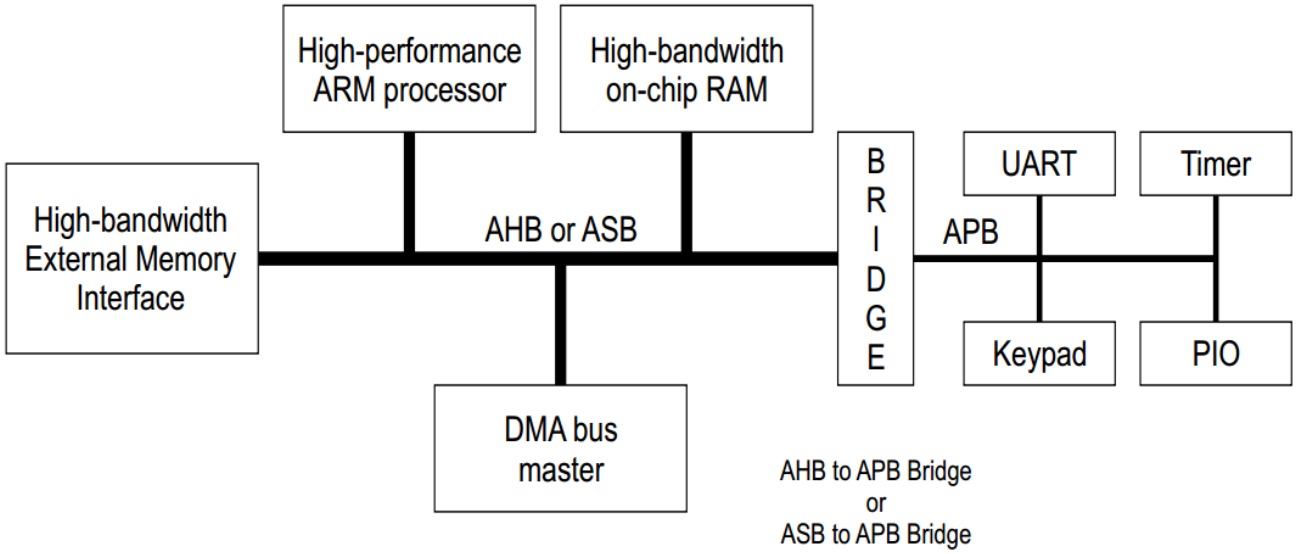
CHAPTER 1: INTRODUCTION TO APB PROTOCOL

1.1 What is APB?

APB stands for **Advanced Peripheral Bus**. It is a part of the AMBA (Advanced Microcontroller Bus Architecture) family of protocols developed by ARM. APB is designed to connect low-speed peripheral devices to a processor or system bus in a simple and efficient way.



1.2 Role of APB in AMBA Bus Architecture

AMBA is a standard bus protocol used widely in System on Chip (SoC) designs. It includes several buses such as AXI (Advanced extensible Interface), AHB (Advanced High-performance Bus), and APB. Among these, APB is the simplest and is mainly used for peripherals that do not require high-speed communication, like timers, UARTs, and simple I/O devices.

APB acts as a bridge that allows communication between the faster system bus (like AHB or AXI) and slower peripheral devices, ensuring that the peripherals get data correctly without complicated timing.

1.3 Basic Features and Purpose of APB

- **Simple interface:** APB uses fewer signals compared to other buses, making it easier to implement and verify.
- **Low power:** Because it operates in a straightforward, clocked manner, it helps reduce power consumption in peripherals.
- **Non-pipelined:** Transactions are performed step-by-step, making it suitable for simple devices that don’t need fast data throughput.
- **Cost-effective:** Ideal for low-cost and low-complexity designs where high speed is not necessary.

1.4 Simple Comparison with Other AMBA Buses

- **AXI:** Used for high-performance, high-frequency data transfer with pipelining and out-of-order transactions.
- **AHB:** Suitable for high-speed data transfer but simpler than AXI.
- **APB:** Used for low-speed, simple peripheral connections where the data bandwidth requirements are low.

CHAPTER 2: EVOLUTION AND HISTORY OF APB

2.1 Origins of AMBA and Need for APB

In the late 1990s, ARM introduced the AMBA bus architecture to standardize how different blocks inside a System on Chip (SoC) communicate. The first versions of AMBA included buses like AHB, which focused on high-speed data transfer. However, ARM realized that many peripheral devices in SoCs are simple and slow and didn't need such a complex, high-speed bus.

To address this, ARM introduced the **Advanced Peripheral Bus (APB)** as a simpler, low-cost bus designed specifically for these slower peripherals like UARTs, timers, and GPIOs.

2.2 Timeline and Versions of APB

- **AMBA 2.0 (Around 2000):** APB was first introduced as part of AMBA 2.0. It defined a simple, non-pipelined bus interface for peripherals. This version is still widely used because of its simplicity and efficiency.
- **AMBA 3.0 (Mid-2000s):** While APB itself remained largely unchanged, AMBA 3 introduced AXI for high-performance needs, keeping APB for peripherals.
- **AMBA 4.0 and Beyond:** APB continued as the go-to bus for low-speed peripherals. Some minor updates and clarifications were added, but the core APB protocol remained simple and stable.

2.3 Why APB Remains Relevant

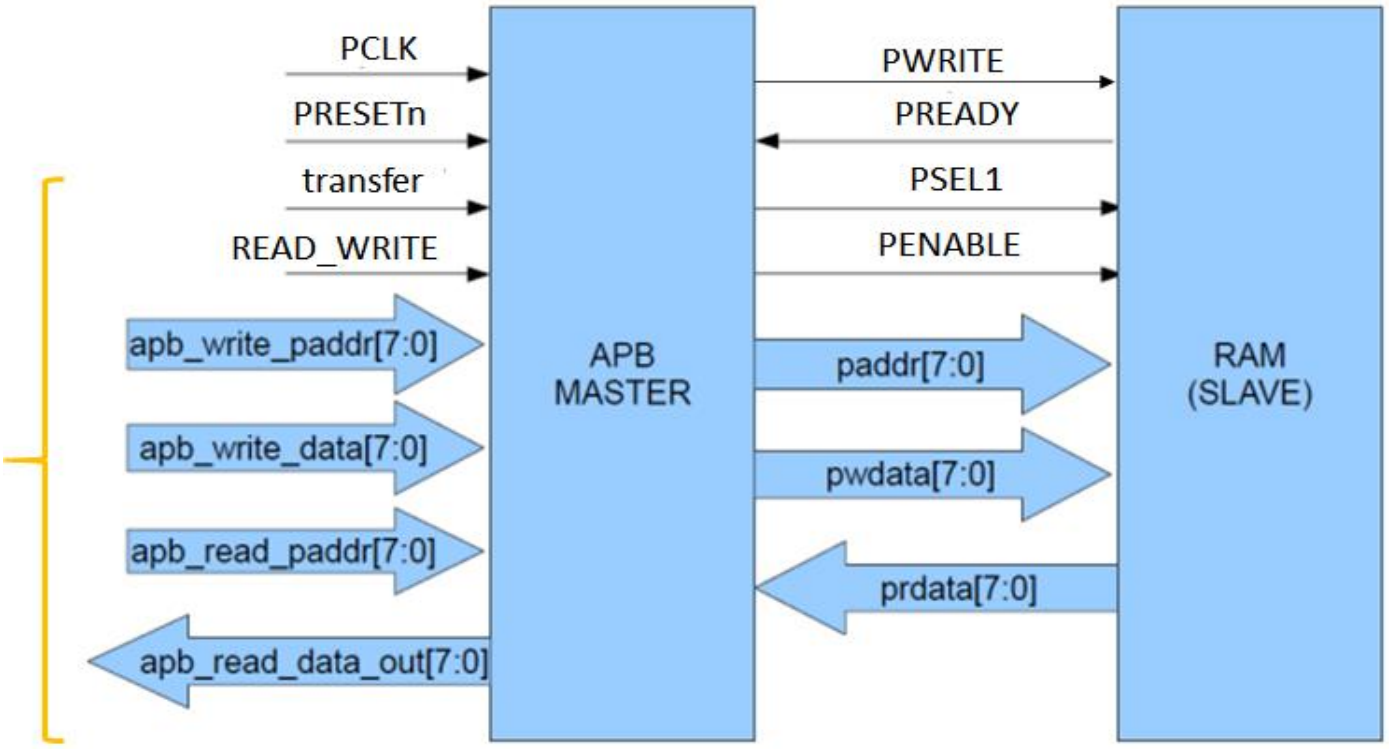
Despite many new bus standards, APB is still heavily used because:

- It is easy to design and verify, which reduces development time and cost.
- It keeps power consumption low, critical in battery-powered devices.
- It fits perfectly for slow peripherals where complex bus features are unnecessary.

CHAPTER 3: APB PROTOCOL ARCHITECTURE AND SIGNALS

3.1 APB Interface Components

The APB system mainly consists of three parts:



- **APB Master:** Usually part of the system bus (like AHB or AXI) that initiates read or write requests to peripherals.
- **APB Slave:** The peripheral device (like UART, timer) that responds to the master's requests.
- **Decoder:** A logic block that helps select which peripheral (slave) is active for communication.

3.2 APB Signals and Their Functions
APB uses a small set of signals to keep things simple:

Signal Name	**Description**
PCLK	Clock signal driving the APB interface.
PRESETn	Active low reset signal to initialize bus.
PADDR	Address bus indicating which peripheral register to access.
PWRITE	Indicates if the operation is a write (1) or read (0).
PENABLE	Enables the transfer during the access phase.
PSEL	Select signal to choose which slave device is active.
PWDATA	Write data bus carrying data to the slave.
PRDATA	Read data bus carrying data from the slave to the master.
PREADY	Indicates if the slave is ready to complete the transaction.
PSLVERR	Error signal if the slave detects a problem during transfer.

3.3 Basic Data Flow in APB

- The **master** places the address and writes/read command on the bus.
- The **decoder** activates the select signal (**PSEL**) for the targeted peripheral.
- The **PENABLE** signal is used to indicate the data phase of the transfer.
- The slave responds by either accepting data or providing data back.
- If the slave needs more time, it can delay the transfer using **PREADY**.
- If an error occurs, **PSLVERR** signals the issue.

3.4 Transaction Phases: Setup and Enable
APB transactions happen in two phases:

- **Setup Phase:** The address, write/read signal, and select signal are set. The **PENABLE** signal is low in this phase.
- **Enable Phase: PENABLE** goes high to start the data transfer. The slave then provides or accepts data.

This two-phase approach simplifies timing and control, making APB easy to implement.

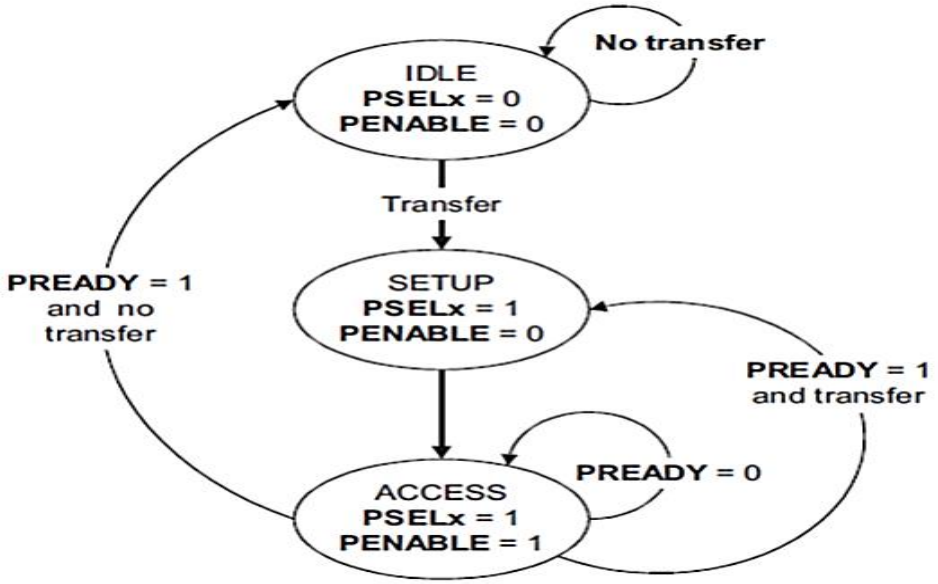
CHAPTER 4: APB PROTOCOL OPERATION AND TIMING

4.1 How APB Transactions Happen
APB transactions follow a simple step-by-step process that ensures data is correctly transferred between the master and the peripheral (slave). Each transaction consists of two main phases:

- **Setup Phase:**

- The master places the address of the peripheral register it wants to access on the **PADDR** lines.
- It also sets the **PWRITE** signal to indicate whether it wants to write data (1) or read data (0).
- The select signal (**PSEL**) is asserted to choose the target slave device.
- During this phase, the **PENABLE** signal is kept low.

- **Enable Phase:**
  - The master asserts the **PENABLE** signal to high.
  - In this phase, the actual data transfer happens:
    - For write operations, data on **PWDATA** is transferred to the slave.
    - For read operations, the slave places data on **PRDATA** for the master to read.
  - The slave asserts **PREADY** when it is ready to complete the transfer. If the slave is not ready, it can hold **PREADY** low to delay the transaction.



4.2 Clock and Timing Requirements

- APB is a synchronous bus, meaning all signals change in relation to the **PCLK** clock signal.
- All data and control signals are expected to be stable at the rising edge of **PCLK** during the enable phase.
- The simple two-phase protocol (setup and enable) helps reduce timing complexity.

4.3 Synchronous Operation and Low-Power Features

- APB's straightforward, clocked operation reduces power usage since peripherals only switch signals when necessary.
- There is no pipelining or burst transfers, so the bus stays idle during the setup phase, allowing peripherals to save power.

4.4 Timing Diagram Explanation

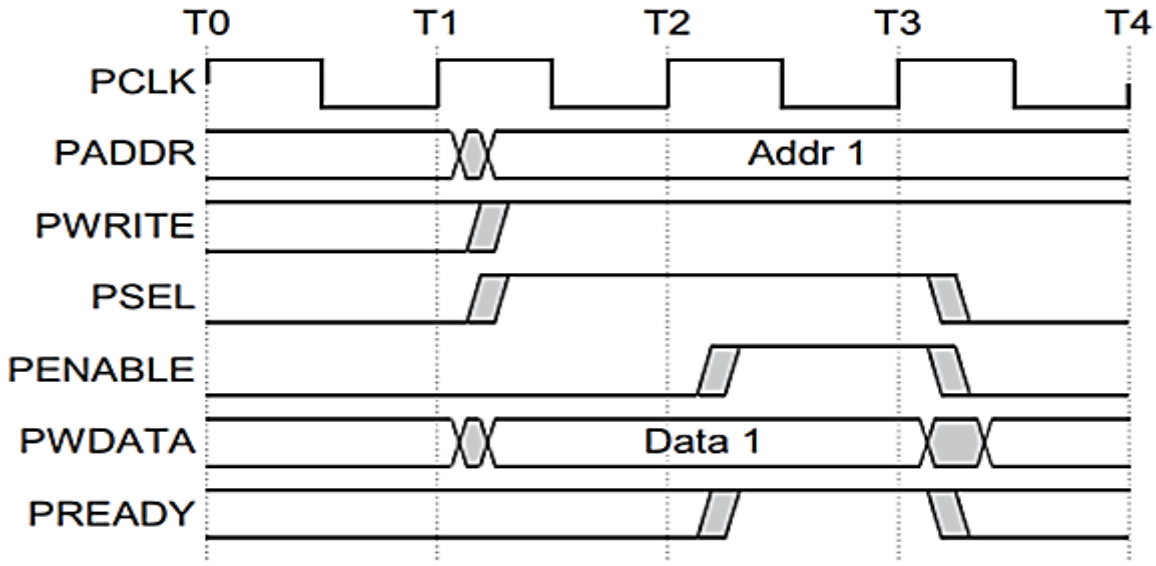
A typical APB timing diagram shows:

- When **PADDR**, **PWRITE**, and **PSEL** signals are set during the setup phase with **PENABLE** low.

- When **PENABLE** goes high to indicate the enable phase and data is transferred.
- The slave asserts **PREADY** to signal completion.
- If **PREADY** is low, the enable phase is extended until the slave is ready.

This simple timing approach allows easy design and debugging of APB interfaces.

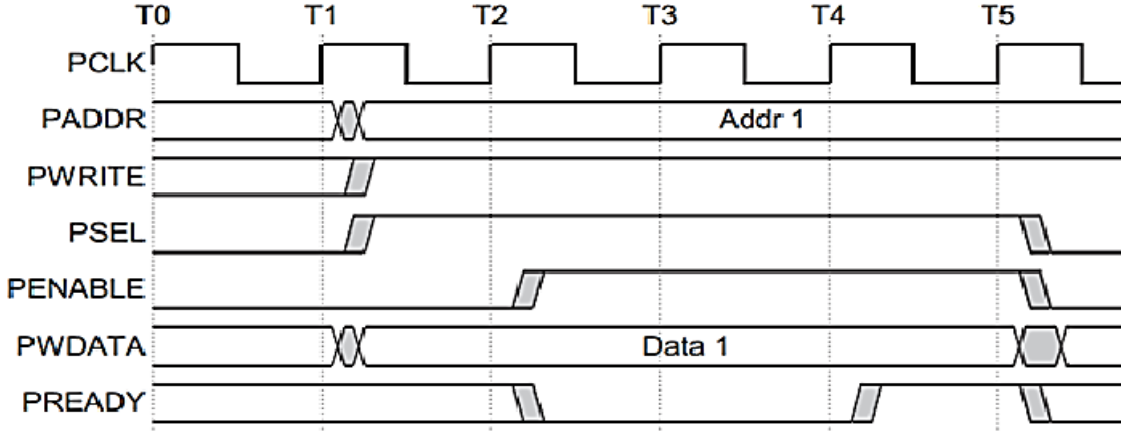
WRITE Transfer *– Without Wait States*



- At T1, a write transfer with address PADDR, PWDATA, PWRITE and PSEL starts.
- They will register at the next rising edge of PCLK, T2.
- This is Setup Phase of Transfer.
- After T2, PENABLE and PREADY are registered at the rising edge of PCLK.
- When asserted, PENABLE indicates starting of ACCESS Phase
- When asserted, PREADY indicates that slave can complete the transfer at the next rising edige of PCLK.
- PADDR, PDATA and control signals all should remain valid till the transfer completes at T3.
- PENABLE signal will be de-asserted at the end of transfer.

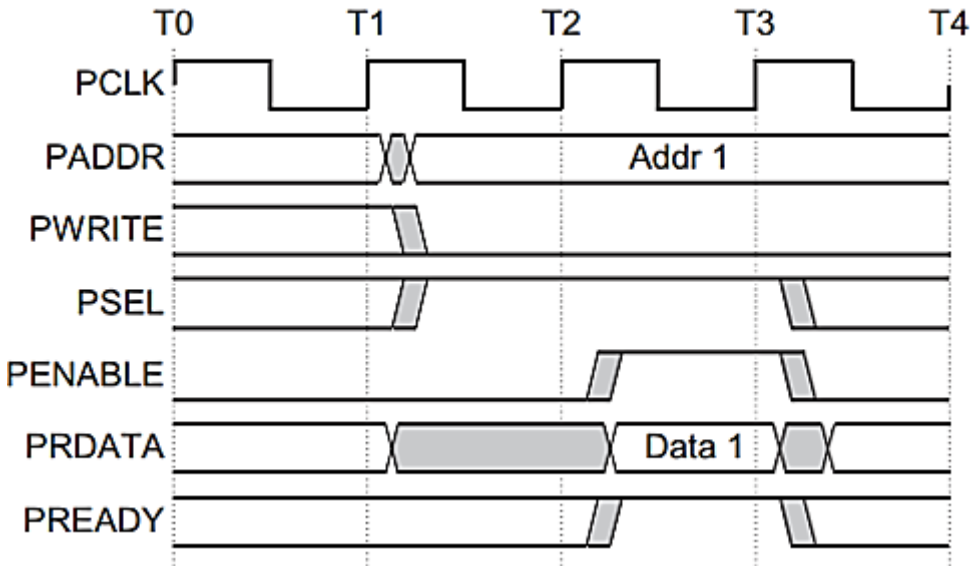
PSEL is also de-asserted, if next transfer is not to the same slave.

WRITE Transfer *– With Wait States*



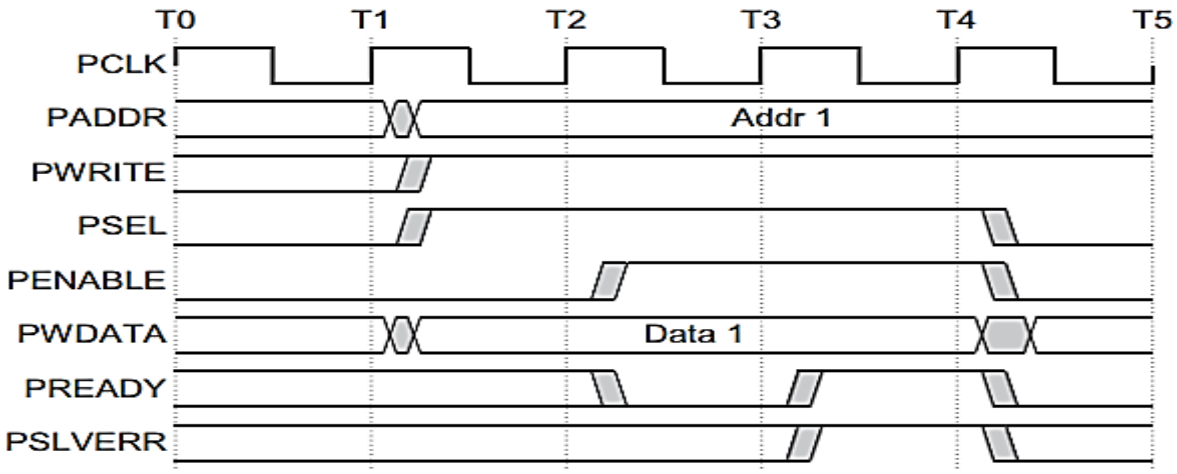
- During the ACCESS Phase, when PENABLE is high, the slave extends the transfer by driving PREADY low.
- The PADDR, PWRITE, PSEL, PENABLE, PWDATA, PSTRB, PPROT signals should remain unchanged while PREADY is low
- PREADY can take any value when PENABLE is low.
- It is recommended that the address and write signals are not changed immediately after a transfer, but remain stable until another access occurs.

READ Transfer *– Without Wait States*



- At T1, a READ transfer with address PADDR, PWRITE and PSEL starts.
- They will be registered at rising edge of PCLK.
- This is SETUP Phase of the transfer.
- After T2, PENABLE and PREADY are registered at the rising edge of PCLK.
- When asserted, PENABLE indicates the starting of ACCESS phase.
- When asserted, PREADY indicates that slave can complete the transfer at next rising edge of PCLK by providing the data on PRDATA.
- Slave must provide the data before the end of read transfer. i.e. before T3.

READ Transfer *– With Wait States*



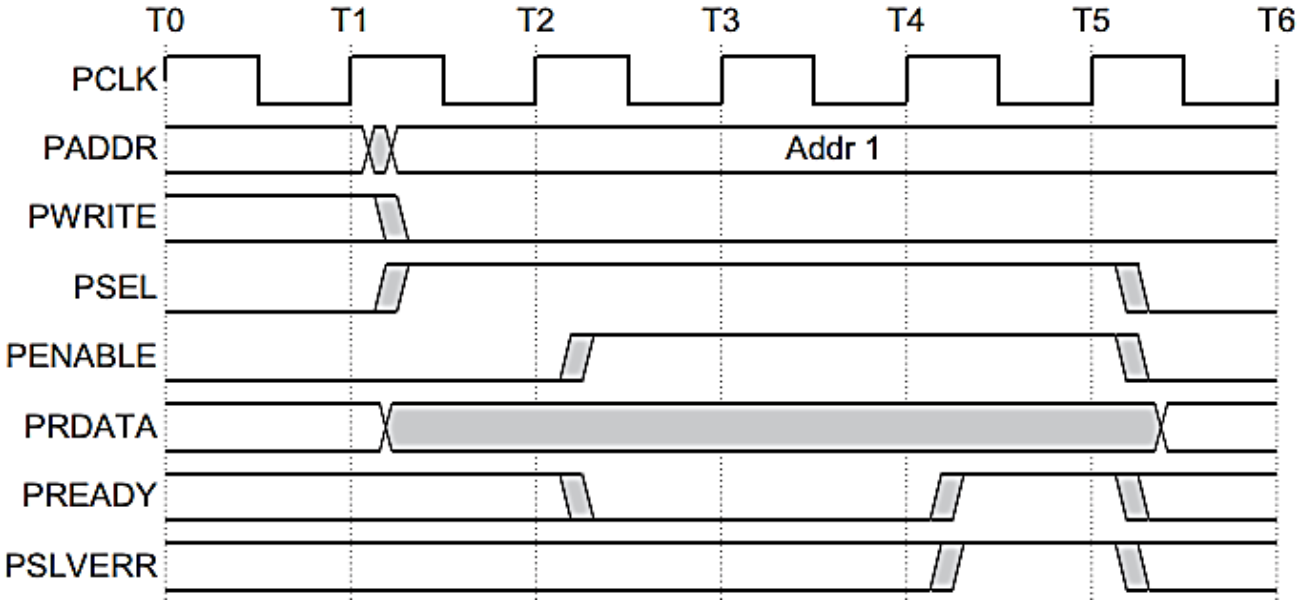
- During the ACCESS Phase, when PENABLE is high, the slave extends the transfer by driving PREADY low.
- The PADDR, PWRITE, PSEL, PENABLE, PPROT signals should remain unchanged while PREADY is low

ERROR Response

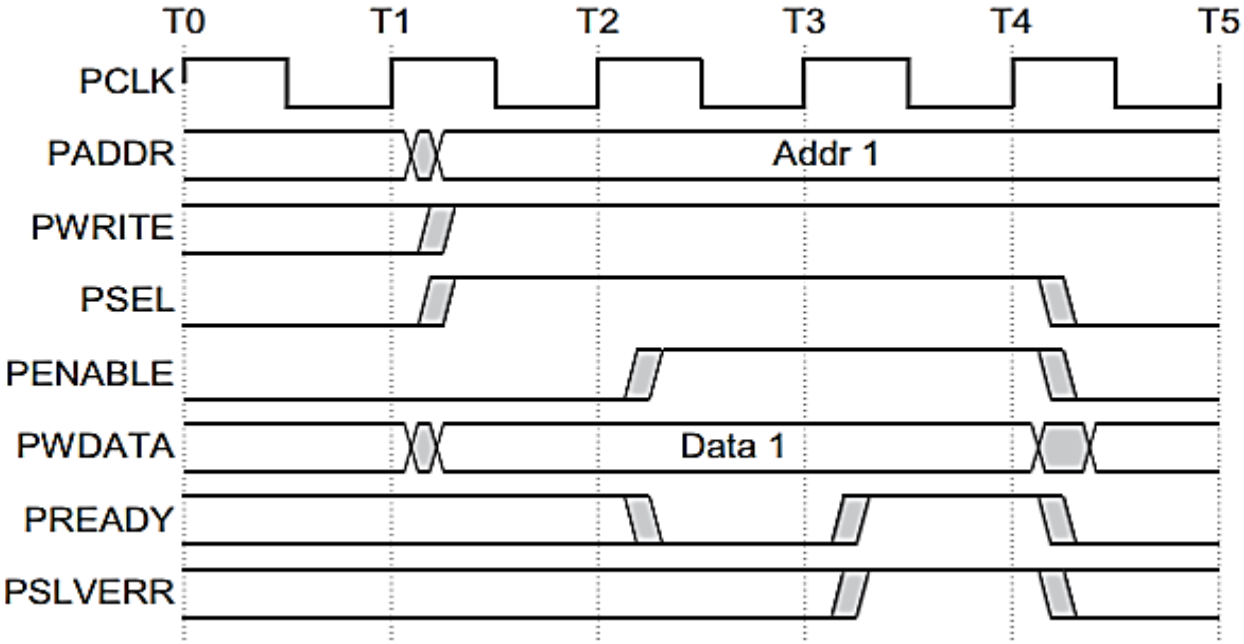
Whenever there is a problem in the transfer, Slave indicates the error response for the transfer by asserting the PSLVERR signal. PSLVERR is only considered valid during the last cycle f and APB transfer, when PSEL, PENABLE and PREADY are all HIGH. It is recommended, but not mandatory that you drive PSLVERR low when it is not being sampled.

Transactions that receive an error response, might or might not have changed the state of peripheral. For example, If APB master performs a write transaction to an APB slave and received an error response, it is not guaranteed that the data is not written on the slave peripheral.

Error Response for a read transfer:



Error Response for a write transfer:



Protection Unit Support:

To support complex system designs, it is often necessary for both the interconnect and other devices in the system to provide protection against illegal transactions. It is provided by Protection Unit in APB Protocol. The signals indicating the protection unit are PPROT [2:0].

The three levels of access protection are

- **PPROT [0]:**
  - LOW indicates Normal Access
  - HIGH indicates Privileged Access
- **PPROT [1]:**
  - LOW indicates Secure Access
  - HIGH indicates Non-Secure Access
- **PPROT[2]:**
  - LOW indicates Data Access
  - HIGH indicates Instruction Access

CHAPTER 5: ADVANTAGES, LIMITATIONS, AND APPLICATIONS OF APB

5.1 Advantages of APB

- **Simplicity:** APB has a simple interface with fewer signals, making it easy to design and verify.
- **Low Power:** Because it uses a simple two-phase handshake and does not support pipelining, APB helps reduce power consumption, which is important for battery-powered devices.
- **Cost-effective:** The reduced complexity leads to smaller area usage and lower cost in hardware implementation.
- **Ideal for Slow Peripherals:** APB perfectly fits peripherals like timers, UARTs, GPIOs, and low-speed sensors that don't require fast data transfer.
- **Good Integration:** Easily connects with high-speed buses like AHB or AXI through bridge logic, allowing smooth communication within the SoC.

5.2 Limitations of APB

- **Low Performance:** APB is not designed for high-speed or high-bandwidth data transfer since it is non-pipelined and simple.
- **No Burst Transfers:** Unlike AHB or AXI, APB does not support burst or pipelined transactions, limiting its use for data-intensive peripherals.
- **Single Access at a Time:** Only one transaction can occur at any moment, which can cause delays if multiple peripherals require access simultaneously.
- **No Error Correction:** Although it has an error signal (PSLVERR), APB does not have complex error correction or advanced flow control features.

5.3 Common Applications of APB

- **Peripheral Controllers:** Such as UART (serial communication), SPI, I2C controllers.
- **Timers and Watchdog Timers:** For managing time-based functions in embedded systems.
- **GPIO (General Purpose Input Output):** Simple digital input and output pins.

- **Interrupt Controllers:** Managing hardware interrupts from peripherals.
- **Configuration and Control Registers:** For setting and reading control information of various components in the SoC

References

1. **ARM AMBA 2.0 Specification**
   ARM Ltd., "AMBA 2.0 Specification," 1999.
   https://developer.arm.com/documentation/ihi0022/b
2. **ARM AMBA 3.0 Specification**
   ARM Ltd., "AMBA 3 APB Protocol Specification," 2003.
   https://developer.arm.com/documentation/ihi0031/b
3. **ARM AMBA 4.0 Specification**
   ARM Ltd., "AMBA 4 APB Protocol Specification," 2010.
   https://developer.arm.com/documentation/ihi0052/a
4. **Overview of AMBA Protocols**
   ARM Ltd., "AMBA Protocol Overview," ARM Developer Documentation.
   https://developer.arm.com/architectures/system-architecture/amba