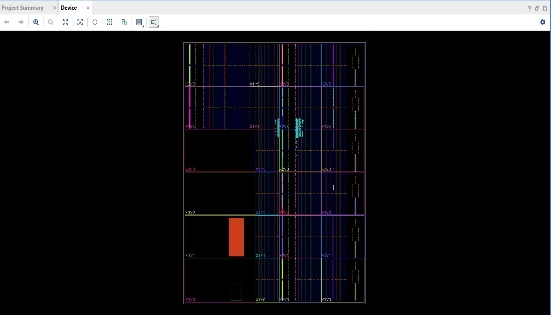
**V. RESULTS**

## Simulation Result



**Fig 10:** Simulation result for AMBA APB

**Implementation**



**Fig 11:** Implementation Design

# CONCLUSION

# This paper on "Design and Verification of AMBA APB protocol using system Verilog" addresses the complexities involved in designing and verifying high-performance bus protocols for on-chip communication. Through the application of system Verilog, a powerful hardware description and verification language, the project demonstrates a methodical approach to model the AMBA APB protocol, highlighting its flexibility and efficiency in handling high-speed data transfers and multiple bus masters.

# REFERENCES

[1]. Anushka Dwivedi, Anand Jatti, “Design and Implementation of AMBA APB Protocol Using System Verilog”, International Research Journal of Modernization in Engineering Technology and Science, Vol. 4, Issue. 07, July 2022.

[2]. Dipti Girdhar Shankar, “Design and Verification of AMBA APB Protocol”, International Journal of Computer Application, Vol. 95 Issue. 21, June 2014.

[3]. Vaishnavi R.K, Bindu. S, Sheik Chandbasha “Design and Verification of APB Protocol using System Verilog and Universal Verification Methodology”, International Research Journal of Engineering and Technology (IRJET), Vol. 6, Issue. 06, June 2019.

[4]. Mukunthan J, A Daniel Raj, Shruthi T “Design and Implementation of AMBA APB Protocol”, IOP Conference Series: Materials Science and Engineering, 2021.

[5]. Dileep Reddy, Sathya Srikanth Palle “Design and Implementation of Advanced Peripheral Bus (APB) Protocol in System Verilog", GIS Science Journal, Vol. 9, Issue. 01, June 2022.

[6]. Panikkar A, Gavankar R, Umarikar N “Design and implementation of area efficient, low power AMBA-APB Bridge for SoC”, International Conference on Green Computing Communication and Electrical Engineering (ICGCCEE), March 2014.