AHB to APB Bridge Design

Project Report

Submitted by

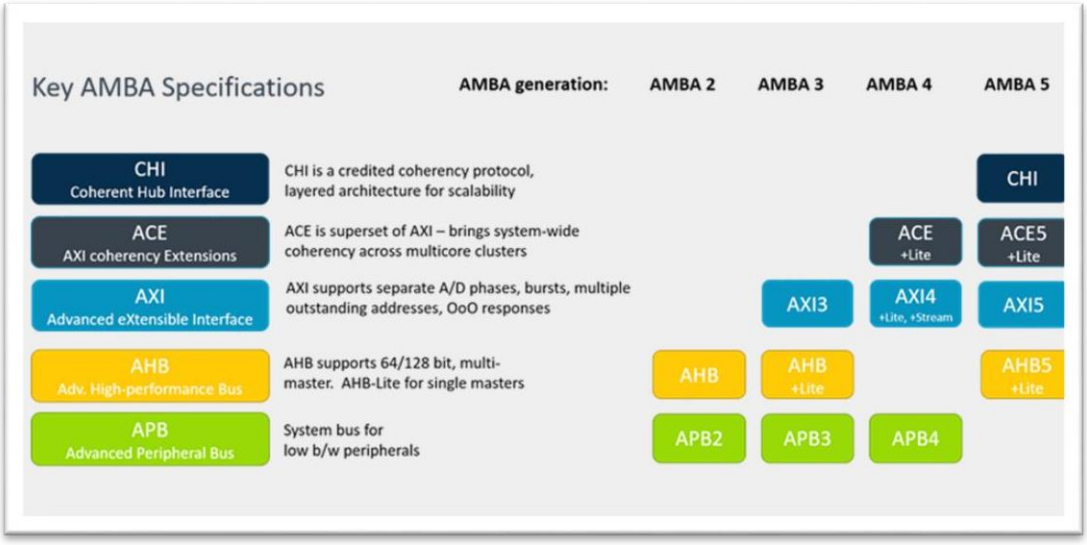
Ruthwika Vemulapalli

To

ABSTRACT

The Advanced Microcontroller Bus Architecture (AMBA) is an open System-on-Chip bus protocol for high performance buses to communicate with low-power devices. In the AMBA High-performance Bus (AHB) a system bus is used to connect a processor, a DSP, and high-performance memory controllers where as the AMBA Advanced Peripheral Bus (APB) is used to connect (Universal Asynchronous Receiver Transmitter) UART. It also contains a Bridge, which connects the AHB and APB buses. Bridges are standard bus-to-bus interfaces that allow IPs connected to different buses to communicate with each other in a standardized way.

ARM AMBA :



AMBA (Advanced Microcontroller Bus Architecture) is a freely-available, open standard for the connection and management of functional blocks in a system-on-chip (SoC). It facilitates right-first-time development of multi-processor designs, with large numbers of controllers and peripherals.

AMBA specifications are royalty-free, platform-independent and can be used withany processor architecture. Due to its widespread adoption, AMBA has a robust ecosystem of partners that ensures compatibility and scalability between IP components from different design teams and vendors.

AHB (AMBA High-performance Bus) :

AMBA AHB is a bus interface suitable for high-performance synthesizable designs. It defines the interface between components, such as masters, interconnects, and slaves.

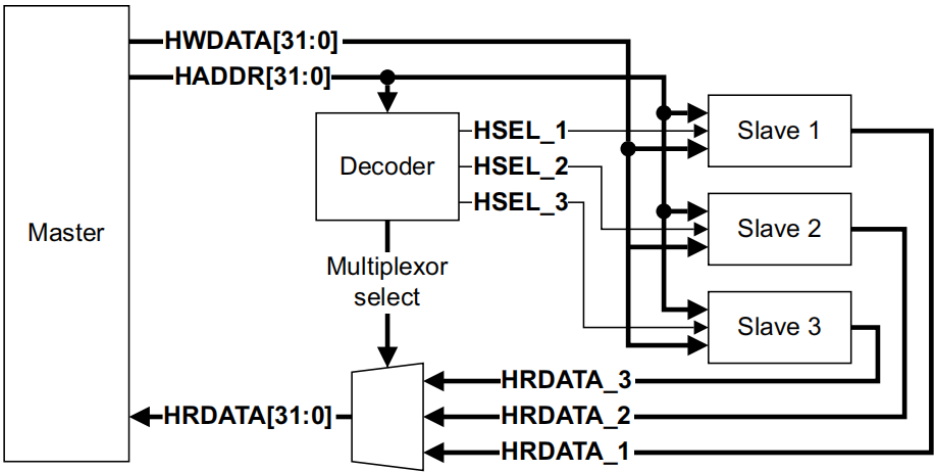
AMBA AHB implements the features required for high-performance, high clock frequency systems including:

- Burst transfers.
- Single clock-edge operation.
- Non-tristate implementation.
- Wide data bus configurations, 64, 128, 256, 512, and 1024 bits.

The most common AHB slaves are internal memory devices, external memory interfaces, and high-bandwidth peripherals. Although low-bandwidth peripherals can be included as AHB slaves, for system performance reasons, they typically reside on the AMBA Advanced Peripheral Bus (APB). Bridging between the higher performance AHB and APB is done using an AHB

slave, known as an APB bridge.

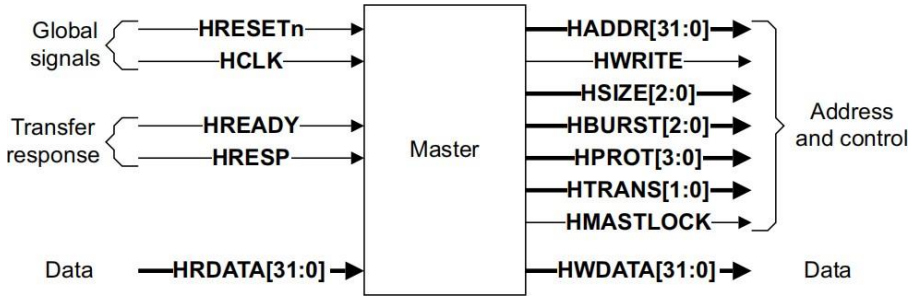
AHB (AMBA High-performance Bus) :



AHB block diagram

Figure shows a single master AHB system design with the AHB master and three AHB slaves. The bus interconnect logic consists of one address decoder and a slave-to-master multiplexor. The decoder monitors the address from the master so that the appropriate slave is selected and the multiplexor routes the corresponding slave output data back to the master. AHB also supports multi- master designs by the use of an interconnect component that provides arbitration and routing signals from different masters to the appropriate slaves.

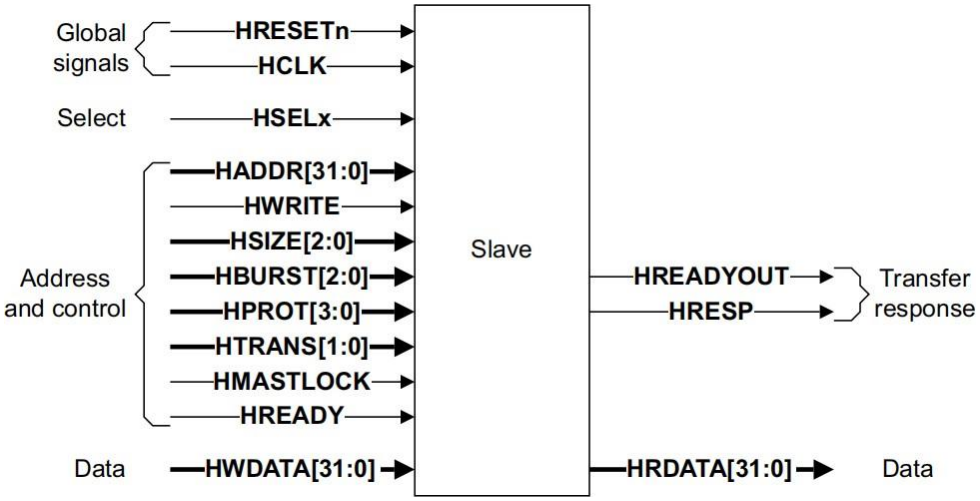
AHB Master Inteface :



AHB master interface

A master provides address and control information to initiate read and writeoperations.

AHB Slave Interface :



AHB slave interface

A slave responds to transfers initiated by masters in the system. The slave uses the HSELx select signal from the decoder to control when it responds to a bus transfer.

The slave signals back to the master:

- The completion or extension of the bus transfer.
- The success or failure of the bus transfer. Figure 1-3 shows a slave interface.

APB (Advanced Peripheral Bus) :

The Advanced Peripheral Bus (APB) is part of the Advanced Microcontroller Bus Architecture (AMBA) protocol family. It defines a low-cost interface that is optimized for minimal power

consumption and reduced interface complexity.

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:

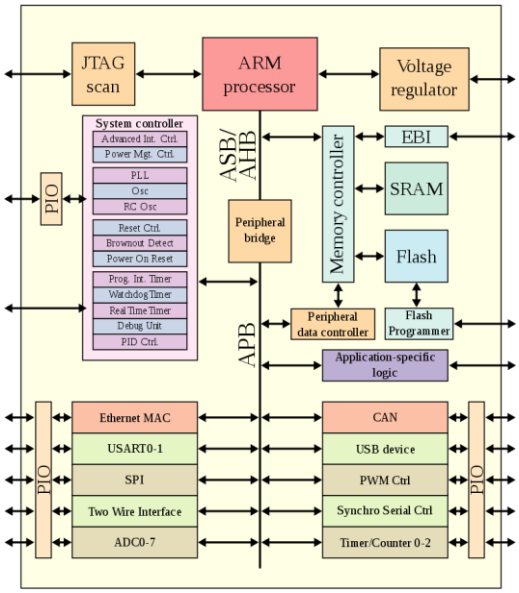
- AMBA Advanced High-performance Bus (AHB)
- AMBA Advanced High-performance Bus Lite (AHB-Lite)
- AMBA Advanced Extensible Interface (AXI)
- AMBA Advanced Extensible Interface Lite (AXI4-Lite)

You can use it to access the programmable control registers of peripheral devices.

AHB to APB Bridge :

The AHB to APB bridge interface is an AHB slave. When accessed (in normal operation or system test) it initiates an access to the APB. APB accesses are of different duration (three HCLK cycles in the EASY for a read, and two cycles for a write). They also have their width fixed to one word, which means it is not possible to write only an 8-bit section of a 32-bit APB register. APB peripherals do not need a PCLK input as the APB access is timed with an enable signal generated by the AHB to APB bridge interface. This makes APB peripherals low power consumption parts, because they are only strobed when accessed.

Importance :

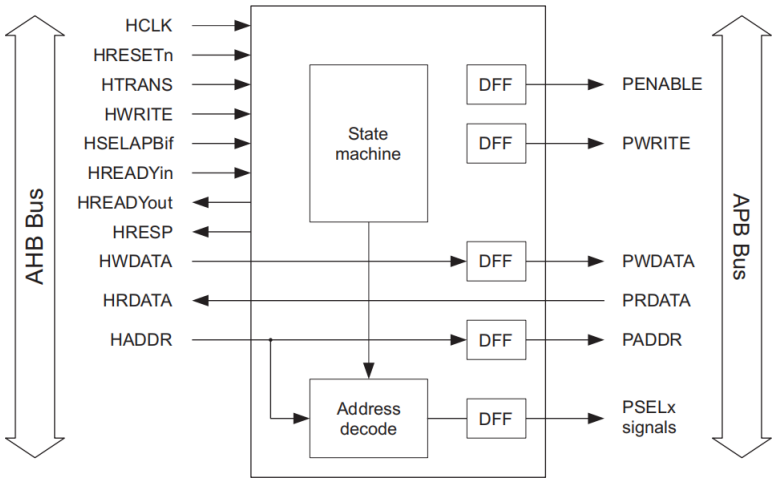


ARM Architecture

The AHB to APB bridge is an AHB slave, providing an interface between the high speed AHB and the low-power APB. Read and write transfers on the AHB are converted into equivalent transfers on the APB. It is required to bridge the communication gap between low bandwidth peripherals on APB with the high bandwidth ARM Processors and/or other high-speed devices on AHB. This ensures that there is no data loss between AHB to APB or APB to AHB data transfers. AHB2APB interfaces AHB and APB. It buffers address, controls and data from the AHB, drives the APB peripherals and return data along with response signal to the AHB.

The AHB2APB interface is designed to operate when AHB and APB clocks have the any combination of frequency and phase. TheAHB2APB performs transfer of data from AHB to APB for write cycle and APB to AHB for Read cycle. Interface between AMBA high performance bus (AHB) and AMBA peripheral bus (APB). It provides latching of address, controls and data signals for APB peripherals.

Architecture



AHB to APB Bridge block diagram

To add new APB peripherals, or alter the system memory map, only the address decode sections need to be modified.

The base addresses of each of the peripherals (timer, interrupt controller, and remap and pause controller) are defined in the AHB to APB bridge interface, which selects the peripheral according to its base address. The whole APB address range is also defined in the bridge.

These base addresses can be implementation-specific. The peripherals standard specifies only the register offsets (from an unspecified base address), register bit meaning, and minimum supported function

The APB data bus is split into two separate directions:

- read (PRDATA), where data travels from the peripherals to the bridge
- write (PWDATA), where data travels from the bridge to the peripherals.

This simplifies driving the buses because turnaround time between the peripherals and bridge is avoided.

In the default system, because the bridge is the only master on the bus, PWDATA is driven continuously. PRDATA is a multiplexed connection of all peripheral PRDATA outputs on the bus, and is only driven when the slaves are selected by the bridge during APB read transfers.

It is possible to combine these two buses into a single bidirectional bus, but precautions must be taken to ensure that there is no bus clash between the bridge and the peripherals.

Signals Description

Signals	Type	Direction	Description
HCLK	Bus clock	Input	This clock times all bus transfers.
HRESETn	Reset	Input	The bus reset signal is active LOW, and is used to reset the system and the bus.
HADDR[31:0]	Address bus	Input	The 32-bit system address bus.
HTRANS[1:0]	Transfer type	Input	This indicates the type of the current transfer, which can be NONSEQUENTIAL, SEQUENTIAL, IDLE or BUSY.
HWRITE	Transfer direction	Input	When HIGH this signal indicates a write transfer, and when LOW, a read transfer.
HWDATA[31:0]	Write data bus	Input	The write data bus is used to transfer data from the master to the bus slaves during write operations. A minimum data bus width of 32 bits is recommended. However, this may easily be extended to allow for higher bandwidth operation.
HSELAPBif	Slave select	Input	Each APB slave has its own slave select signal, and this signal indicates that the current transfer is intended for the selected slave. This signal is a combinatorial decode of the address bus.
HRDATA[31:0]	Read data bus	Output	The read data bus is used to transfer data from bus slaves to the bus master during read operations. A minimum data bus width of 32 bits is recommended. However, this may easily be extended to allow for higher bandwidth operation.

HREADYin HREADYout	Transfer done	Input/output	When HIGH the HREADY signal indicates that a transfer has finished on the bus. This signal may be driven LOW to extend a transfer.
HRESP[1:0]	Transfer response	Output	The transfer response provides additional information on the status of a transfer. This module will always generate the OKAY response.
PRDATA[31:0]	Peripheral read data bus	Input	The peripheral read data bus is driven by the selected peripheral bus slave during read cycles (when PWRITE is LOW).
PWDATA[31:0]	Peripheral write data bus	Output	The peripheral write data bus is continuously driven by this module, changing during write cycles (when PWRITE is HIGH).
PENABLE	Peripheral enable	Output	This enable signal is used to time all accesses on the peripheral bus. PENABLE goes HIGH on the second clock rising edge of the transfer, and LOW on the third (last) rising clock edge of the transfer.
PSELx	Peripheral slave select	Output	There is one of these signals for each APB peripheral present in the system. The signal indicates that the slave device is selected, and that a data transfer is required. It has the same timing as the peripheral address bus. It becomes HIGH at the same time as PADDR, but will be set LOW at the end of the transfer.

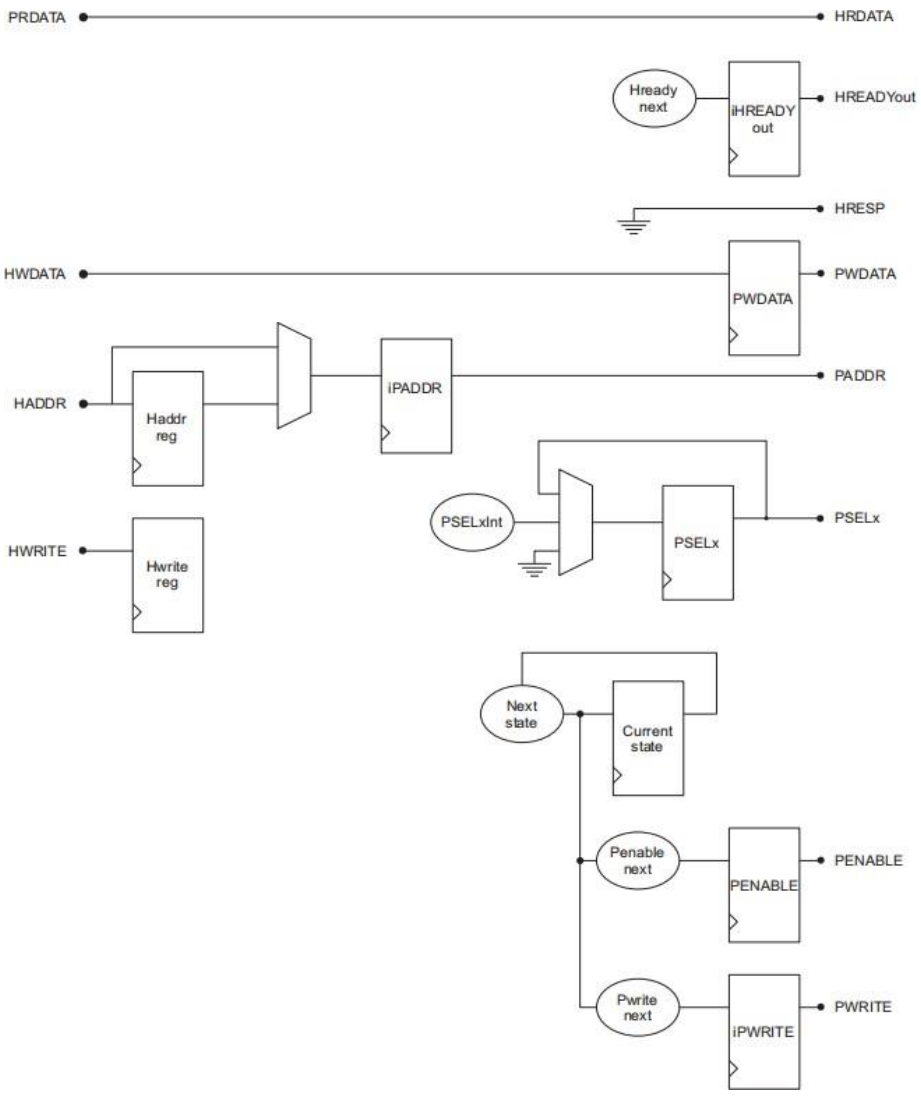
PADDR[31:0]	Peripheral address bus	Output	This is the APB address bus, which may be up to 32 bits wide and is used by individual peripherals for decoding register accesses to that peripheral. The address becomes valid after the first rising edge of the clock at the start of the transfer. If there is a following APB transfer, then the address will change to the new value, otherwise it will hold its current value until the start of the next APB transfer.
PWRITE	Peripheral transfer direction	Output	This signal indicates a write to a peripheral when HIGH, and a read from a peripheral when LOW. It has the same timing as the peripheral address bus.

The APB bridge responds to transaction requests from the currently granted AHB master. The AHB transactions are then converted into APB transactions.

- the AHB transactions with the HREADYout signal
- the generation of all APB output signals.

The individual PSELx signals are decoded from HADDR, using the state machine to enable the outputs while the APB transaction is being performed.
If an undefined location is accessed, operation of the system continues as normal, but no peripherals are selected.

Bridge Module Description



The AHB to APB bridge comprises a state machine, which is used to control the generation of the APB and AHB output signals, and the address decoding logic which is used to generate the APB peripheral select lines.

All registers used in the system are clocked from the rising edge of the system clock HCLK, and use the asynchronous reset HRESETn.

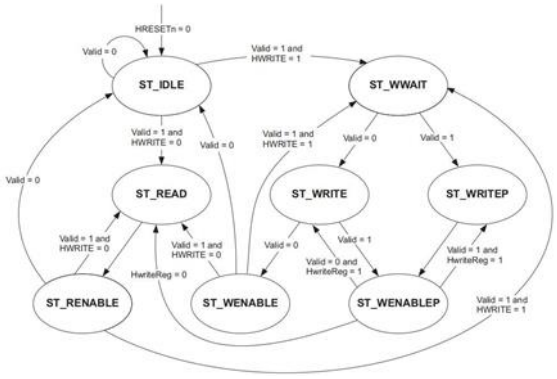
AHB slave bus interface :

This module uses the standard AHB slave bus interface, which comprises:

- the valid transfer detection logic which is used to determine when a valid transfer is accessing the slave.
- the address and control registers, which are used to store the information from the address phase of the transfer for use in the data phase.

Due to the different AHB to APB timing of read and write transfers, either the current or the previous address input value is needed to correctly generate the APB transfer. A multiplexor is therefore used to select between the current address input or the registered address, for read and write transfers respectively.

APB transfer state machine



The transfer state machine is used to control the application of APB transfers based on the AHB inputs. The state diagram in Figure shows the operation of the state machine, which is controlled by its current state and the AHB slave interface.

The AHB slave interface state machine operates through several key states:

1. **ST_IDLE**: APB buses hold their previous values, and PSEL/PENABLE are LOW, transitioning to ST_READ for read transfers or ST_WWAIT for write transfers.
2. **ST_READ**: The address is decoded, PSEL is HIGH, and PWRITE is LOW, with a wait state ensuring APB read data is ready before the AHB transfer completes.
3. **ST_WWAIT**: A wait state allows the AHB write transfer to complete, then moves to ST_WRITE.
4. **ST_WRITE**: Address and data are driven onto PADDR and HWDATA, with PSEL and PWRITE HIGH, completing without a wait state.
5. **ST_WRITEP**: Similar to ST_WRITE but with an inserted wait state, ensuring only one pending transfer between APB and AHB.

6. **ST_RENABLE**: PENABLE is driven HIGH, completing the APB read transfer and transitioning based on further transfers.
7. **ST_WENABLE**: PENABLE is HIGH, completing the APB write transfer and transitioning based on further transfers.
8. **ST_WENABLEP**: A wait state is inserted if a read follows a write, handling pending transfers efficiently.

Bridge Functional Description

APB output signal generation

The generation of all APB output signals is based on the status of the transfer state machine:

- **PWDATA** is a registered version of the HWDATA input, which is only enabled during a write transfer. As the bridge is the only bus master on the APB, then it can drive PWDATA continuously.
- **PENABLE** is only set HIGH during one of three enable states, in the last cycle of an APB transfer. A register is used to generate this output from the next state of the transfer state machine.
- **PSELx** outputs are decoded from the current transfer address. They are only valid during the read, write and enable states, and are all driven LOW at all other times so that no peripherals are selected when no transfers are being performed.
- **PADDR** is a registered version of the currently selected address input (HADDR or the address register) and only changes when the read and write states are entered at the start of the APB transfer.
- **PWRITE** is set HIGH during a write transfer, and only changes when a new APB transfer is started. A register is used to generate this output from the next state of the transfer state machine.
- The APBen signal is used as an enable on the PSEL, PWRITE and PADDR output registers, ensuring that these signals only change when a new APB transfer is started, when the next state is ST_READ, ST_WRITE, or ST_WRITEP.
-

AHB output signal generation

HRDATA is directly driven with the current value of PRDATA. APB slaves only drive read data during the enable phase of the APB transfer, with PRDATA set LOW at all other times, so bus clash is avoided on HRDATA (assuming OR bus connections for both the AHB and APB read data buses).

- **HREADYout** is driven with a registered signal to improve the output timing. Wait states are inserted by the APB bridge during the ST_READ and ST_WRITEP states, and during the ST_WENABLEP state when the next transfer to be performed is a read.
- **HRESP** is continuously held LOW, as the APB bridge does not generate SPLIT, RETRY or ERROR responses.

AHB to APB bridge design code :

```
`define IDLE 3'b000
`define READ 3'b001
`define WWAIT 3'b010
`define WRITE 3'b011
`define WRITEP 3'b100
`define WENABLE 3'b101
`define WENABLEP 3'b110
`define RENABLE 3'b111
`timescale 1ns/1ps
module ahb2apb(HCLK, HRESETn, HSELAPB, HADDR, HWRITE, HTRANS,HWDATA,
HRESP, HRDATA, HREADY, PRDATA, PSEL, PENABLE, PADDR, PWRITE, PWDATA);
//_____AHB Slave Interface_____
input wire HCLK, HRESETn, HSELAPB, HWRITE;
input wire [1:0]HTRANS;
input wire [31:0]HADDR, HWDATA;output
reg HRESP, HREADY;
output reg [31:0]HRDATA;
//____APB Output Signals__ input wire [31:0]PRDATA;
output reg PSEL, PENABLE, PWRITE; output reg [31:0]PADDR, PWDATA;
//_____Implementation signals_____ reg [31:0]TMP_HADDR, TMP_HWDATA;
reg [2:0] ps,ns; reg valid, HWrite;
always @(*) begin
//____Valid logic___
if (HSELAPB==1'b1 && (HTRANS==2'b10 || HTRANS==2'b11))
valid=1'b1; else
valid=1'b0;
if(HRESETn==1'b0) //Asynchronous Active-low Reset ns=`IDLE;
HRESP=1'b0; //Always OKAY Response end
always @(posedge HCLK) begin
ps=ns; end
```

```
always @(ps)begin case(ps)
      `IDLE :
      begin
      PSEL=1'b0;
        PENABLE=1'b0;
        HREADY=1'b1;
        if(valid==1'b0)
          ns=`IDLE;
        else if(valid==1'b1 && HWRITE==1'b0)
          ns=`READ;
        else if(valid==1'b1 && HWRITE==1'b1)
          ns=`WWAIT;
      end
      `READ :
      begin
      PSEL=1'b1;
        PADDR=HADDR;
        PWRITE=1'b0;
        PENABLE=1'b0;
        HREADY=1'b0;
        ns=`RENABLE;
      end
     WWAIT :
    begin PENABLE=1'b0;
   TMP_HADDR=HADDR; HWrite=HWRITE;
   if(valid==1'b0) ns=`WRITE;
   else if(valid==1'b1) ns=`WRITEP;
   end
`WRITE :
  begin PSEL=1'b1;
 PADDR=TMP_HADDR;        PWDATA=HWDATA;        PWRITE=1'b1;        PENABLE=1'b0;
HREADY=1'b0;
 if(valid==1'b0) ns=`WENABLE;
else if(valid==1'b1) ns=`WENABLEP;
end
`WRITEP :
begin
PSEL=1'b1; PADDR=TMP_HADDR; PWDATA=HWDATA; PWRITE=1'b1; PENABLE=1'b0;
HREADY=1'b0; TMP_HADDR=HADDR; HWrite=HWRITE; ns=`WENABLEP;
end
`WENABLE :
begin PENABLE=1'b1; HREADY=1'b1;
if(valid==1'b1 && HWRITE==1'b0) ns=`READ;
else if(valid==1'b1 && HWRITE==1'b1) ns=`WWAIT;
else if(valid==1'b0) ns=`IDLE;
end
`WENABLEP :
begin PENABLE=1'b1; HREADY=1'b1;
```

```
if(valid==1'b0 && HWrite==1'b1) ns=`WRITE;
else if(valid==1'b1 && HWrite==1'b1) ns=`WRITEP;
else if(HWrite==1'b0) ns=`READ;
end
`RENABLE :
begin PENABLE=1'b1;
HRDATA=PRDATA; HREADY=1'b1;
if(valid==1'b1 && HWRITE==1'b0) ns=`READ;
else if(valid==1'b1 && HWRITE==1'b1) ns=`WWAIT;
else if(valid==1'b0) ns=`IDLE;
end endcase end
endmodule
```

In the AHB to APB bridge design module, I have used some local signals. The description of these signals is given below :

1. ns : Refers to the next state.

2. ps : Refers to the present state.

3. valid : The bridge only performs operation if the valid signal is high, when the bridge is selected by HSELAPB signal and also the AHB transfer type is NONSEQ or SEQ.

4. TMP_HADDR, TMP_HWRITE : Both the signals holds the address and the transfer direction information of the pending transfer respectively.

3.1 Testbench Code :

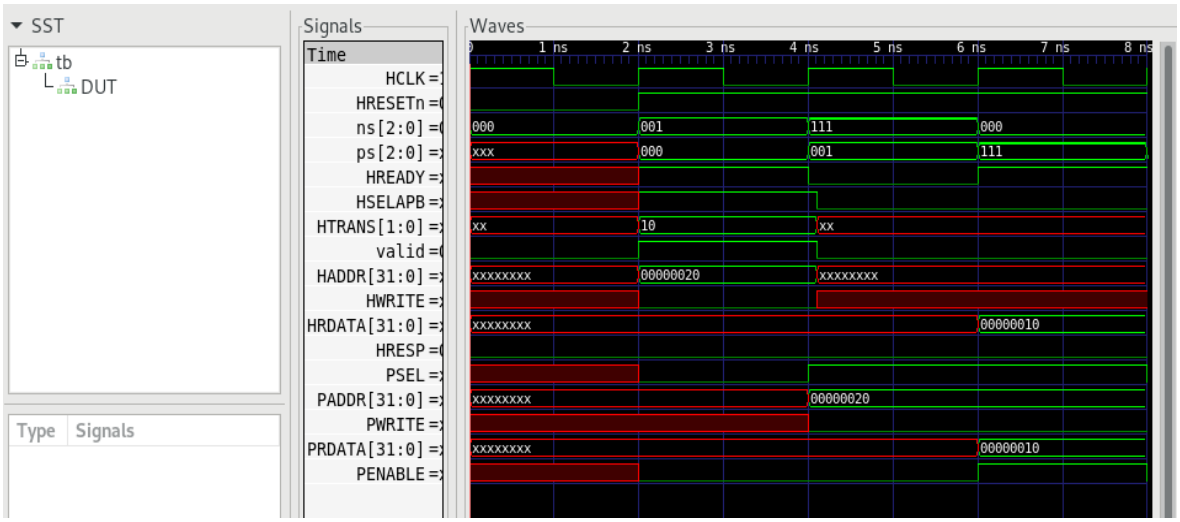
```
`timescale 1ns/1ps module tb;
//_____AHB Slave Interface_ reg HCLK, HRESETn, HSELAPB, HWRITE;
reg [1:0]HTRANS;
reg [31:0]HADDR, HWDATA;
wire HRESP;
wire [31:0]HRDATA;
//_____APB Output Signals__                    reg [31:0]PRDATA;
wire PSEL, PENABLE, PWRITE, HREADY; wire [31:0]PADDR, PWDATA;
always #1 HCLK=~HCLK;

`ifdef Single_Read initial
begin
$dumpfile("Single_Read.vcd");
$dumpvars; end
initial begin
//_____Single Read Transfer_ HCLK=1'b1;
HRESETn=1'b0;
#2 HRESETn=1'b1; HWRITE=1'b0; HSELAPB=1'b1; HTRANS=2'b10; HADDR=32;
#2.1 HWRITE=1'bx; HSELAPB=1'b0; HTRANS=2'bxx;
HADDR=32'hxxxx_xxxx; #1.9 PRDATA=16;
#2 $finish; end
`endif
```

```
`ifdef Single_Write initial
begin
$dumpfile("Single_Write.vcd");
$dumpvars;
end
initial begin
//_____Single WRITE Transfer_____ HCLK=1'b1;
HRESETn=1'b0;
#2 HRESETn=1'b1; HWRITE=1'b1; HSELAPB=1'b1; HTRANS=2'b10; HADDR=32'h0000_0000;
#2 HWDATA=32'h0000_00ff; HSELAPB=1'b0;
#0.1 HWRITE=1'bx; HTRANS=2'bxx;
HADDR=32'hxxxx_xxxx; #6 $finish;
end
`endif
`ifdef Burst_Read initial
begin
$dumpfile("Burst_Read.vcd");
$dumpvars; end
initial begin
//_____Burst Read Transfer__ HCLK=1'b1;
HRESETn=1'b0;
#2       //IDLE State
HRESETn=1'b1; HWRITE=1'b0; HSELAPB=1'b1; HTRANS=2'b10; HADDR=32'h0000_0000;
#2.1     //READ State HTRANS=2'b11; HADDR=32'h0000_0100;
#1.9     //RENABLE State PRDATA=32'hFFFF_FFFF;
#2.1
HADDR=32'h0000_1000; #1.9
    PRDATA=32'hFFFF_FFFB; #2.1 HADDR=32'h0000_1100; #1.9 PRDATA=32'hFFFF_FFF8;
#2.1
    HWRITE=1'bx;
   HADDR=32'hxxxx_xxxx; HTRANS=2'bxx; HSELAPB=1'bx;
#1.9
   PRDATA=32'hFFFF_FFF4;
#6 $finish; end
`endif
`ifdef Burst_Write initial
begin
$dumpfile("Burst_Write.vcd");
$dumpvars; end
initial begin
//_____Burst WRITE Transfer_____
HCLK=1'b1; HRESETn=1'b0;
#2       //IDLE State
HRESETn=1'b1; HWRITE=1'b1; HSELAPB=1'b1; HTRANS=2'b10; HADDR=32'h0000_0000;
#2.1     //WWAIT State HWDATA=32'h0000_000F; HADDR=32'h0000_0100; HTRANS=2'b11;
#2       //WRITEP State HWDATA=32'h0000_00F0; HADDR=32'h0000_1000;
#4;      //WENABLE State HWDATA=32'h0000_0F00; HADDR=32'h0000_1100;
#4
```

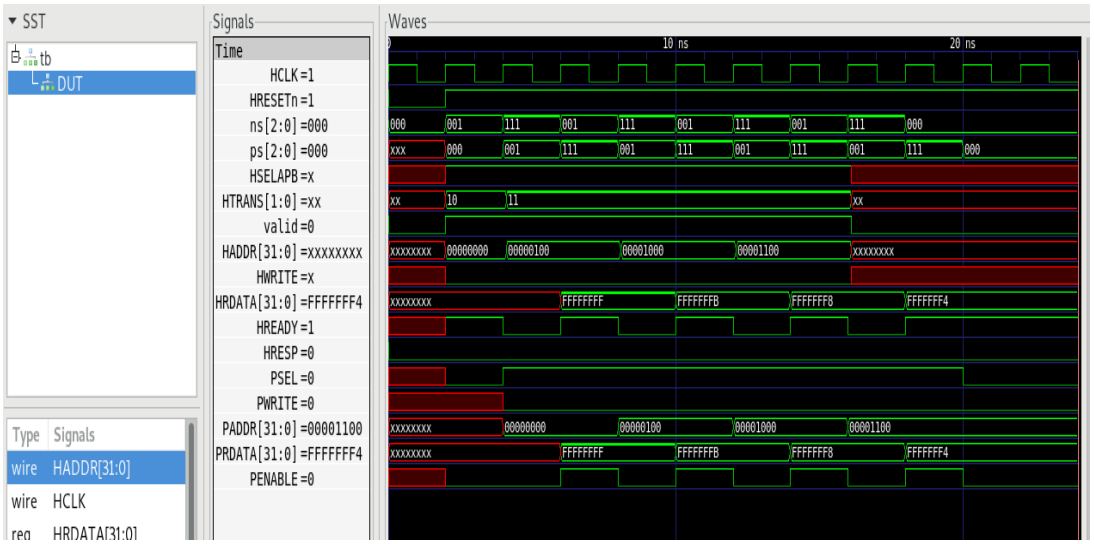
```
HWDATA=32'h0000_F000;
HADDR=32'hxxxx_xxxx; HWRITE=1'bx; HSELAPB=1'bx; HTRANS=2'bxx;
#4
HWDATA=32'hxxxx_xxxx; #8 $finish;
end
`endif
ahb2apb DUT(HCLK, HRESETn, HSELAPB, HADDR, HWRITE, HTRANS, HWDATA, HRESP,
HRDATA, HREADY, PRDATA, PSEL, PENABLE, PADDR, PWRITE, PWDATA);
endmodule
```

Simulation Results

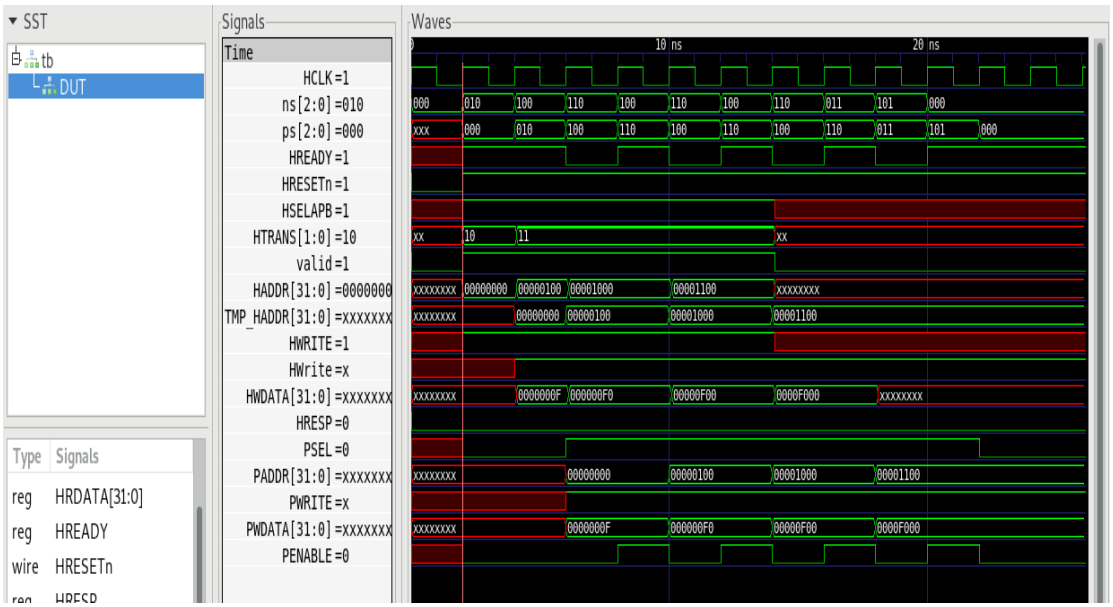


Single read transfer

Single write Transfer



Burst read transfer



Burst write transfer

Conclusion

The development of the synthesizable AHB to APB Bridge in verilog HDL was done. The HCLK and PENABLE mechanism was implemented for making it the low-power consuming system. The functional verification of the bridge was done by driving various testcases to the design for testing the features. The multimaster and multislave AHB to APB bridge is one of the futurescope