# Implementation of APB Master Bridge using Verilog


| Team Members Names |
| --- |
| Fatma Ali Gamal Eldin Mohammed |
| Tasneem Abo El-Esaad Abd El-Razik Abo El-Esaad |
| Mohamed Hosam Elden Abd Alhakim Abd Alhady |
| Hadeer Muhammed Ali Abdo Saleh |
| Fatma Hazem Abd El-salam Mohamed |
| Yousef Sherif Abdel Fadil |


**Submitted to:**

**Prof. Ashraf Salem**

**Prof. Mohamed Desouky**

**Siemens Company**

Date of Submission: 15/9/2022

# Table of Contents :



# Table of figures

# Introduction to AMBA and APB buses

The ARM Advanced Microcontroller Bus Architecture (AMBA) is an open-standard, on-chip interconnect protocol for the connecting and management of functional blocks in system-on-a-chip (SoC) designs. It makes it easier to create multi-processor designs with plenty of controllers and components that are bus-based.

The AMBA specification defines an on-chip communications standard for designing high-performance embedded microcontrollers.

AMBA 3 specification defines four buses/interfaces:

- Advanced eXtensible Interface (AXI3 or AXI v1.0)
- Advanced High-performance Bus Lite (AHB-Lite v1.0)
- Advanced Peripheral Bus (APB3 v1.0)
- Advanced Trace Bus (ATB v1.0)

## Advanced eXtensible Interface (AXI)

A bus protocol that can issue multiple outstanding addresses, support separate address/control and data phases, unaligned data transfers using byte strobes, burst-based transactions with only the start address issued, separate read and write data channels to enable cheap DMA, and the simple addition of register stages to provide timing closure.

## Advanced High-performance Bus (AHB)

A bus protocol that separates the address/control and data phases along a defined pipeline.

Only a portion of the AMBA AXI protocol's features are supported. Only a portion of the whole AMBA AHB protocol specification should typically be employed, according to ARM, as it contains a number of characteristics that are not typically needed for master and slave IP developments.

A subset of the protocol is usually used. This subset is defined as the AMBA AHB-Lite protocol..

## The Advanced Peripheral Bus (APB)

A bus intended for control accesses with low bandwidth, such as register interfaces on system peripherals.Similar to AHB, this bus contains an address and data phase but a considerably smaller, lower complexity signal list (for example no bursts). Its interface was created for a low frequency system with a small bit size (32 bits).
The APB protocol is not pipelined, use it to connect to low-bandwidth peripherals that do not require the high performance of the AXI protocol.

The APB protocol relates a signal transition to the rising edge of the clock, to simplify the integration of APB peripherals into any design flow. Every transfer takes at least two cycles.
The APB can interface with:

• AMBA Advanced High-performance Bus (AHB)

• AMBA Advanced High-performance Bus Lite (AHB-Lite)
• AMBA Advanced Extensible Interface (AXI)
• AMBA Advanced Extensible Interface Lite (AXI4-Lite)

# AMBA APB Protocol Versions

## APB Specification Rev E

The oldest version of APB Protocol which was released in 1998 and now it is outdated.

## AMBA Specification Rev 2 (Issue A)

The second version of APB Protocol (first release for V1). It was released in 2003 which also called as APB 2 and defines the main simple operation of the protocol, its signals and the main two components of the bus, APB bridge and APB slave.

## AMBA 3 APB Protocol Specification v1.0 (Issue B)

It is the second release for V1. It was released in 2004 which also called as APB 3 and defines additional signals, PREADY and PSLVERR, which used in two new functionalities:

- Write and read with wait states: PREADY is used by APB master and slave to indicate that the transfer is done.
- Reporting an error: PSLVERR is used by APB master and slave to indicate that the transfer is done with an error.

## AMBA APB Protocol Specification v2.0 (Issue C)

It is the first release for version 2. It was released in 2010 which is also called APB4 and defines additional signals, PPROT and PSTRB, which used in two new functionalities:

- Protection of transaction: PPROT is used by APB master and slave to indicate that the transfer is secure or not.
- Sparse data transfer: PSTRB is used by APB master and slave to enable sparse on the write data bus.

## AMBA APB Protocol Specification (Issue D):

It is the second release for version 2. It is the most recent release, in 2021. It is called as APB 5 and defines new signals and functionalities:

- Wake up signaling: PWAKEUP is used to indicate any activity associated with any APB interface.
- User signaling: It is used to add signals that is may be required by the user and not defined in APB protocol.
- Parity protection and check signals.

## Here we will discuss, design using verilog and implement APB 4 master.

# APB Master

The master of APB bus is the bridge between the previous system bus and APB bus. This bridge is also a slave to the previous system bus. So it has two interfaces, previous bus slave interface and APB master interface. Its functionality is to convert complex transfer of the previous system bus to more simpler one of APB bus. It performs some functions:

- Capturing the address, write and write data signals and hold them valid throughout the transfer.
- Decoding the address to select which slave of APB to complete the transfer.
- Sends write data to the slave during write transfer.
- Capture read data from the slave to be sent to previous system bus during read transfer.
- Generate PENABLE signal which times the start of the transfer.

**Here we will design only the master interface with APB bus.**

# Ports Summary

| Signal | Type | Width | Description |
|---|---|---|---|
| PCLK | Input | 1 | The system clock signal. It's a global signal. All transfers are timed at the rising edge. |
| PRESETn | Input | 1 | The system global reset signal. It is used to get all sequential elements to reset state. |
| PRDATA | Input | Parameterized | Read data bus can be up to 32 bits. It is used by the slave to drive the data to the master in read transaction, when PWRITE is low. |
| PREADY | Input | 1 | It is used by the slave to inform the master if the transfer completed or not.<br>When high it indicates the completion of the transfer.<br>When low it extends the transfer until it goes high. |
| PSLVERR | Input | 1 | It is used by the slave to tell the master that the transfer has failed.<br>When high it indicates that there is an error.<br>When low it indicates that the transfer completed successfully. |
| Transfer | Input | 1 | It is used to initiate a transfer. |
| IN_ADDR | Input | Parameterized | Address bus can be up to 32 bits. It is received from the previous system bus. It is decoded to generate PSEL signal to each slave . |
| IN_DATA | Input | Parameterized | Write data bus can be up to 32 bits. It is received from the previous system bus. It is sent to the slave to be written in write transfer. |
| IN_WRITE | Input | 1 | It is received from the previous system bus. It is used to indicate the direction of the transfer.<br>When high it indicates write transaction.<br>When low it indicates read transaction. |

| | | | |
|---|---|---|---|
| IN_PROT | Input | 3 | PROT bus is received from the previous system bus and it is used to indicate if the transaction is secure or not. |
| IN_STRB | Input | Parameterized | STRB bus is received from the previous system bus and it is used to define write strobes.<br>It indicates which byte to be updated during a write transfer. Each 8 bits have 1 PSTRB bit.<br>It should be low during read transfer. |
| PADRR | Output | Parameterized | Address bus can be up to 32 bits. It is sent to the slave to access registers. It becomes valid at the start of the transfer. If there is no following transfer, it value won't change. |
| PWDATA | Output | Parameterized | Write data bus can be up to 32 bits. It is sent to the slave to be written in write transfer. It becomes valid at the start of write transfer. If there is no following transfer, it value won't change. |
| PWRITE | Output | 1 | It is used to indicate the direction of the transfer. It becomes valid at the start of the transfer and remains unchanged until there is another transfer.<br>When high it indicates write transaction.<br>When low it indicates read transaction. |
| PENABLE | Output | 1 | It is used to indicate the start of enable state, The second clock cycle of the transfer, it becomes valid after one clock cycle and should remain unchanged until the transfer completes. |
| PSELx | Output | Parameterized | This bus is used to select from which slave the data transfer is required. There is a signal for each slave. There could be up to 16 slaves. It becomes valid at the start if the transfer and remains unchanged during the transfer. At the end of the transfer it becomes unvalid. |
| PPROT | Output | 3 | It is used to indicate the protection level of the transaction, secure or not, normal or privileged, data or instruction access. It becomes valid at the start if the transfer and remains unchanged during the transfer. |
| PSTRP | Output | Parameterized | It is used to define write strobes. It indicates which byte to be updated during a write transfer. Each 8 bits have 1 PSTRB bit. It should be low during read transfer. It becomes valid at the start if the transfer and remains unchanged during the transfer. |
| OUT_SLVERR | Output | 1 | It indicates transfer failure. It is sent to the previous system bus to take the decision.<br>When high it indicates that there is an error.<br>When low it indicates that the transfer completed successfully. |
| OUT_RADATA | Output | Parameterized | Read data bus can be up to 32 bits. It is used to drive the data back to the master of the previous system bus in read transaction, when PWRITE is low. |

# Operating States

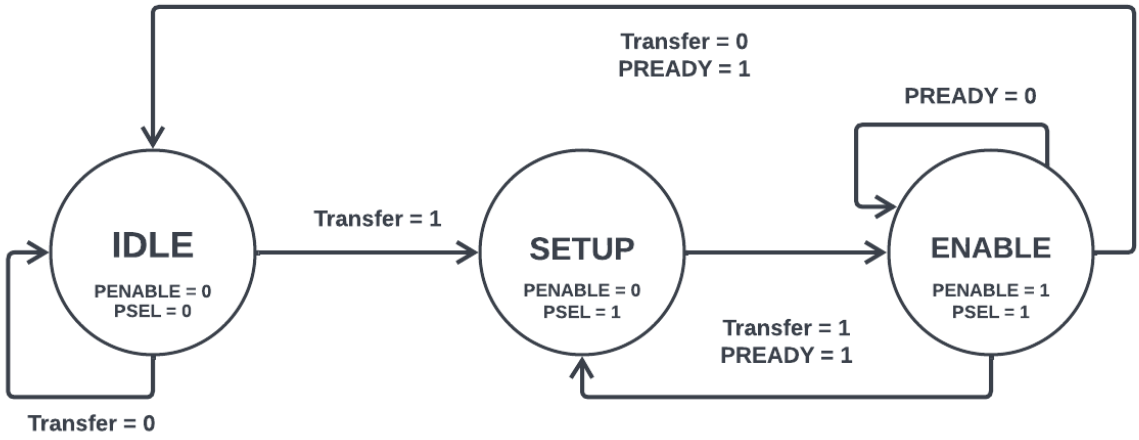


*Figure 1 : State Diagram*

| State Name | Description |
|---|---|
| **IDLE** | This is the default state of the APB where the **PENABLE** and **PSEL** signals are both low. |
| **SETUP** | The APB enters the SETUP state, where the proper choosen signal, **PSELx**, is asserted.<br>The bus always moves to the ENABLE state on the clock's rising edge after spending just one clock cycle in the SETUP state. Any write operation is performed in this state |
| **ENABLE** | The APB enters the ENABLE state waiting for the slave to raise its **PREADY** signal to allow it to move to the next state<br>It's decided upon the **Transfer** Signal if it's still high, moves back to the Setup state but if it's found low, it goes directly to the IDLE state .<br>Any read operation is performed in this state |

# Transfers

## Write Transfer with no wait states

In this transfer, he master sends to a desired slave some data to be written along with the address to be written in.

The operation begins with raising the **Transfer** signal to one then at the next positive edge of the clock **T1** the actual sending operation begins as follows:

At **T1**, the master is triggered to move from its idle state to the setup state where the master sets the **PSEL** signal with value that calls the slave in desire and since it's a write transfer so the **INWRITE** signal is found one and so sends **PWRITE, PWDATA** along with the desired **PADDR. PSTRB** and **PPROT** are sent also to the slave.

And then without any conditions the master moves to the enable state at **T2.**

At **T2**, the master sets **PENABLE** to high and the salve raises its **PREADY** signal. The transfer should be completed during this clock cycle. **Transfer** is set to low.

The salve sets its **PREADY** signal to low triggering the master to move back to the idle state at **T3** and hence **PENABLE, PSEL** is set to low while **PWRITE, PADDR, PSTRB, PPROT** and **PWDATA** signals remain unchanged to reduce power consumption.

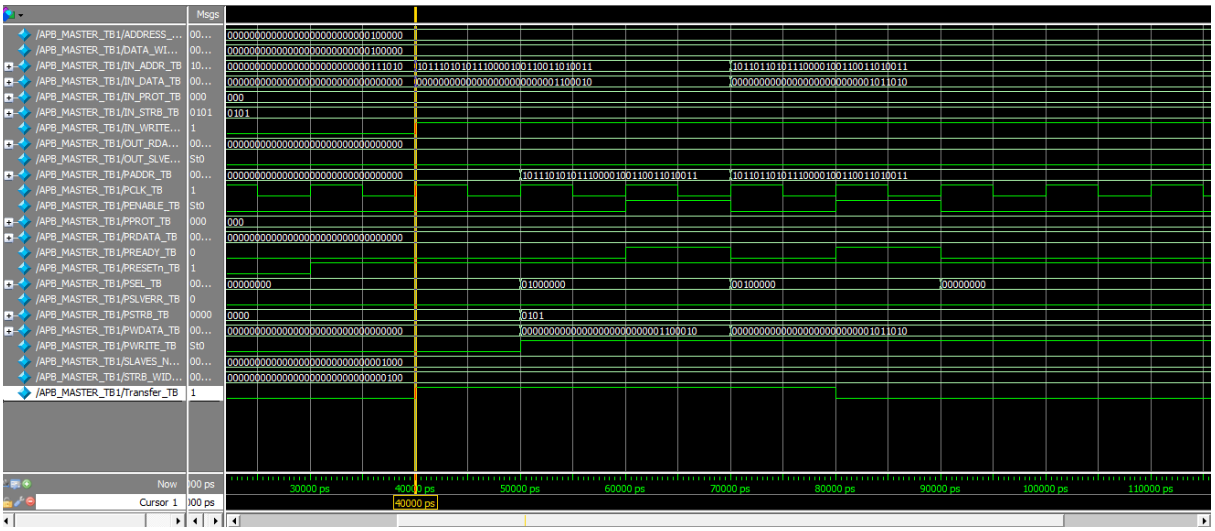


*Figure 2 : Waveform of Write Transfer with no wait states*

## Write with wait states

The operation begins with raising the **Transfer** signal to one then at the next positive edge of the clock **T1** the actual sending operation begins as follows:

At **T1**, the master is triggered to move from its idle state to the setup state where the master sets the **PSEL** signal with value that calls the slave in desire and since it's a write transfer so the **INWRITE** signal is found one and so sends **PWRITE , PWDATA** along with the desired **PADDR. PSTRB** and **PPROT** are sent also to the slave.

And then without any conditions the master moves to the enable state at **T2**.

At **T2**, the master sets **PENABLE** to high and the salve sets its **PREADY** signal to low which indicate that there are waiting states where the master keeps everything on its buses unchanged, **PENABLE, PSEL, PWRITE, PADDR, PSTRB, PPROT** and **PWDATA**, while remaining in enable state.

When the slave raises its **PREADY** signal**,** the transfer should be completed during this clock cycle.

After the transfer is done, the salve sets its **PREADY** signal to low triggering the master to move back to the idle state at **T3** and hence **PENABLE, PSEL** is set to low while **PWRITE, PADDR, PSTRB, PPROT** and **PWDATA** signals remain unchanged to reduce power consumption.

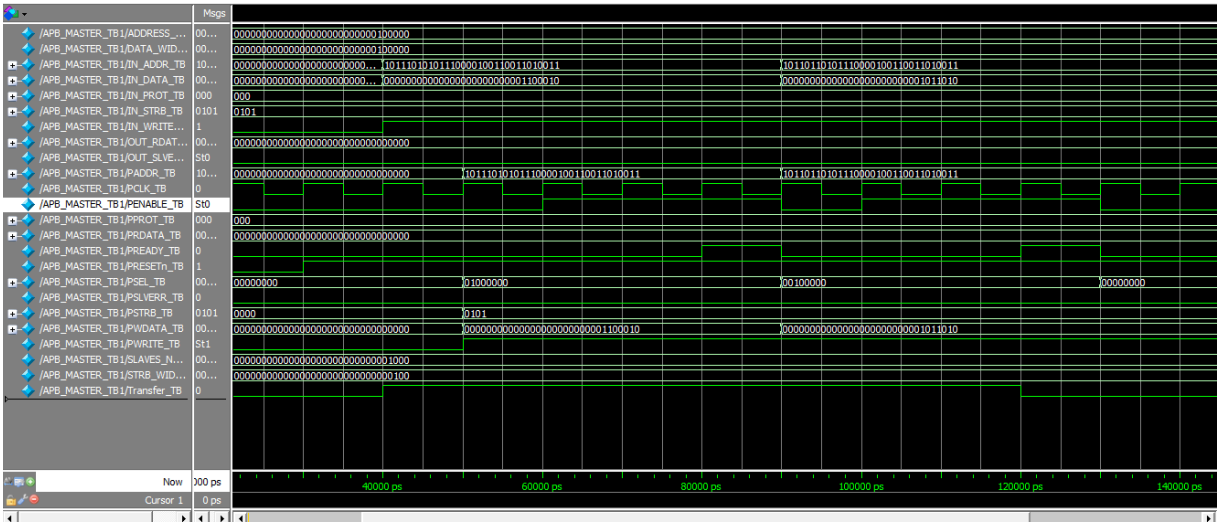


*Figure 3 : Waveform of Write Transfer with wait states*

## Read with no wait states

In this transfer, the master requests from a desired slave the data stored in a specified address.

The operation begins with raising the **Transfer** signal to one then at the next positive edge of the clock **T1** the actual sending operation begins as follows:

At **T1**, the master is triggered to move from its idle state to the setup state where the master sets the **PSEL** signal with value that calls the slave in desire and since it is a read transfer so the **INWRITE** signal is found low and so sends **PWRITE** along with the desired **PADDR. PSTRB** should be set to low during read transfer.

And then without any conditions the master moves to the enable state at **T2.**

At **T2**, the master sets **PENABLE** to high and the salve raises its **PREADY** signal. . The slave should provide the desired data before the end of this cycle.

The master receives the data in desire and sends it to the previous system bus through **OUT_RDATA** and the **Transfer** signal is set low.

After the transfer is done, the salve sets its **PREADY** signal to low triggering the master to move back to the idle state at **T3** and hence **PENABLE, PSEL** is set to low while **PWRITE, PADDR, PSTRB, PPROT** and **OUT_RDATA** signals remain unchanged to reduce power consumption.

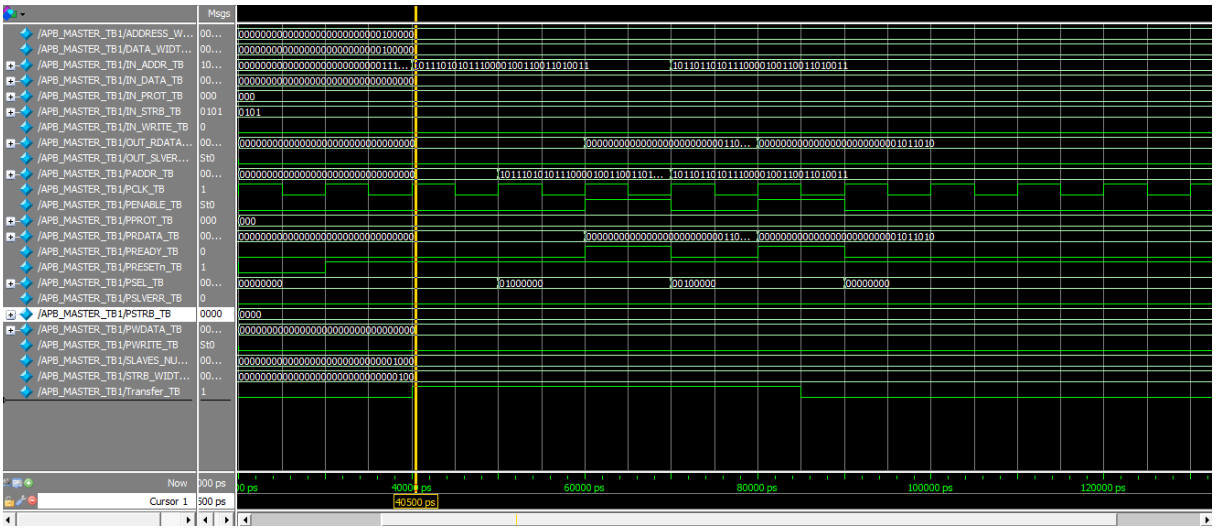


*Figure 4 :Waveform of Read transfer with no wait states*

## Read Transfer with wait states

In this transfer, the master requests from a desired slave the data stored in a specified address .

The operation begins with raising the **Transfer** signal to one then at the next positive edge of the clock **T1** the actual sending operation begins as follows:

At **T1**, the master is triggered to move from its idle state to the setup state where the master sets the **PSEL** signal with value that calls the slave in desire and since its a read transfer so the **INWRITE** signal is found low and so sends **PWRITE** along with the desired **PADDR. PSTRB** should be set to low during read transfer.

And then without any conditions the master moves to the enable state at **T2.**

At **T2**, the master sets **PENABLE** to high and the salve sets its **PREADY** signal to low which indicate that there are waiting states where the master keeps everything on its buses unchanged, **PENABLE, PSEL, PWRITE, PADDR, PSTRB** and **PPROT**, while remaining in enable state.

When the slave raises its **PREADY** signal**,** it should provide the desired data before the end of this cycle.

The master receives the data in desire and sends it to the previous system bus through **OUT_RDATA** and the **Transfer** signal is set low.

After the transfer is done, the salve sets its **PREADY** signal to low triggering the master to move back to the idle state at **T3** and hence **PENABLE, PSEL** is set to low while **PWRITE, PADDR, PSTRB, PPROT** and **OUT_RDATA** signals remain unchanged to reduce power consumption.

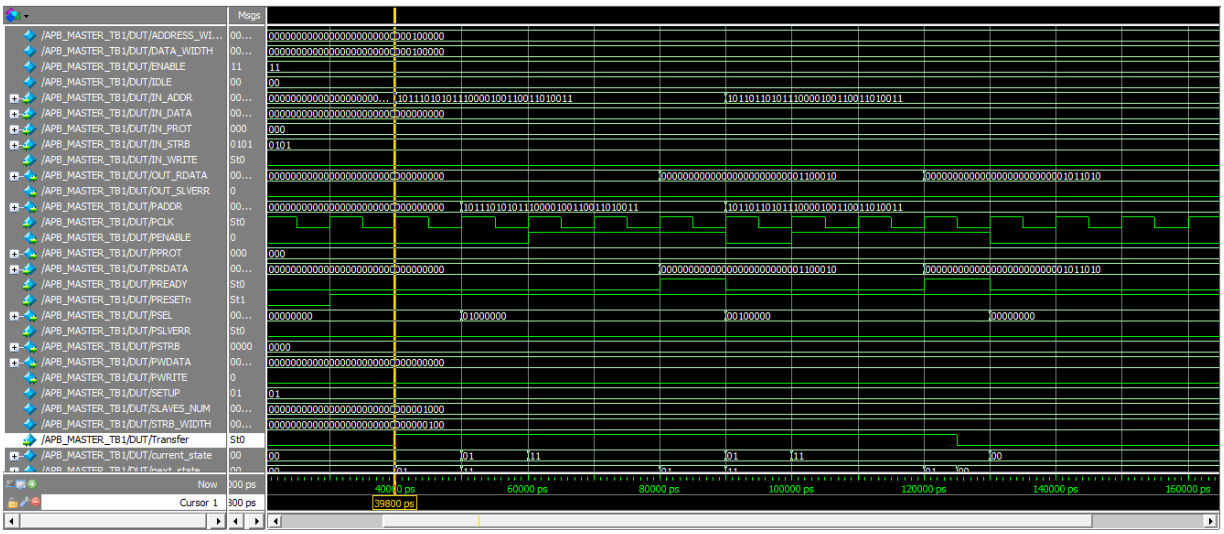


*Figure 5 : Waveform of Read Transfer with wait states*

## Multiple Transfers

This operation only differs from the normal read and write transfers in the enable state as the **Transfer** signal is not set to low so the master does not move back to the idle state at **T3** but instead goes to the setup state preparing for another Transfer whether it targets the same slave or not .

## Write and read with an error

Error could happen in both write or read transfers if the address is out of range or the read data is not valid… etc.

Salve uses **PSLVERR** signal to indicate that the transfer completed with an error. It drives it during the last cycle of the transfer while **PENABLE**, **PSEL** and **PREADY** are high. If the slave doesn't have a **PSLVERR** bin then it should be driven to low.

When high that indicates an error, when low that indicates that the transfer completed successfully.

APB master returns back to the idle state and report it to the master of the previous system bus through **OUT_PSLVERR** to take the decision.

# Address Decoding

We use bus address to select a slave by decoding some bits in the address, by default bits from 29 to 26 for 16 slaves.

Our design can support up to 8 slaves so we need 3 bits, from 28 to 26.

There is a 3*8 decoder that takes three bits from the address and generate synchronized PSEL signals which are input to the slaves, only one bit from **PSEL** bus should be high at a time which indicates that the corresponding slave is selected to complete the current transfer.

# Information about the Code and Testbench

## Parameters

| Parameter Name | Description | Default value |
|---|---|---|
| DATA_WIDTH | Specifies the Data bus width | 32 Bits |
| ADDRESS_WIDTH | Specifies the Address bus width | 32 Bits |
| SLAVES_NUM | Specifies the number of slave to be connected to the APB bus with a maximum allowable value of 8 salves . | 8 slaves |
| STRB_WIDTH | Specifies the Strobe bus width | 4 Bits |

## LocalParam

Those parameters specifies the **three** finite machine states of the transfer operation **Gray Encoded**

| LocalParam Name | Encoded Value |
|---|---|
| IDLE | 2'b0 |
| SETUP | 2'b1 |
| Enable | 2'b3 |

## Internal Registers

Two registers one specifies the current state and another for the next state

## Code Structure

One module of Three Always Blocks

| Always Blocks | Description | Type |
|---|---|---|
| First | For reset and moving to the next state with the positive edge clock | Sequential |
| Second | For specifying the next state | Combinational |
| Third | For specifying outputs in each state | Sequential |
| Fourth | For Address decoding | Sequential |

## The code is tested with four test cases where each is implemented in a Task which are :

- Test write operation with no wait
- Test write operation with waits
- Test read operation with no waits
- Test read operation with waits

# Synthesis

We will synthesis our code to generate the schematic and validate that it is a synthesizable code using Design Compiler by running a TCL Script. The synthesis is done using 130n technology with 1.2 supply voltage and operates at 25°$C$ and typical corner operation of NMOS and PMOS. There are no constraints used for STA.

## TCL Script

syn_script.tcl

```
################## Design Compiler Library Files #setup ######################

#Add the path of the libraries to the search_path variable
lappend search_path /home/IC/APB/APB_MASTER/std_cells
lappend search_path /home/IC/APB/APB_MASTER/rtl

set TTLIB "scmetro_tsmc_cl013g_rvt_tt_1p2v_25c.db"

## Standard Cell library
set target_library $TTLIB

## Standard Cell & Hard Macros libraries
set link_library [list * $TTLIB]

######################## Reading RTL Files ###################################
read_file -format verilog APB_MASTER.v

################### Liniking All The Design Parts ##########################
link

###################### Mapping and optimization ########################
compile

#########################################################################
# Write out Design after initial compile
#########################################################################
write_file -format verilog -output APB_MASTER_mapped.v

################ starting graphical user interface ######################
gui_start

```

*Figure 6 :TCL Script*

## Terminal output and .log file

By running this command **syn_script.tcl > syn.log** in the terminal the output is saved in a .log file.

As seen there are no latches, loops or errors in the design. Compilation, mapping and optimization completed successfully.

```
Statistics for case statements in always block at line 77 in file
        '/home/IC/APB/APB_MASTER/rtl/APB_MASTER.v'
===============================================
|           Line           |  full/ parallel  |
===============================================
|            79            |    auto/auto     |
===============================================

Statistics for case statements in always block at line 117 in file
        '/home/IC/APB/APB_MASTER/rtl/APB_MASTER.v'
===============================================
|           Line           |  full/ parallel  |
===============================================
|           129            |    auto/auto     |
===============================================

Inferred memory devices in process
        in routine APB_MASTER line 62 in file
                '/home/IC/APB/APB_MASTER/rtl/APB_MASTER.v'.
=======================================================================================
|    Register Name     |   Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
=======================================================================================
|  current_state_reg   | Flip-flop |   2   |  Y  | N  |  Y  | N  | N  | N  | N  |
=======================================================================================

Inferred memory devices in process
        in routine APB_MASTER line 117 in file
                '/home/IC/APB/APB_MASTER/rtl/APB_MASTER.v'.
=======================================================================================
|    Register Name     |   Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
=======================================================================================
|      PSEL_reg        | Flip-flop |   8   |  Y  | N  |  Y  | N  | N  | N  | N  |
=======================================================================================

Inferred memory devices in process
        in routine APB_MASTER line 184 in file
                '/home/IC/APB/APB_MASTER/rtl/APB_MASTER.v'.
=======================================================================================
|    Register Name     |   Type    | Width | Bus | MB | AR | AS | SR | SS | ST |
=======================================================================================
|     PWRITE_reg       | Flip-flop |   1   |  N  | N  |  Y  | N  | N  | N  | N  |
|      PSTRB_reg       | Flip-flop |   4   |  Y  | N  |  Y  | N  | N  | N  | N  |
|    OUT_RDATA_reg     | Flip-flop |  32   |  Y  | N  |  Y  | N  | N  | N  | N  |
|      PPROT_reg       | Flip-flop |   3   |  Y  | N  |  Y  | N  | N  | N  | N  |
|   OUT_SLVERR_reg     | Flip-flop |   1   |  N  | N  |  Y  | N  | N  | N  | N  |
|     PENABLE_reg      | Flip-flop |   1   |  N  | N  |  Y  | N  | N  | N  | N  |
|      PADDR_reg       | Flip-flop |  32   |  Y  | N  |  Y  | N  | N  | N  | N  |
|     PWDATA_reg       | Flip-flop |  32   |  Y  | N  |  Y  | N  | N  | N  | N  |
=======================================================================================
Presto compilation completed successfully.
```

*Figure 7 :Terminal Output*

## Block Diagram

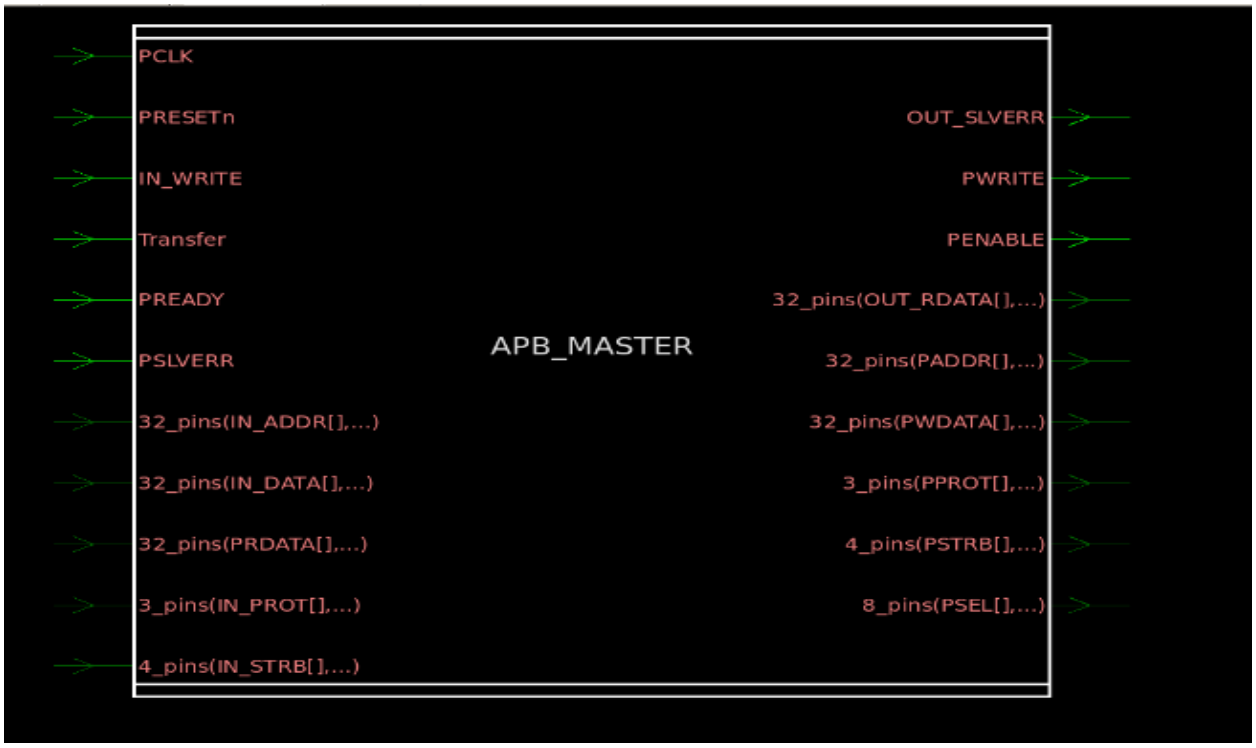


*Figure 8 :Block Diagram*

## Schematic

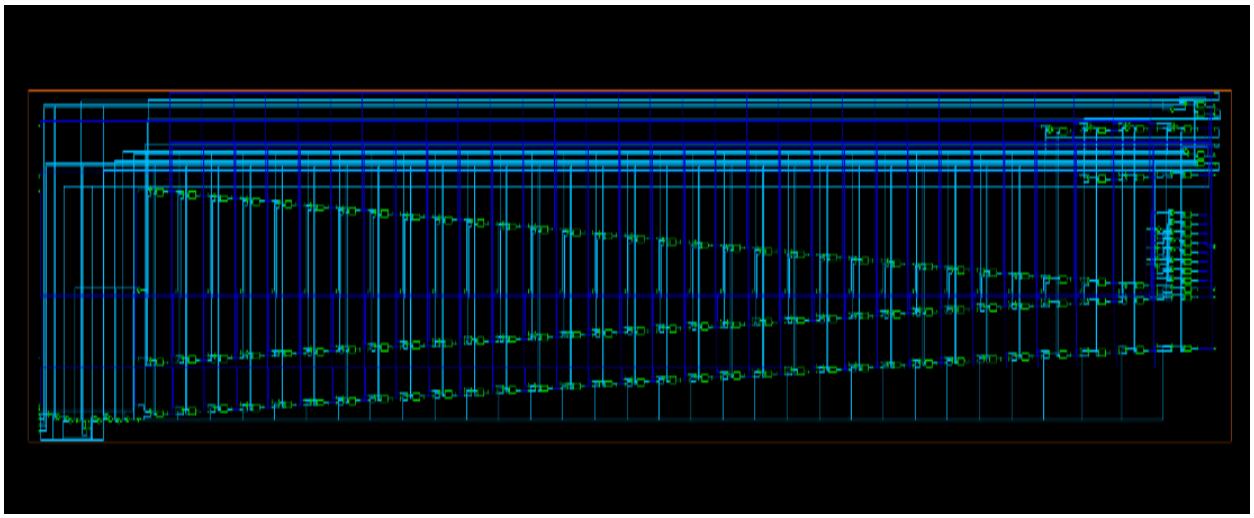

*Figure 9 : Schematic Diagram*