IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Design Of Comprehensive Bus Protocol(Cbp) On Chip Interface For Communication

[1] Pinnu Varshini, [2] Dr. M.N. Giri Prasad,

[1] PG Scholar, Department of ECE, JNTUCEA, Ananthapuramu 515002, Andhra Pradesh, India

[2] Adjunct Professor,Department of ECE,JNTUCEA,Ananthapuramu 515002,Andhra Pradesh,India

Abstract:

In this paper, we present the design and verification of a Comprehensive Bus Protocol (CBP) for RISC-V based system-on-Chip (SoC) architectures. CBP is designed to provide a low-complexity communication interface between the processor and essential peripherals like GPIOs and SPI. Unlike traditional designs that require separate data lines or large multiplexers for each peripheral—leading to higher cost and complexity—CBP uses a shared bus architecture with address-based decoding. A centralized CBP decoder (reg_top) reads the upper bits of the address (paddr[31:16]) to select the appropriate peripheral, while each peripheral's internal register decoder uses the lower bits (paddr[15:0]) to access specific registers. This approach eliminates the need for individual data paths and complex muxing, making the hardware more scalable, efficient, and easier to integrate. The CBP is implemented in Verilog HDL, and functional verification is performed using SystemVerilog to ensure robust and reliable operation within embedded and safety-critical systems

Keywords:Comprehensive Bus Protocol (CBP) ,RISC-V SoC ,System-on-Chip (SoC) ,Low Power Design ,GPIO ,Shared Bus Architecture ,Address-Based Decoding ,CBP Decoder ,Register Mapping ,Verilog HDL ,SystemVerilog .

1.INTRODUCTION

In modern digital systems, especially System-on-Chip (SoC) architectures, efficient communication between internal components such as processors, memory controllers, and peripheral devices is critical for overall system performance and power efficiency. As these systems grow in complexity and density, the demand for reliable, high-speed, and low-power interconnect protocols has intensified. While traditional bus standards have proven effective for many applications, they often introduce overhead and complexity that are unnecessary for low-bandwidth peripheral communication.

The **Advanced Micro controller Bus Architecture (AMBA)**, developed by ARM® in 1996, has long served as the industry standard for inter-module communication within SoCs. AMBA introduced a hierarchy of bus protocols—ASB, AHB, APB, ATB, and later AXI—that cater to different performance and complexity requirements. However, many of these protocols are better suited for high-performance applications and may not be ideal for simpler tasks involving peripherals like UART, SPI, and I2C. These low-bandwidth components demand a streamlined, low-latency communication mechanism that minimizes power consumption and silicon footprint .

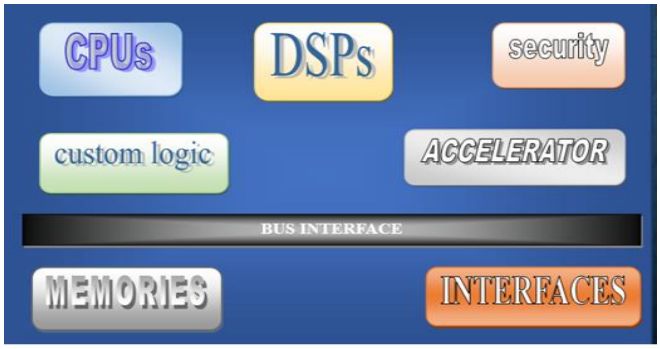


Fig 1: General Block Diagram

To address this specific requirement, the **Comprehensive Bus Protocol (CBP)** has been designed as a lightweight, synchronous, and address-based protocol tailored for peripheral communication in RISC-V-based SoCs. CBP is a non-pipelined protocol with a simple three-phase operation (Idle, Setup, Enable), which greatly simplifies timing analysis and hardware implementation. The protocol ensures deterministic behavior and predictable latencies—essential characteristics for real-time and low-power systems.

One of the key benefits of CBP is its **compatibility with RISC-V-based architectures**, which are increasingly popular in both academic and commercial domains due to their open-source and modular design. CBP's simplicity aligns well with RISC-V's philosophy of minimalism and customizability, allowing designers to tailor the bus interface specifically for their application's needs without incurring the overhead of generalized interconnect protocols.

2. PROPOSED SYSTEM

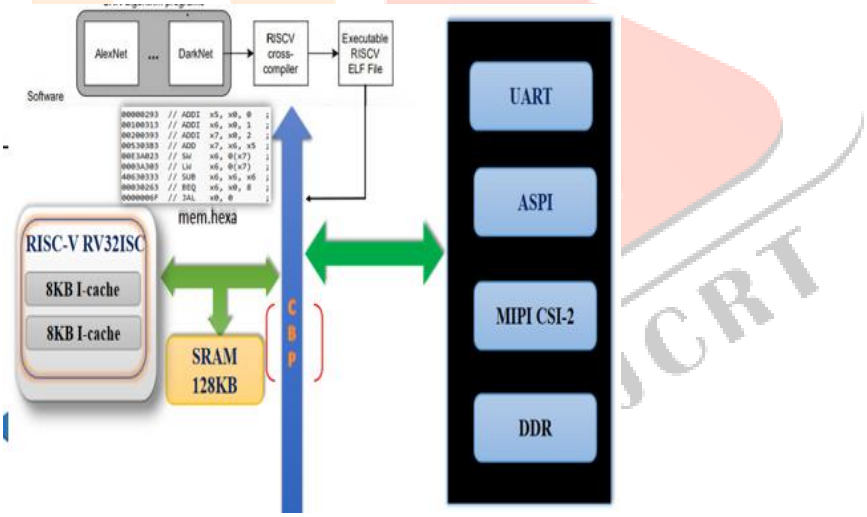


Fig:Proposed system

Comprehensive Bus Protocol (CBP) distinguishes itself through its **address-based design**, which eliminates the need for dedicated buses or complex multiplexers for each peripheral. Instead, all peripherals share a single CBP bus, and address decoding logic determines the target register or peripheral during communication. This modular approach supports scalability and makes the system design more maintainable and extensible without significantly increasing resource usage.

In the CBP-based communication design, instead of allocating separate data lines for each peripheral, a **single shared CBP bus** is used. This shared approach simplifies the interconnect structure and reduces hardware overhead. A central **CBP Decoder module (reg_top)** reads the upper 16 bits of the address (paddr[31:16]) to determine which peripheral (e.g., GPIO, SPI) should be activated. Once a peripheral is selected, its internal **register decoder** interprets the lower 16 bits (paddr[15:0]) of the address to access the appropriate internal register.

Without this **address-based selection mechanism**, the system would require individual data paths for every peripheral, making the hardware more **expensive, complex**, and harder to scale. Additionally, managing these paths would demand a large **multiplexer (MUX)** setup to route data correctly, resulting in inefficiency and

increased resource usage. In contrast, **address-based decoding** offers a clean, efficient, and scalable architecture, well-suited for low-power SoC designs where simplicity and resource optimization are critical.

ADDRESS MAPPING

Example: Address Mapping for Peripherals

Peripheral	Base Address (paddr[31:16])	Function
UART	0x0000_0000	Serial Communication
SPI	0x0001_0000	Serial Peripheral Interface
GPIO	0x0002_0000	General-Purpose I/O

the address mapping used for different peripherals in the CBP-based system, where each peripheral is assigned a unique base address defined by the upper 16 bits of the address bus (paddr[31:16]). For instance, the UART is mapped to 0x0000_0000 for serial communication, SPI to 0x0001_0000 for serial peripheral interfacing, GPIO to 0x0002_0000 for general-purpose I/O operations, the Timer to 0x0003_0000 for system timing functions, and DDR to 0x0004_0000 for memory control.Without address-based design, the CBP Master would not know which peripheral to communicate with!

Example: Adding a New I2C Peripheral

New Peripheral	Base Address (paddr[31:16])
I2C	0x0005_0000

The CBP Master sends requests to 0x0005_0000, and the CBP Decoder routes it to I2C .

The CBP Register Decoder (module) performs CPB Address Decoding (process).

the internal register mapping within a peripheral based on the lower 16 bits of the address (paddr[15:0]), along with their access permissions for read (pwrite=0) and write (pwrite=1) operations. Registers like ctrl0, ctrl2, ctrl4, and ctrl5 support both read and write operations (RW), while ctrl1 is read-only (RO) and ctrl3 is write-only (WO). Attempting an invalid access, such as writing to ctrl1 or reading from ctrl3, triggers a protocol error (pslverr=1). Any address outside the defined range is considered invalid and results in both read and write errors. This structure ensures robust access control and error handling in the CBP system by enforcing permission checks at the register level.

paddr[15:0] (Offset Address)	Register	Access Type	Read (pwrite=0)	Write (pwrite=1)
0x0000	ctrl0	RW	✓ Allowed	✓ Allowed
0x0004	ctrl1	RO	✓ Allowed	✗ Not Allowed (pslverr=1)
0x0008	ctrl2	RW	✓ Allowed	✓ Allowed
0x000C	ctrl3	WO	✗ Not Allowed (pslverr=1)	✓ Allowed
0x0010	ctrl4	RW	✓ Allowed	✓ Allowed
0x0014	ctrl5	RW	✓ Allowed	✓ Allowed
Others(default)	✗ Invalid	_	✗ Not Allowed (pslverr=1)	✗ Not Allowed (pslverr=1)

Fig:Table of control register

3.METHODOLOGY:

Operating states

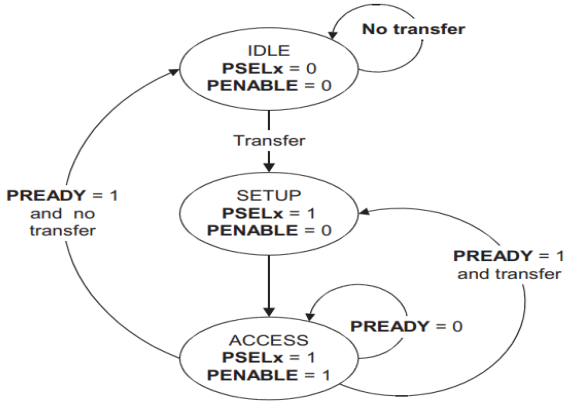


Fig :operating states

The **Finite State Machine (FSM)** governing a CBP (Comprehensive Bus Protocol) transfer cycle operates with three distinct states: **IDLE**, **SETUP**, and **ACCESS**, reflecting a simplified, synchronous control flow. In the **IDLE** state, the bus remains inactive with both control signals, PSELx and PENABLE, deasserted (0). This indicates that no data transaction is occurring. When a new transfer is initiated by the master, the FSM transitions to the **SETUP** state. Here, PSELx is asserted (1), while PENABLE remains low (0). The SETUP phase is responsible for asserting the peripheral select line and setting up the address and control signals required for the transfer.

The FSM then transitions to the **ACCESS** state, where both PSELx and PENABLE are asserted (1). This state signifies the actual data transfer phase between the master and the addressed slave peripheral. The FSM stays in ACCESS as long as the slave's PREADY signal is low, indicating the slave is not yet ready to complete the transfer. Once PREADY is asserted (1), the data transaction is successfully completed. At this point, if no new transfer is requested, the FSM returns to the IDLE state. Otherwise, it loops back to the SETUP state to initiate another transfer. This **two-cycle handshake mechanism** ensures reliable, deterministic communication, making CBP ideal for time-critical and low-power peripheral interactions in RISC-V SoC designs.

4.RESULT ANALYSIS:

THE COMPLETE DESIGN HAS BEEN DEVELOPED, AND SIMULATED USING MENTOR GRAPHICS QUESTASIM AND GVIM .THE IMPLEMENTATION AND RESULTS ARE ILLUSTRATED IN THE FIGURES BELOW.

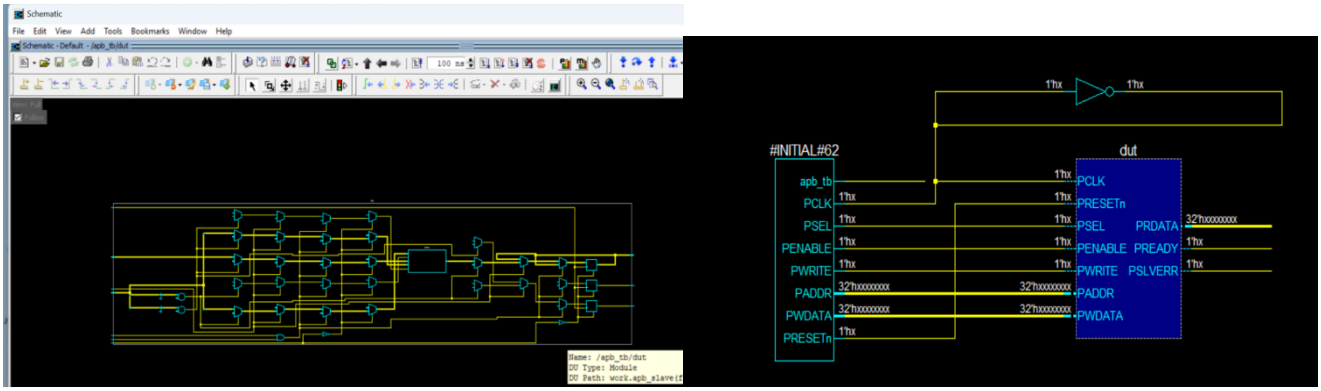


Fig.: C B P Schematic generated using Mentorgraphics Questasim

During Write Operation:

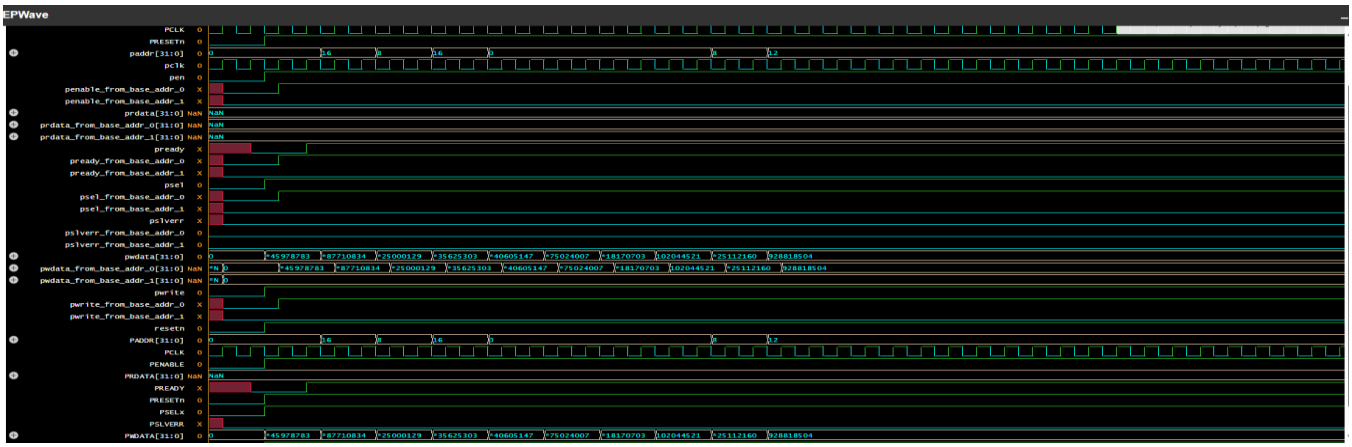


Fig:during write operation

To perform a write in CBP, the master first drives the address (PADDR), enables the peripheral (PSEL = 1), and indicates a write (PWRITE = 1) during the setup phase. It then asserts PENABLE = 1 in the access phase while sending the write data on PWDATA. The slave responds with PREADY = 1 when ready. The master then ends the transaction by lowering both PSEL and PENABLE.

During Read Operation:

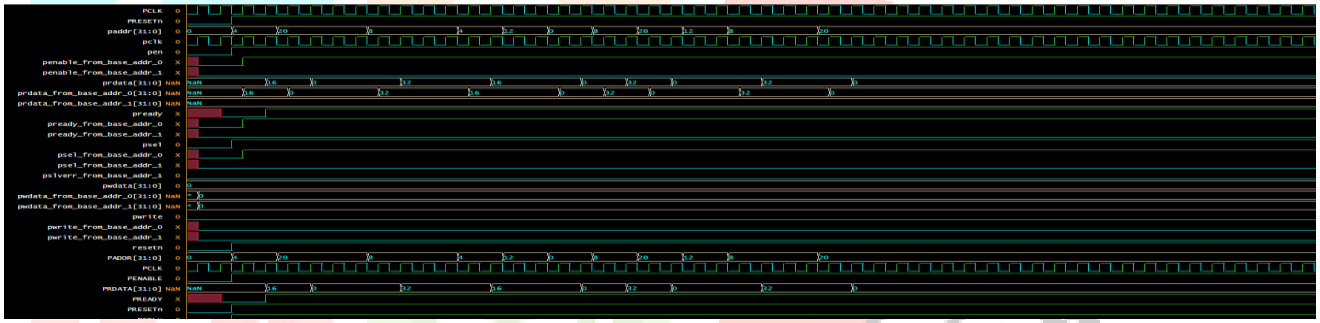
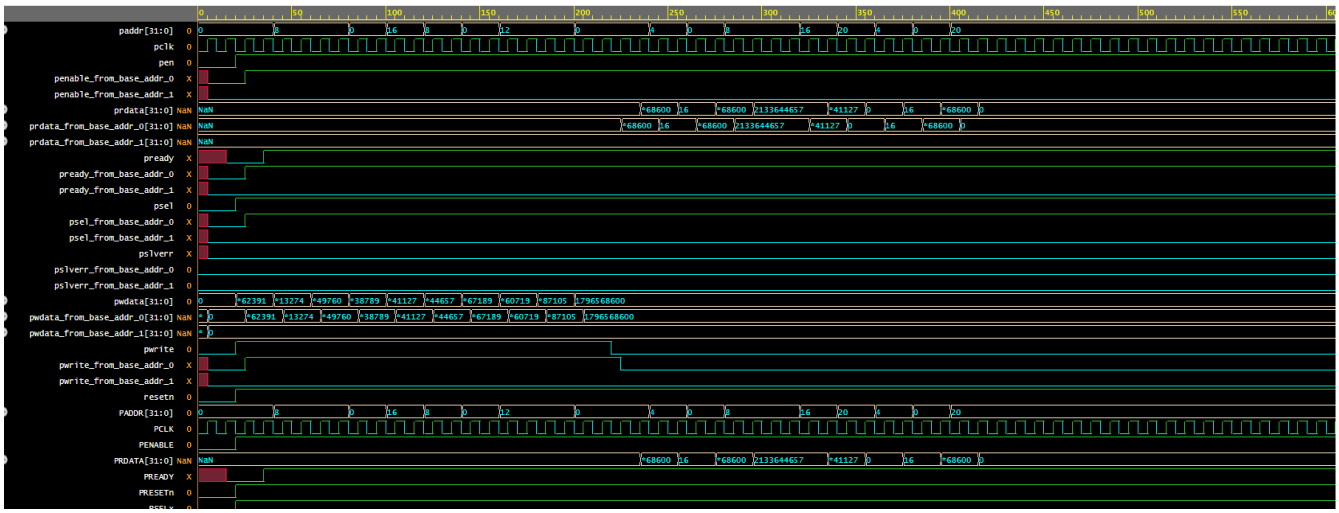


Fig:during read operation

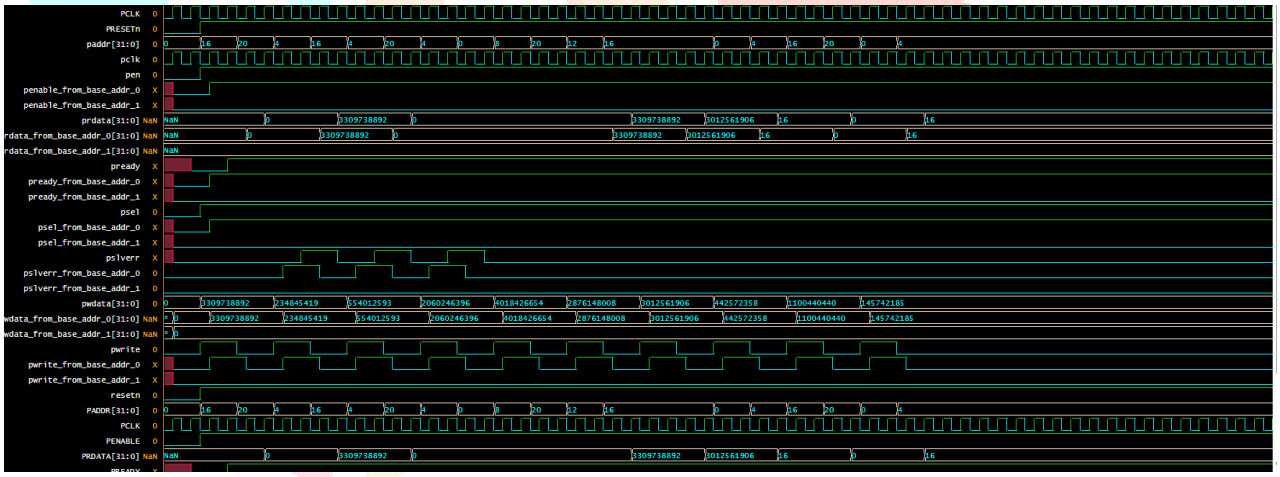
In a CBP read operation, the master begins by placing the address on PADDR, setting PWRITE = 0 to indicate a read, and asserting PSEL = 1 to select the peripheral—this is the setup phase. In the access phase, the master asserts PENABLE = 1, and the slave places the read data on PRDATA when ready, signaling this by setting PREADY = 1. Once the master detects PREADY, it deasserts both PSEL and PENABLE to complete the read transaction.

During Read & write Operation:



In CBP, both read and write operations follow a two-phase protocol. During the **setup phase**, the master places the address on PADDR, sets PSEL = 1 to select the peripheral, and sets PWRITE to 1 for a write or 0 for a read. In the **access phase**, the master asserts PENABLE = 1. For a **write**, it drives the data on PWDATA; for a **read**, the slave places data on PRDATA. The slave signals readiness by setting PREADY = 1, after which the master deasserts PSEL and PENABLE to complete the transaction.

During error response :



In both CBP read and write operations, the transaction begins with the **setup phase**, where the master drives the address on PADDR, sets PSEL = 1 to select the slave, and sets PWRITE to indicate read (0) or write (1). In the **access phase**, the master asserts PENABLE. For writes, it sends data on PWDATA; for reads, it waits for data on PRDATA. The slave then responds by setting PREADY = 1. If an error occurs, the slave also asserts PSLVERR = 1, indicating the transaction failed. Upon detecting PREADY, the master deasserts PSEL and PENABLE to complete the operation, regardless of success or error.

Functional coverage: Functional coverage measures **which features or scenarios** in a design or system have been exercised by a test suite

```
covergroup cgi;
  option.per_instance = 1;
  e1: coverpoint PSELx {bins a = {0, 1};}
  e2: coverpoint PENABLE {bins b = {0, 1};}
  e3: coverpoint PWRITE {bins c = {0, 1};}
  e4: coverpoint PADDR {bins n[] = {32'h00, 32'h04, 32'h08, 32'h0c, 32'h10, 32'h14};}
  e5: coverpoint PWDATA {bins a0 = {PWDATA[0]};
                         bins a1 = {PWDATA[1]};
                         bins a2 = {PWDATA[2]};
                         bins a3 = {PWDATA[3]};
                         bins a4 = {PWDATA[4]};
                         bins a5 = {PWDATA[5]};
                         bins a6 = {PWDATA[6]};
                         bins a7 = {PWDATA[7]};
                         bins a8 = {PWDATA[8]};
                         bins a9 = {PWDATA[9]};
                         bins b0 = {PWDATA[10]};
                         bins b1 = {PWDATA[11]};
                         bins b2 = {PWDATA[12]};
                         bins b3 = {PWDATA[13]};
                         bins b4 = {PWDATA[14]};
                         bins b5 = {PWDATA[15]};
                         bins b6 = {PWDATA[16]};
                         bins b7 = {PWDATA[17]};
                         bins b8 = {PWDATA[18]};
                         bins b9 = {PWDATA[19]};
                         bins c0 = {PWDATA[20]};
                         bins c1 = {PWDATA[21]};
                         bins c2 = {PWDATA[22]};
                         bins c3 = {PWDATA[23]};
                         bins c4 = {PWDATA[24]};
                         bins c5 = {PWDATA[25]};
                         bins c6 = {PWDATA[26]};
                         bins c7 = {PWDATA[27]};
                         bins c8 = {PWDATA[28]};
                         bins c9 = {PWDATA[29]};
                         bins d0 = {PWDATA[30]};
                         bins d1 = {PWDATA[31]};
                        }
  e6: coverpoint PREADY {bins d = {0, 1};}
  e7: coverpoint PSLVERR {bins e = {0, 1};}
endgroup
```

```
write Coverage = 92.8571
write Coverage pselx   = 100
write Coverage pen     = 100
write Coverage pwrite  = 100
write Coverage paddr   = 50
write Coverage pwdata  = 100
write Coverage pready  = 100
write Coverage pslverr = 100
Read coverage is 88.0952
Read Coverage pselx   = 100
Read Coverage pen     = 100
Read Coverage pwrite  = 100
Read Coverage paddr   = 16.6667
Read Coverage prdata  = 100
Read Coverage pready  = 100
Read Coverage pslverr = 100
```

BY USING FUNCTIONAL COVERAGE . WE MEASURE WITH DIFFERENT TEST CASES. HERE WE HAVE WRITTEN FOR BOTH WRITE AND READ FUNCTION GROUPS WHICH HAVE DIFFERENT FUNCTION POINTS .(BINS).

CONCLUSION:

This work presents a streamlined method for peripheral communication using address-based decoding, which promotes a modular and scalable system design while minimizing hardware complexity by removing the need for multiple data paths. The protocol implementation was validated using SystemVerilog in the QuestaSim (gVim) simulation environment, adhering to industry standards. Functional verification was further strengthened through the use of SystemVerilog covergroups, which were employed to monitor key protocol signals including PSELx, PENABLE, PWRITE, PADDR, PRDATA, PREADY, and PSLVERR. This ensured thorough validation of the protocol's behavior and contributed to achieving high functional coverage. The proposed design methodology proves effective for reliable and efficient integration of peripherals in modern System-on-Chip (SoC) architectures.

REFERENCES:

1. · N. J. Gupta, "APB Protocol. Define the APB signal interface," Medium, [Online]. Available: https://niharikajgupta.medium.com/apb-protocol-232d485394d8Medium

2. · "Verification of Advanced Peripheral Bus Protocol (APB V2.0)," International Research Journal of Engineering and Technology (IRJET), vol. 8, no. 6, pp. 796–800, June 2021. [Online]. Available: https://mail.irjet.net/archives/V8/i6/IRJET-V8I6796.pdfIRJET

3. · · "Design Bus Function Model for APB Slave (AMBA Based)," International Research Journal of Modernization in Engineering Technology and Science (IRJMETS), vol. 6, no. 4, pp. 1945–1950, Apr. 2024. [Online]. Available: https://www.irjmets.com/uploadedfiles/paper//issue_4_april_2024/52387/final/fin_irjmets1712767050.pdfIRJMETS

· · "QuestaSim Fact Sheet," Siemens EDA, [Online]. Available: https://static.sw.cdn.siemens.com/siemens-disw-assets/public/QzFgMxW5gizEDRIAZYTQE/en-US/Siemens-SW-QuestaSim-FS-85329-D5.pdfIJERA+2static.sw.cdn.siemens.com+2Siemens Community+2