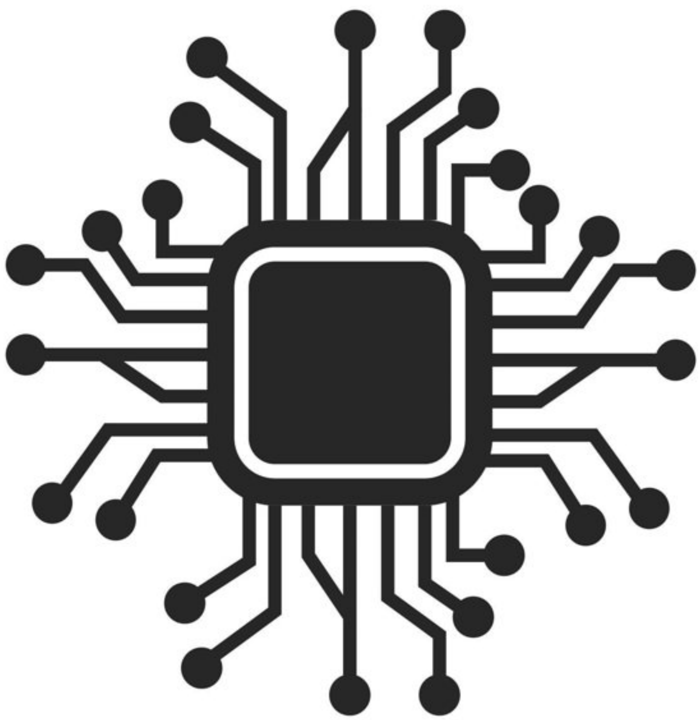
AMBA® APB Bus System Design



Prepared by

Omar Mohamed Hussein Mostafa

Abstract

This report begins by presenting an overview of the AMBA Advanced Peripheral Bus (APB) protocol, describing its operation, transaction phases, and suitability for simple, low-power peripheral communication. The fundamental concepts of address, control, and data transfer within the Setup and Access phases are explained to establish a clear understanding of the protocol. Following this, the report details the design and implementation of a complete APB system consisting of a master and two peripherals: a RAM memory and a one-shot timer. The integration is achieved through an address-decoding wrapper module, demonstrating the application of the APB protocol in building modular and extensible system-on-chip components.

Table of Contents

1. INTRODUCTION 5

2. AMBA APB SIGNALS 7

3. AMBA APB OPERATION STATES 8

4. TRANSFERS 9

- 4.1. WRITE STATE WITH NO WAITS 9
- 4.2. WRITE STATE WITH WAIT STATES 10
- 4.3. READ STATE WITH NO WAITS 11
- 4.4. READ STATE WITH WAITS 11
- 4.5. WRITE STROBES 12
- 4.6. PROTECTION 12
- 4.7. ERROR 13

5. PROJECT USING THE AMBA APB PROTOCOL WITH A MASTER AND 2 SLAVES: RAM SLAVE AND TIMER SLAVE. 14

- 5.1. PROJECT OVERVIEW 14
- 5.2. MASTER MODULE 15
  - 5.2.1. PARAMETERS 15
  - 5.2.2. SIGNALS 16
- 5.3. DECODER MODULE 17
  - 5.3.1. PARAMETERS 17
  - 5.3.2. SIGNALS 17
- 5.4. APB RAM 18
  - 5.4.1. PARAMETERS 18
  - 5.4.2. SIGNALS 18
- 5.5. APB TIMER SLAVE 19
  - 5.5.1. PARAMETERS IN THE TIMER SLAVE MODULE 19
  - 5.5.2. SIGNALS 19
- 5.6. WRAPPED MODULE 20
  - 5.6.1. PARAMETERS IN THE WRAPPER MODULE 20
  - 5.6.2. SIGNALS 20
- 5.7. FLOW SEQUENCE 20
- 5.8. VERILOG CODE 22
  - 5.8.1. RTL CODE FOR MASTER 22
  - 5.8.2. RTL CODE FOR DECODER 24
  - 5.8.3. RTL CODE FOR THE APB RAM MEMORY SLAVE 25
  - 5.8.4. RTL CODE FOR THE APB TIMER SLAVE 26
  - 5.8.5. RTL CODE FOR THE WRAPPED MODULE 27
  - 5.8.6. TESTBENCH CODE 29
- 5.9. QUESTASIM SIMULATION 31
  - 5.9.1. APB RAM MEMORY 31
  - 5.9.2. APB TIMER 31

5.10. SYNTHESIS TOOL 32
5.10.1. ELABORATION 32
5.10.2. SYNTHESIS 34

Table of Figures

Figure 1 AMBA architecture with AHB for high-speed and APB for peripherals via bridge. 5
Figure 2 Block diagram of the APB communication protocol with connected peripherals. 6
Figure 3 APB master communication with APB slave 8
Figure 4 State Diagram for the APB Master 8
Figure 5 Write transfer with no wait states 9
Figure 6 Write transfer with wait states. 10
Figure 7 Read transfer with no wait states 11
Figure 8 Read transfer with wait states 11
Figure 9 Byte Lane mapping..12
Figure 10 Block diagram for the master and the slave communication 21

1.Introduction

The AMBA Advanced Peripheral Bus (APB) is widely used for low-power, low-bandwidth communication with simple peripherals such as UARTs, timers, and keypad interfaces. Unlike the Advanced High-performance Bus (AHB), which is designed for higher bandwidth and performance, the APB provides a lightweight interface that reduces complexity and power consumption.

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)

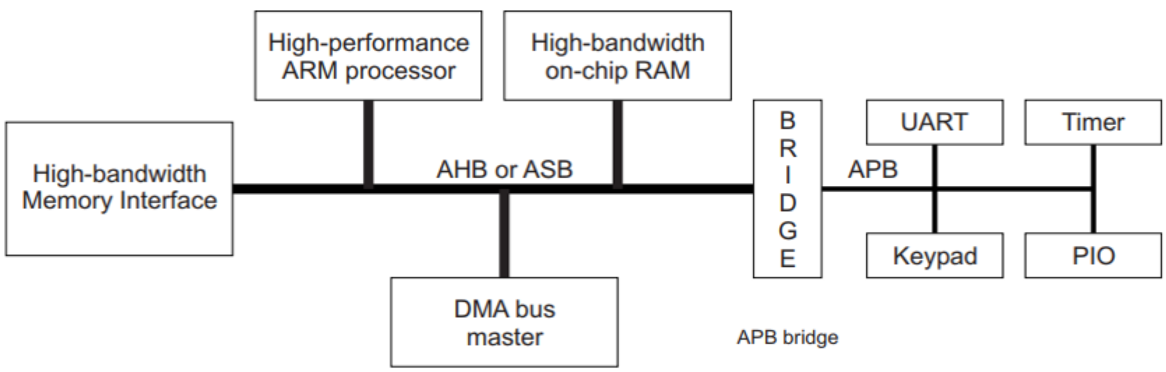


Figure 1 AMBA architecture with AHB for high-speed and APB for peripherals via bridge.

APB is a non-pipelined protocol, meaning each transfer is completed before the next one begins. This makes it suitable for low-bandwidth peripherals that only require simple read or write operations.

The APB bus consists of an APB master/bridge and multiple peripheral slaves. The master selects a target slave through selection lines and performs a read or write operation. Each transfer requires at least two clock cycles to complete.

Block diagram illustrating the APB protocol with an APB master/bridge, and peripheral slaves.

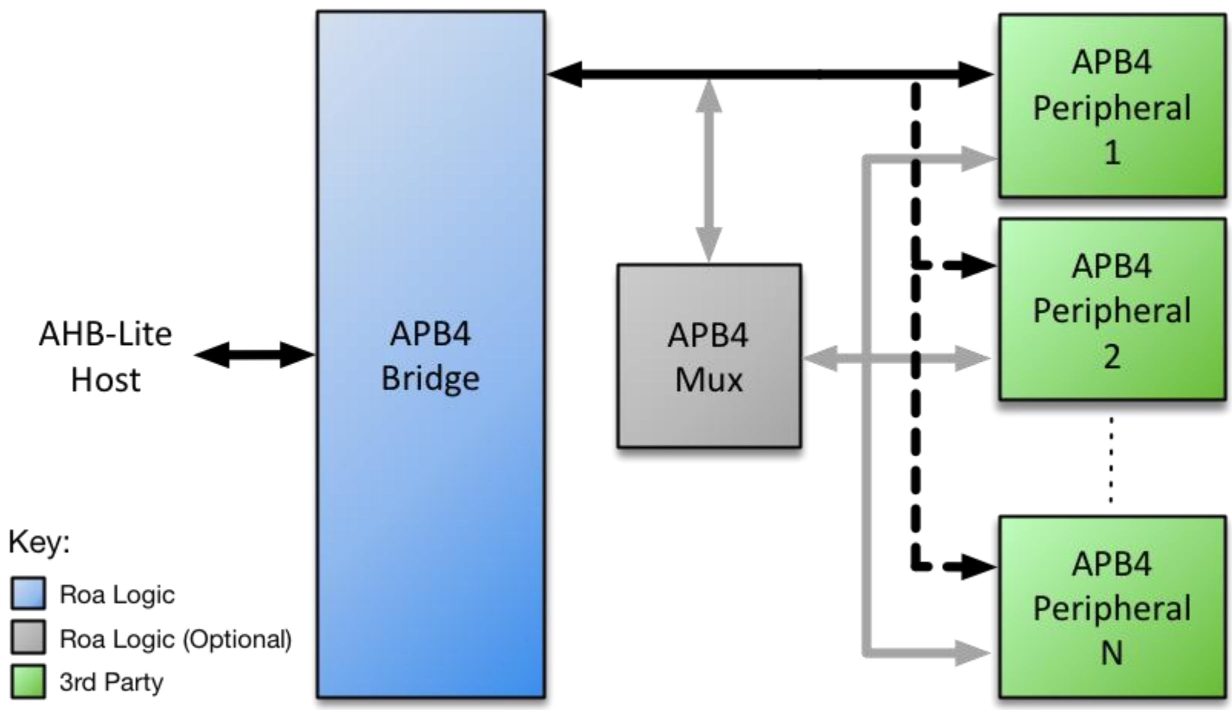


Figure 2 Block diagram of the APB communication protocol with connected peripherals.

2. AMBA APB Signals

Signal	Source	Description
PCLK	Clock Source	Clock. The rising edge of PCLK times all transfers on the APB.
PRESET_n	System Bus equivalent	Reset. The APB reset signal is synchronous active LOW
PADDR	APB Bridge	Address. This is the APB address bus. It can be up to 32 bits wide.
PPROT	APB Bridge	Protection type. This signal indicates the normal, privileged, or secure protection level of the transaction and whether the transaction is data access or an instruction access
PSELx	APB Bridge	Select. The APB bridge unit generates this signal to each peripheral bus slave. It indicates that the slave device is selected and that a data transfer is required. There is a PSELx signal for each slave.
PENABLE	APB Bridge	Enable. This signal indicates the second and subsequent cycles of an APB transfer.
PWIRTE	APB Bridge	Direction. This signal indicates an APB write access when HIGH and an APB read access when LOW.
PWDATA	APB Bridge	Write data. This bus is driven by the peripheral bus bridge unit during write cycles when PWRITE is HIGH. This bus can be up to 32 bits wide.
PSTRB	APB Bridge	Write strobes. This signal indicates which byte lanes to update during a write transfer.
PREADY	Slave Interface	Ready. The slave uses this signal to extend an APB transfer.
PRDATA	Slave Interface	Read Data. The selected slave drives this bus during read cycles when PWRITE is LOW. This bus can be up to 32-bits wide.
PSLVERR	Slave Interface	This signal indicates a transfer failure. APB peripherals are **not required** to support the PSLVERR pin.

3.AMBA APB Operation States

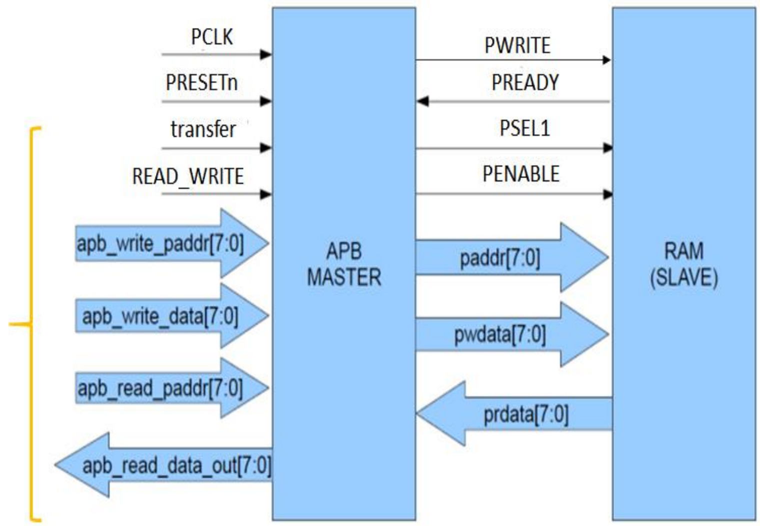


Figure 3 APB master communication with APB slave

The APB master is triggered externally by a transfer request signal. Once triggered, it initiates a transaction with the selected peripheral using the address and data buses.

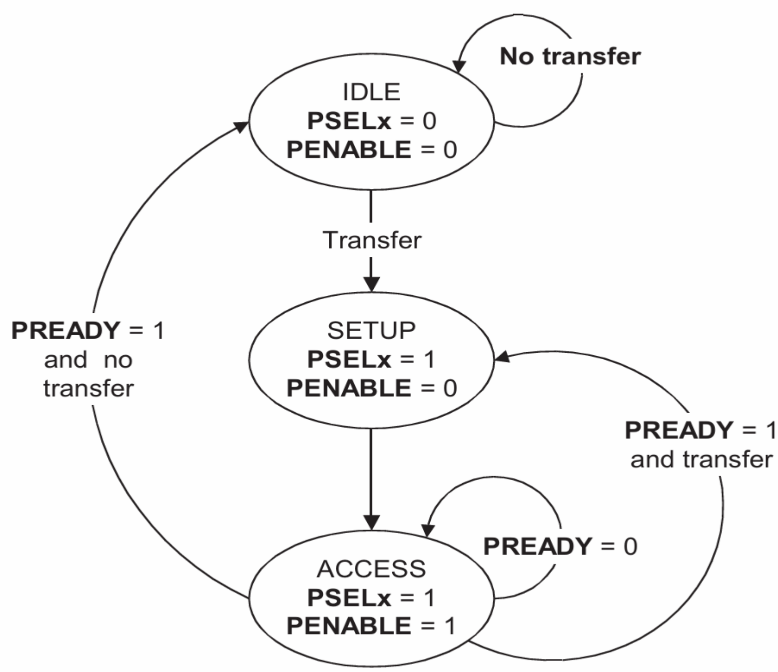


Figure 4 State Diagram for the APB Master

The AMBA APB has three states:

- **IDLE:** No peripheral is selected, and no transfer takes place.
- **IDLE to Setup:** This transition occur when the Transfer request is sent.
- **SETUP:** The APB master selects the target slave, sets the address, and determines the operation (read or write). These control signals are prepared for the peripheral.
- **SETUP to ACCESS:** this transition occur in the second cycle with rising the PENABLE.
- **ACCESS:** The master asserts PENABLE to enable the transfer. At the next rising edge, if the slave indicates PREADY, the transfer is completed.

4.Transfers

4.1.Write state with no waits

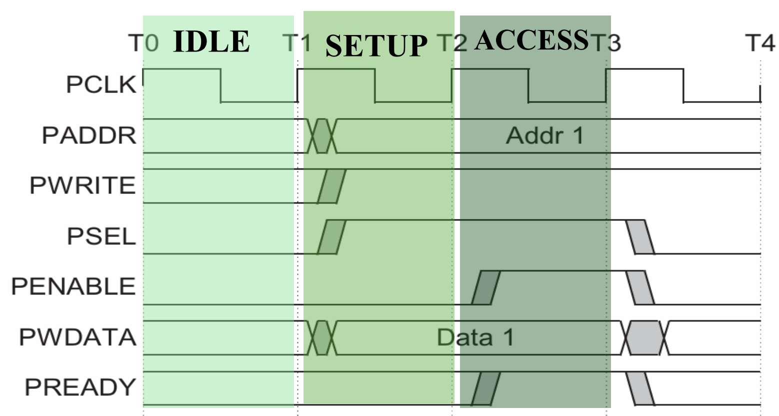


Figure 5 Write transfer with no wait states

- At the **IDLE** state between T0 and T1 the master receives a request through the transfer signal.
- At T1, a write transfer starts with address **PADDR**, write data **PWDATA**, write signal **PWRITE**, and select signal **PSEL**, being registered at the rising edge of PCLK. This is called the Setup phase of the write transfer.
- At T2, signal **PENABLE**, and ready signal **PREADY**, are registered at the rising edge of **PCLK**. When asserted, **PENABLE** indicates the start of the Access phase of the transfer. When asserted, **PREADY** indicates that the slave can complete the transfer at the **next rising edge of PCLK**.

The address **PADDR**, write data **PWDATA**, and control signals all remain valid until the **<u>transfer completes at T3</u>**, the end of the Access phase. The enable signal PENABLE is deasserted at the end of the transfer. The select signal PSEL, is also deasserted unless the transfer is to be followed immediately by another transfer to the same peripheral.

4.2.Write state with wait states

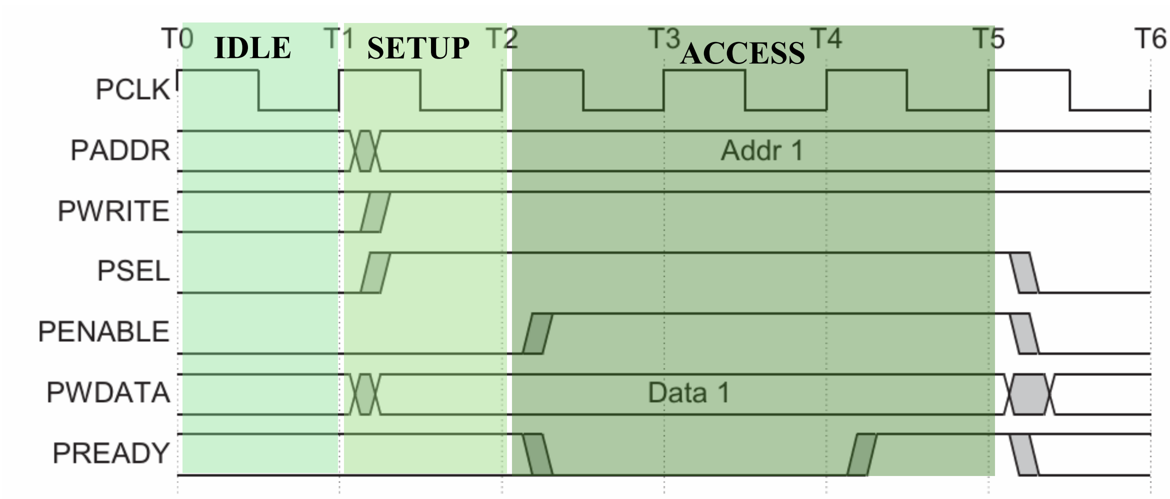


Figure 6 Write transfer with wait states.

- The same happens like the previous write state except when PENABLE is HIGH, the slave extends the transfer by driving **PREADY** LOW. All the signals remain unchanged while **PREADY** remains LOW.

4.3.Read state with no waits

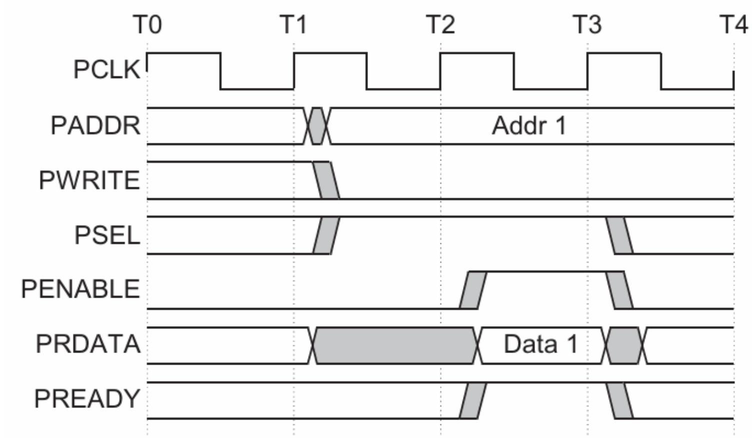


Figure 7 Read transfer with no wait states

- The same process as the write state except for the **PWRITE** is held LOW.

4.4.Read state with waits

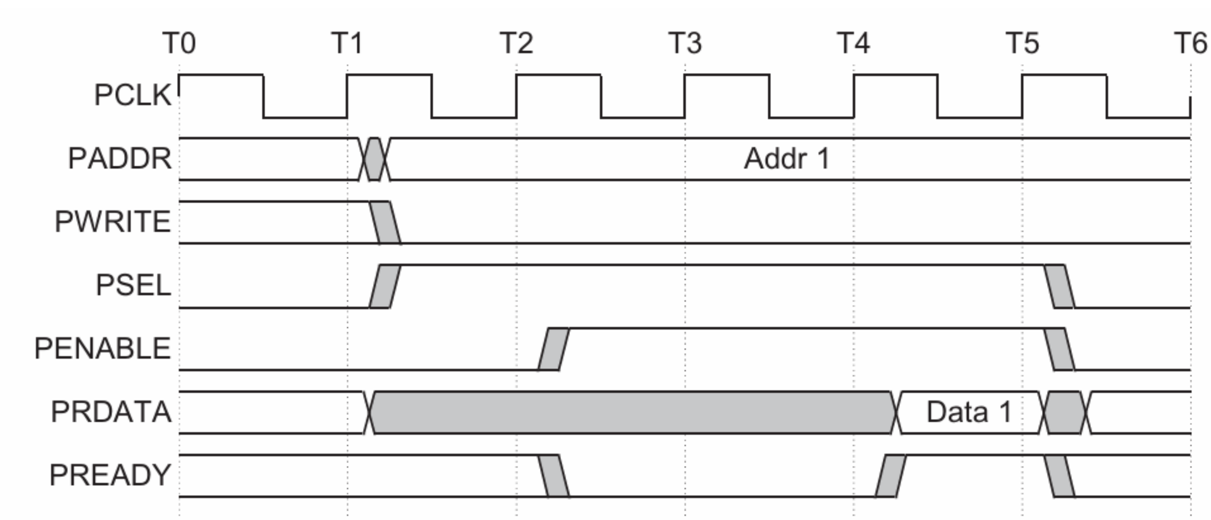


Figure 8 Read transfer with wait states

- The same process as the write state with wait cycles except for the **PWRITE** is held LOW.

✓ **<u>Comment:</u>** **PREADY** can take any value when **PENABLE** is LOW. This ensures that peripherals that have a fixed two cycle access can tie **PREADY** HIGH.

4.5.Write Strobes

31 24	23 16	15 8	7 0
PSTRB[3]	PSTRB[2]	PSTRB[1]	PSTRB[0]

Figure 9 Byte Lane mapping

The write strobe signals, **PSTRB**, enable sparse data transfer on the write data bus. Each write strobe signal corresponds to one byte of the write data bus. When asserted HIGH, a write strobe indicates that the corresponding byte lane of the write data bus contains valid information. There is one write strobe for each eight bits of the write data bus, so **PSTRB[n]** corresponds to **PWDATA[(8n + 7) :(8n)]**. Figure 9 shows this relationship on a 32-bit data bus where in this case the **PSTRB** would be a 4-bit bus, each bit enables to write a certain byte into the same byte index in the slave register.

4.6.Protection

Protection unit support, supports **complex system designs**, it is often necessary for both the interconnect and other devices in the system to provide protection against illegal transactions. For the APB interface, this protection is provided by the **PPROT[2:0]** signals.

The three levels of access protection are:

- **<u>Normal or privileged PPROT[0]</u>**
  - LOW indicates a normal access.
  - HIGH indicates a privileged access.
  - This is used by some masters to indicate their processing mode.
  - A privileged processing mode typically has a greater level of access within a system.
- **<u>Secure or non-secure, PPROT[1]</u>**
  - LOW indicates a secure access.
  - HIGH indicates a non-secure access.
  - This is used in systems where a greater degree of differentiation between processing modes is required.

  - Note This bit is configured so that when it is HIGH then the transaction is considered non-secure and when LOW, the transaction is considered as secure.
- **Data or Instruction, PPROT[2]**
  - LOW indicates data access.
  - HIGH indicates instruction access.
  - This bit gives an indication if the transaction is a data or instruction access.
  - Note This indication is provided as a hint and is not accurate in all cases. For example, where a transaction contains a mix of instruction and data items. It is recommended that, by default, an access is marked as a data access unless it is specifically known to be an instruction access.

4.7.Error

PSLVERR is the error response signal in the APB protocol. You use it only in the last cycle of a transfer when **PSEL**=1, **PENABLE**=1, and **PREADY**=1, this means that otherwise in this case the **PSLVERR** is ignored so it's recommended to be held LOW when it's not sampled. This signal indicates that something went wrong during the transfer. For Example:

- **Invalid address access:** The master tried to access a register that doesn't exist in the slave.
- **Unsupported operation:** writing to a read-only register.
- **Peripheral internal error:**The slave cannot complete the requested operation correctly.

PSLVERR must only be valid in the final cycle of the transfer. If the slave drives **PSLVERR**= 1, the master considers the transfer unsuccessful.

APB peripherals are not required to support the **PSLVERR** pin. This is true for both existing and new APB peripheral designs. Where a peripheral does not include this pin then the appropriate input to the APB bridge is tied LOW.

5.Project using the AMBA APB protocol with a Master and 2 Slaves: Ram slave and Timer slave.

5.1.Project Overview

The designed system implements the AMBA APB protocol with one master and two slaves: a timer and a memory module. The master generates all APB control signals and supports both read and write operations following the standard Setup and Access phases. A parameterized decoder is used to map address ranges to the corresponding slave select signals, ensuring only one slave is active at a time. The timer slave provides a programmable down-counter with registers for load, current value, control, and interrupt clear, supporting one-shot operation and interrupt generation when the count reaches zero. The memory slave acts as a small APB-mapped register file, parameterized in width and depth, and supports word-aligned read and write operations with byte strobe support. Address and data widths across modules are fully parameterized, with the timer using a 2-bit address space and the memory using a 6-bit address space. The overall design is compliant with the AMBA APB protocol, synthesizable, and verified in simulation.

This Project ignores the PSLVERR and the PPROT signals.

5.2.Master module

The APB master block is responsible for initiating and controlling all bus transactions with the connected slave peripherals. It generates the required APB signals, including the address bus (PADDR), write data (PWDATA), read/write control (PWRITE), and select/enable strobes (PSEL, PENABLE), while receiving read data (PRDATA) and status (PREADY, PSLVERR) from the slaves. The master follows the APB protocol sequence of IDLE → SETUP → ACCESS states to ensure proper communication. Addressing is implemented as byte-addressable, meaning each address refers to an individual byte, and word-aligned accesses are performed internally based on the data bus width. This design enables the master to interface seamlessly with both memory and timer slaves, managing read and write operations in compliance with the AMBA APB standard.

The APB master module in this design uses a byte-addressable addressing scheme rather than word addressing. This means that each increment in the address corresponds to a single byte in memory, regardless of the data bus width. For example, with a 32-bit (4-byte) data bus, consecutive word-aligned addresses differ by 4 (0x00, 0x04, 0x08, …). This convention is consistent with standard ARM-based APB implementations and allows finer granularity when accessing memory and peripheral registers, while the slaves internally decode only the relevant word-aligned addresses.

5.2.1.Parameters

Parameters	Accessibility	Default	Description
DATA_WIDTH	Global	32	Input Data Width, up to 32 bits, also defines the memory Word width.
SLAVE_NUM	Global	2	Number of slaves derived by the master.
MAIN_ADDR_WIDTH	Global	32	The main address width driven by the master.
DATA_BYTE_NUM	Local	4	Byte number in the DATA bus.
IDLE	Local	0	Reset State
SETUP	Local	1	Setup state
ACCESS	Local	2	Access state

5.2.2.Signals

Signals	Port	Width	Description
Transfer	Input	1	Transfer signal from the higher logic to start operation.
PCLK	Input	1	System clock.
PRESET_n	Input	1	System reset.
PADDR	Output	32	Address driven to the slave.
PSEL	Output	2	Selection signal to choose the slave.
PENABLE	Output	1	Enable signal.
PWRITE	Output	1	Write/Read signal , when HIGH indicates write.
PWDATA	Output	32	Data input to be written.
PSTRB	Output	4	Strobe signal to validate certain bytes of the data input.
PREADY	Input	1	Ready signal to complete transfer.
PRDATA	Input	32	Data read from the slave.
APB_RDATA	Output	32	Data read to the higher logic.
APB_ADDR	Input	32	Address driven by the higher logic.
APB_WDATA	Input	32	Data input by the higher logic.
APB_STRB	Input	4	Strobe signals driven by the higher logic.
APB_WRITE	Input	1	Write/Read signals from the higher logic
APB_READY	Output	1	Ready signal to the higher logic to complete transfer.

5.3.Decoder module

The decoder block is responsible for decoding the PSEL from the master and drive the selection inputs of the slaves, also receiving the ready and read data from the slave and choosing to pass the valid one to the master.

5.3.1.Parameters

Parameters	Accessibility	Default	Description
DATA_WIDTH	Global	32	Input Data Width, up to 32 bits, also defines the memory Word width.
SLAVE_NUM	Global	2	Number of slaves derived by the master.

5.3.2.Signals

Signals	Port	Width	Description
PSEL	Input	2	Selection signal from the master.
PREADY	Output	1	Valid Ready signal to the master.
PRDATA	Output	32	Valid data read to the slave.
PREADY0	Input	1	Ready signal from the first slave.
PREADY1	Input	1	Ready signal from the second slave.
PRDATA0	Input	32	Read data from the first slave.
PRDATA1	Input	32	Read data from the second slave.

5.4.APB RAM

The memory slave is implemented as a parameterized RAM block that can be accessed through the AMBA APB interface. It supports both read and write operations, with configurable address width and data width to allow flexibility in storage capacity. Each memory location is addressable via the APB address bus, enabling the master to store or retrieve data efficiently. This block acts as a general-purpose storage unit within the system, providing simple and reliable memory-mapped access for verification and testing of the APB protocol.

5.4.1.Parameters

Parameters	Accessibility	Default	Description
DATA_WIDTH	Global	32	Input Data Width, up to 32 bits, also defines the memory Word width.
RAM_DEPTH	Global	64	Memory Ram Word number.
DATA_BYTE_NUM	Local	4	Byte number in the DATA bus.
RAM_ADDR_WIDTH	Local	8	Ram Address bits.
BYTE_ENCODING_BITS	Local	2	Bits needed to encode the Bytes.
MAIN_ADDR_WIDTH	Global	32	The main address width driven by the master.

5.4.2.Signals

Signals	Port	Width	Description
PCLK	Input	1	System clock.
PRESET_n	Input	1	System reset.
PADDR	Input	32	Address driven from the master.
PSEL	Input	2	Selection signal to activate the slave.
PENABLE	Input	1	Enable signal.
PWRITE	Input	1	Write/Read signal , when HIGH indicates write.
PWDATA	Input	32	Data input to be written.
PSTRB	Input	4	Strobe signal to validate certain bytes of the data input.
PREADY	Output	1	Ready signal to complete transfer.
PRDATA	Output	32	Data read output.

5.5.APB Timer Slave

The timer slave is an APB-compliant peripheral that provides a simple countdown timing function controlled by the master. It contains registers for the load value, current counter value, control, and interrupt status/clear. On reset, the timer initializes its counter, and once enabled, it decrements on each clock cycle until reaching zero. At that point, it generates an interrupt request to the system and then stops, as the design supports only one-shot mode. The registers are accessed through the APB interface with byte-addressable addressing, allowing the master to configure the timer, observe its current value, and clear the interrupt in compliance with the AMBA APB protocol.

5.5.1.Parameters in the Timer Slave module

Parameters	Accessibility	Default	Description
DATA_WIDTH	Global	32	Input Data Width, up to 32 bits, also defines the memory Word width.
DATA_BYTE_NUM	Local	4	Byte number in each DATA bus.
TIMER_ADDR_WIDTH	Local		Timer Address bits.
BYTE_ENCODING_BITS	Local	2	Bits needed to encode the Bytes.
MAIN_ADDR_WIDTH	Global	32	The main address width driven by the master.
CNTRL	Local	0	Address state which enables the timer , external clock ,external enables and the interrupt signals.
VALUE	Local	1	Address state points at the Timer value.
RELOAD	Local	2	Address state points at Timer Reload value.
INT	Local	3	Address state points at Interrupt signal.

5.5.2.Signals

Signals	Port	Width	Description
PCLK	Input	1	System clock.
PRESET_n	Input	1	System reset.
PADDR	Input	32	Address driven from the master.
PSEL	Input	2	Selection signal to activate the slave.
PENABLE	Input	1	Enable signal.
PWRITE	Input	1	Write/Read signal , when HIGH indicates write.
PWDATA	Input	32	Data input to be written.
PSTRB	Input	4	Strobe signal to validate certain bytes of the data input.
PREADY	Output	1	Ready signal to complete transfer.
PRDATA	Output	32	Data read output.

5.6.Wrapped module

The wrapper module is the top-level block that integrates the APB master and its connected slaves. In addition to instantiating the master, timer, and memory, the wrapper also embeds the address decoding logic internally. This decoder generates one-hot select signals based on the master's address and ensures that only the targeted slave is activated during a transfer. By combining both the system integration and the decoding function inside a single block, the wrapper simplifies the overall design and manages all routing between the master and the slaves.

5.6.1.Parameters in the Wrapper module

Parameters	Accessibility	Default	Description
DATA_WIDTH	Global	32	Input Data Width, up to 32 bits, also defines the memory Word width.
RAM_DEPTH	Global	64	Memory Ram Word number.
SLAVE_NUM	Global	2	Number of slaves derived by the master.
MAIN_ADDR_WIDTH	Global	32	The main address width driven by the master.
DATA_BYTE_NUM	Local	4	Byte number in each DATA bus.
BYTE_ENCODING_BITS	Local	2	Bits needed to encode the Bytes.

5.6.2.Signals

Signals	Port	Width	Description
Transfer	Input	1	Transfer signal to start operation.
PCLK	Input	1	System clock.
PRESET_n	Input	1	System reset.
RDATA	Output	32	Data read.
ADDR	Input	32	Address.
WDATA	Input	32	Data input.
STRB	Input	4	Strobe signals.
WRITE	Input	1	Write/Read signals.
READY	Output	1	Ready signal.

5.7.Flow sequence

- The master receives a transfer request consisting of an address, a write/read command, and data. The address encodes both the peripheral selection and the register location within that peripheral.

  Notice: Only one peripheral is selected at a time as the master only sends one data bus to the slave and receives one bus in the read state.

- Once the master received the transfer signals it goes through the state diagram as explained before.
- In the **SETUP** state, the slave receives the selection, address and the read/write command, along with write data if applicable.
- In the **ACCESS** state, the slave is enabled. If the slave is ready, the transfer is completed on the next rising clock edge. If the slave is not ready, the transfer is extended until the slave asserts readiness, after which it is completed on the next rising clock edge.

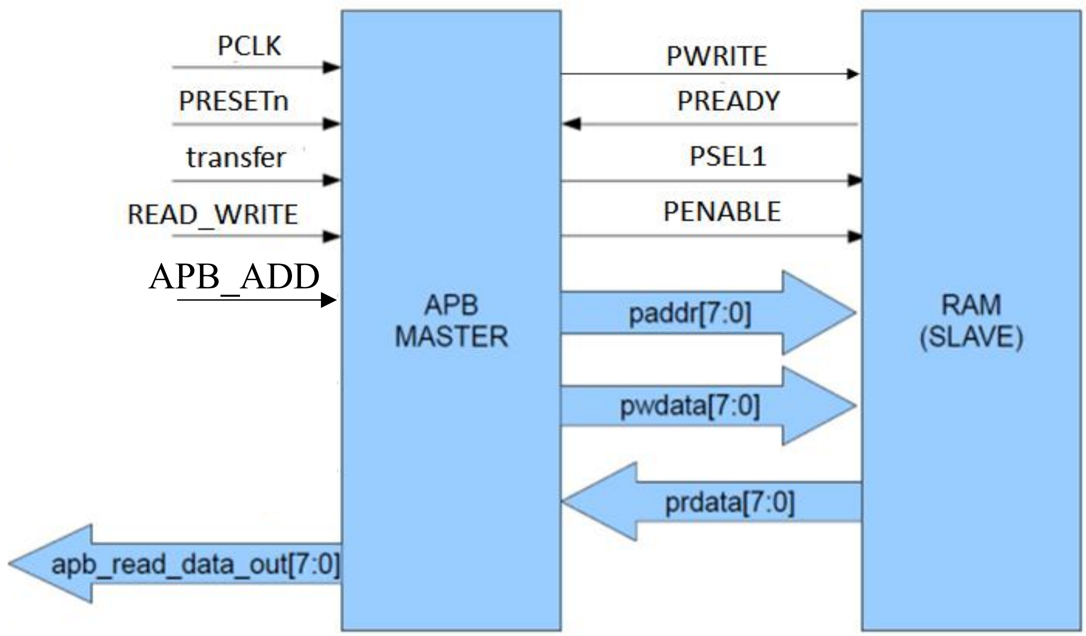


Figure 10 Block diagram for the master and the slave communication

5.8.Verilog Code

5.8.1.RTL Code for Master

```verilog
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//PREPARED BY : OMAR MOHAMED HUSSEIN MOSTAFA
//MODULE      : MASTER
///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
module APB_master#(parameter DATA_WIDTH=32,parameter SLAVE_NUM=2,parameter MAIN_ADDR_WIDTH=32)
        (transfer,PCLK,PRESET_n,PADDR,PSEL,PENABLE,PWRITE,PWDATA,PSTRB,PREADY,PRDATA,
                APB_RDATA,APB_ADDR,APB_WDATA,APB_STRB,APB_WRITE,APB_PREADY
        );

localparam DATA_BYTE_NUM=DATA_WIDTH/8 ;                 //Default=4 Bytes

//States Encoding
localparam IDLE  =2'b00;
localparam SETUP =2'b01;
localparam ACCESS=2'b10;
// Internal signals for the next and current states
reg [1:0]cs,ns;


//system inputs
input                           PCLK      ;             //system Clock
input                           PRESET_n  ;             //system Reset
//higher request to the master
input                           transfer  ;             //Transfer signal to instantiate the APB
input                           APB_WRITE ;             //Operation Write or Read
input      [DATA_BYTE_NUM  -1:0] APB_STRB  ;             //Strobe select from the higher logic
input      [DATA_WIDTH     -1:0] APB_WDATA ;             //Data from the higher logic
input      [MAIN_ADDR_WIDTH-1:0] APB_ADDR  ;             //encoming address encodes both the slave select and the byte addressing.
//output from the master to the higher logic
output     [DATA_WIDTH     -1:0] APB_RDATA ;             //output Read data from the master to the higher logic
output                          APB_PREADY;             //output Ready signal to the higher logic
//output from master to the slave
output reg [MAIN_ADDR_WIDTH-1:0] PADDR     ;             //Address sent to the slave in byte addressed style
output reg [DATA_WIDTH     -1:0] PWDATA    ;             //Data transfered to the slave
output reg [DATA_BYTE_NUM  -1:0] PSTRB     ;             //Strobe signal to validate certain DATA input bytes
output reg [SLAVE_NUM      -1:0] PSEL      ;             //Select signal to be decoded to each slave
output reg                      PENABLE   ;             //Enable signal to access the slaves
output reg                      PWRITE    ;             //Operation signal to whether read or write
//input for the master from the slave
input                           PREADY    ;             //Ready signal from the slave to the master to end transfer
input      [DATA_WIDTH     -1:0] PRDATA    ;             //Data read to be sent to the master

assign APB_PREADY=PREADY;                               //Ready signal to be sent to the higher logic
assign APB_RDATA =PRDATA;                               //Data read sent to the higher logic

//output  logic
always @(*) begin
    case(cs)
      IDLE: begin
        {PADDR,PWDATA,PSTRB,PSEL,PENABLE,PWRITE}='b0;
      end
      SETUP: begin
        //NOTE: THE APB ADDR COMMING FROM THE HIGHER LOGIC CONTAINS  BOTH ADDRESS TO BE  ACCESSED AND THE SELECTED SLAVE  //
        //APB_ADDR ENCODED AS FOLLOWS ------> APB_ADDR={PSELECTION[SLAVE_NUM-1:0],ADDRESS ACCESSED IN THE SLAVES REGISTER}//
        PADDR=APB_ADDR[MAIN_ADDR_WIDTH-1  :0        ];  //Register the 32 bit address
        PSEL =APB_ADDR[MAIN_ADDR_WIDTH-1 -:SLAVE_NUM];  //decode from the ADDR the PSEL based on the SLave num starting from the MSB

        PWDATA =APB_WDATA;                              //register the data to be written
        PENABLE=0;                                      //SETUP ----> PENABLE=0
        if(APB_WRITE) begin
          PSTRB =APB_STRB;                              //register the PSTROBE only in writing phase
        end
        else  begin
          PSTRB ='b0;                                   //STROBES must be zero while reading
        end
        PWRITE =APB_WRITE;                              //register the operation to be implemented
      end

```

```verilog
      ACCESS: begin
        PADDR=APB_ADDR[MAIN_ADDR_WIDTH-1  :0        ];  //Register the 32 bit address
        PSEL =APB_ADDR[MAIN_ADDR_WIDTH-1 -:SLAVE_NUM];  //decode from the ADDR the PSEL based on the SLave num starting from the MSB

        PWDATA =APB_WDATA;                              //register the data to be written
        if(APB_WRITE) begin
          PSTRB =APB_STRB;                              //register the PSTROBE only in writing phase
        end
        else  begin
          PSTRB ='b0;                                   //STROBES must be zero while reading
        end
        PWRITE =APB_WRITE;                              //register the operation to be implemented
        PENABLE=1;                                      //ACCESS -----> PENABLE is high
      end
      default:{PADDR,PWDATA,PSTRB,PSEL,PENABLE,PWRITE}='b0;
    endcase
end

//memory logic
always @(posedge PCLK) begin
    if(!PRESET_n) begin
        ns<=0;
    end
    else begin
      cs<=ns;
    end
end

//next state logic
always @(*) begin
    case(cs)
        IDLE: begin
            if(!transfer) begin
                ns=IDLE;
            end
            else begin
                ns=SETUP;
            end
        end
        SETUP: ns=ACCESS;
        ACCESS: begin
          if(!PREADY) begin
            ns=ACCESS;
          end
          else if (PREADY && transfer) begin            //Another transfer request
            ns=SETUP;
          end
          else if (PREADY && !transfer) begin           //No other transfers
            ns=IDLE;
          end
          else begin
            ns=IDLE;
          end
        end
        default : ns=IDLE;
    endcase
end
endmodule
```

5.8.2.RTL Code for Decoder

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//MODULE: DECODER
//NOTE  : THIS DECODER IS MADE TO DECODE TO ONLY TWO SLAVES , IF MORE , THE DECODER WILL  NEED MODIFICATIONS
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
module decoder #(parameter DATA_WIDTH=32,parameter SLAVE_NUM=2)(PSEL,PSEL0,PSEL1,PRDATA,PRDATA0,PRDATA1,PREADY0,PREADY1,PREADY);


//Master to DECODER
input      [SLAVE_NUM -1:0] PSEL   ;
//DECODER TO SLAVES
output reg                  PSEL0  ;
output reg                  PSEL1  ;
//SLAVES TO DECODER
input      [DATA_WIDTH-1:0] PRDATA0;
input      [DATA_WIDTH-1:0] PRDATA1;
input                       PREADY0;
input                       PREADY1;
//DECODER TO MASTER
output reg                  PREADY ;
output reg [DATA_WIDTH-1:0] PRDATA ;


always @(*) begin
    case (PSEL)
        2'h1 :  begin
            PSEL0=1;
            PSEL1=0;
            PRDATA=PRDATA0;
            PREADY=PREADY0;
        end
        2'h2 : begin
            PSEL0=0;
            PSEL1=1;
            PRDATA=PRDATA1;
            PREADY=PREADY1;
        end
        default: begin
            PSEL0=0;
            PSEL1=0;
            PRDATA=0;
            PREADY=0;
        end
    endcase
end
endmodule
```

5.8.3.RTL Code for the APB RAM memory slave

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//MODULE   : RAM MEMORY Slave
//SLAVE_NUM: ZERO
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

module MEM_slave#(parameter DATA_WIDTH=32 , parameter RAM_DEPTH=64,parameter MAIN_ADDR_WIDTH=32)(
  PCLK,PRESET_n,PADDR,PSEL,PENABLE,PWRITE,PWDATA,PSTRB,PREADY,PRDATA
  );

localparam DATA_BYTE_NUM      = DATA_WIDTH/8                       ;   //Default=4 Bytes
localparam BYTE_ENCODING_BITS = $clog2(DATA_BYTE_NUM)              ;   //Default=2 bits
localparam RAM_ADDR_WIDTH     = BYTE_ENCODING_BITS+$clog2(RAM_DEPTH);   //Default=2 bits + 6 bits = 8 bits

//INPUTS TO THE SLAVE
input [MAIN_ADDR_WIDTH-1:0] PADDR    ;               //32 bit Address coming from the master encoded in byte address style
input [DATA_WIDTH     -1:0] PWDATA   ;               //32 Data to be written in the MEMORY
input [DATA_BYTE_NUM  -1:0] PSTRB    ;               //Strobe signal to select the valid input bytes to be written
input                       PCLK     ;               //System Clock
input                       PRESET_n ;               //System Reset
input                       PSEL     ;               //Select signal to activate the Slave
input                       PENABLE  ;               //Enable signal to allow the Access of the Slave
input                       PWRITE   ;               //Write/Read signal to control the operation whether read or write
integer i;

//OUTPUTS FROM THE SLAVE
output reg [DATA_WIDTH-1:0] PRDATA;
output reg                  PREADY;

//======================Decoding the PADDR as it's Byte addressed not word addressed====================//
//========================PADDR[PSelection,word_addr,bytes location to be accessed]=======================//
wire [RAM_ADDR_WIDTH-BYTE_ENCODING_BITS-1:0] word_addr;             //Default [8-2-1:0] -> [5:0] -> 6  bits
assign word_addr= PADDR[RAM_ADDR_WIDTH -1:BYTE_ENCODING_BITS];      //Default [7    :2] ----------> 6  bits
reg  [DATA_WIDTH                      -1:0] temp_data;              //Default [31   :0] ----------> 32 bits
reg flag;                                                           //Flag asserted when Data is written

reg [DATA_WIDTH-1:0]MEM[RAM_DEPTH-1:0];                             //MEMORY RAM With 64 word each of 32 bits

always @(posedge PCLK) begin
    if(!PRESET_n) begin
      {PREADY,PRDATA,flag,temp_data} <= 'b0;
    end
    else begin
      PREADY<=0;
      if(PSEL) begin
        temp_data<=MEM[word_addr];
        if(PENABLE) begin
          if(PWRITE) begin
            PREADY<=0;
            if(!flag) begin
            for(i=0;i<=DATA_BYTE_NUM-1;i=i+1) begin
              if(PSTRB[i]) begin
               temp_data[i*8 +:8] <= PWDATA[i*8 +:8];
              end
            end
            PREADY<=1;
            flag<=1;
            end
            else begin
              MEM[word_addr]<=temp_data;
              flag<=0;
            end
          end
          else  begin
            PREADY<=1;
            PRDATA<=MEM[word_addr];
          end
        end
      end
    end
end
endmodule
```

5.8.4.RTL Code for the APB Timer slave

```verilog
/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//MODULE   : TIMER Slave
//SLAVE_NUM: ONE
/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

module Timer_slave#(parameter DATA_WIDTH=32,parameter MAIN_ADDR_WIDTH=32)(
        PCLK,PRESET_n,PADDR,PSEL,PENABLE,PWRITE,PWDATA,PSTRB,PREADY,PRDATA
        );

localparam DATA_BYTE_NUM      = DATA_WIDTH/8               ;       //Default=4 Bytes
localparam BYTE_ENCODING_BITS = $clog2(DATA_BYTE_NUM)      ;       //Default=2 bits
localparam TIMER_ADDR_WIDTH   = BYTE_ENCODING_BITS+$clog2(4);       //Default=2 bits + 2 bits = 4 bits

//INPUTS TO THE SLAVE
input [MAIN_ADDR_WIDTH-1:0]   PADDR     ;                          //32 bit Address coming from the master encoded in byte address style
input [DATA_WIDTH-1     :0]   PWDATA    ;                          //32 Data to be written
input [DATA_BYTE_NUM-1  :0]   PSTRB     ;                          //Strobe signal to select the valid input bytes to be written
input                         PCLK      ;                          //System Clock
input                         PRESET_n  ;                          //System Reset
input                         PSEL      ;                          //Select signal to activate the Slave
input                         PENABLE   ;                          //Enable signal to allow the Access of the Slave
input                         PWRITE    ;                          //Write/Read signal to control the operation whether read or write
integer i;

//OUTPUTS FROM THE SLAVE
output reg [DATA_WIDTH-1:0]   PRDATA    ;                          //32 bit output going to the selector
output reg                    PREADY    ;                          //Ready  signal going to the selector

//internal Address states of the timer
localparam CNTRL =4'h0;
localparam VALUE =4'h4;
localparam RELOAD=4'h8;
localparam INT   =4'hc;

//Timer signals
reg [DATA_WIDTH-1:0]          TIMER_VALUE ;                        //the timer countdown value
reg [DATA_WIDTH-1:0]          TIMER_RELOAD;                        //the timer reload values to start from
reg                           INT_EN      ;                        //the timer interrupt enable signal
reg                           EXT_CLK     ;                        //external timer used when EXT_CLK is activated
reg                           EXT_EN      ;                        //external enable used when EXT_EN is activated
reg                           TIMER_EN    ;                        //timer enable signal driven by the master
reg                           INT_STATUS  ;                        //indicates the timer state
//INT_STATUS--->indicates when reading  whether counting down or the timer has finished, while when writing it resets the state//

//Always block for the interferance with the master
always @(posedge PCLK) begin
    if(!PRESET_n) begin
      PREADY <=0  ;
      PRDATA <='b0;
      {INT_EN,EXT_CLK,EXT_EN,TIMER_EN}<='b0;
    end
    else  begin
        if(PSEL) begin
            PREADY<=1;
            if(PENABLE) begin
                PREADY<=0;
              if (PWRITE) begin
                case(PADDR[TIMER_ADDR_WIDTH-1:0])                  //only these bits we are looking for
                    CNTRL: begin
                        if(PSTRB[0]) begin
                            {INT_EN,EXT_CLK,EXT_EN,TIMER_EN}<=PWDATA[3:0];
                        end
                    end
                    VALUE: begin
                      for(i=0;i<=DATA_BYTE_NUM-1;i=i+1) begin
                          if(PSTRB[i]) begin
                           TIMER_VALUE[i*8 +:8] <= PWDATA[i*8 +:8];
                          end
                        end
                    end
```

5.8.5.RTL Code for the Wrapped module

```verilog
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//                                                       <-----------------------                                  //
//                                                       |         ----------> SLAVE0                              //
//               HIGHER_LOGIC <---------> MASTER <--------> DECODER -                                              //
//                                                       |         ----------> SLAVE1                              //
//                                                       <-----------------------                                  //
//                                                                                                                 //
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
module APB_Wrapper#(parameter DATA_WIDTH=32 , parameter RAM_DEPTH=64,parameter SLAVE_NUM=2,parameter MAIN_ADDR_WIDTH=32)
                        (PCLK,ADDR,WDATA,RDATA,PRESET_n,transfer,WRITE,READY,STRB);

localparam DATA_BYTE_NUM      = DATA_WIDTH/8;
localparam BYTE_ENCODING_BITS = $clog2(DATA_BYTE_NUM);       //Default=2 bits

//Inputs from the Higher Logic to MASTER
input                          PCLK    ;
input                          PRESET_n;
input                          transfer;
input                          WRITE   ;
input [DATA_WIDTH        -1:0] WDATA   ;
input [DATA_BYTE_NUM     -1:0] STRB    ;
input [MAIN_ADDR_WIDTH   -1:0] ADDR    ;

//Outputs to the Higher logic FROM MASTER
output[DATA_WIDTH        -1:0] RDATA   ;
output                         READY   ;

//MASTER WIRES WITH DECODER
wire  [SLAVE_NUM         -1:0] PSEL    ;
wire  [DATA_WIDTH        -1:0] PRDATA  ;
wire                           PREADY  ;

//SLAVE0 with the DECODER
wire                           PSEL0   ;
wire                           PREADY0 ;
wire  [DATA_WIDTH        -1:0] PRDATA0 ;

//SLAVE1 with the DECODER
wire                            PSEL1  ;
wire                           PREADY1 ;
wire  [DATA_WIDTH        -1:0] PRDATA1 ;

//MASTER with both slaves
wire  [MAIN_ADDR_WIDTH    -1:0] PADDR   ;
wire  [DATA_WIDTH         -1:0] PWDATA  ;
wire  [DATA_BYTE_NUM      -1:0] PSTRB   ;
wire                            PENABLE ;
wire                            PWRITE  ;

APB_master #(DATA_WIDTH,SLAVE_NUM,MAIN_ADDR_WIDTH)MASTER(
        .transfer(transfer)
        ,.PCLK(PCLK)
        ,.PRESET_n(PRESET_n)
        ,.PADDR(PADDR)
        ,.PSEL(PSEL)
        ,.PENABLE(PENABLE)
        ,.PWRITE(PWRITE)
        ,.PWDATA(PWDATA)
        ,.PSTRB(PSTRB)
        ,.PREADY(PREADY)
        ,.PRDATA(PRDATA)
        ,.APB_RDATA(RDATA)
        ,.APB_ADDR(ADDR)
        ,.APB_WDATA(WDATA)
        ,.APB_STRB(STRB)
        ,.APB_WRITE(WRITE)
        ,.APB_PREADY(READY));
```

```
MEM_slave  #(DATA_WIDTH,RAM_DEPTH,MAIN_ADDR_WIDTH) MEM_SLAVE (
          .PCLK(PCLK)
         ,.PRESET_n(PRESET_n)
         ,.PADDR(PADDR)
         ,.PSEL(PSEL0)
         ,.PENABLE(PENABLE)
         ,.PWRITE(PWRITE)
         ,.PWDATA(PWDATA)
         ,.PSTRB(PSTRB)
         ,.PREADY(PREADY0)
         ,.PRDATA(PRDATA0));
Timer_slave #(DATA_WIDTH,MAIN_ADDR_WIDTH) TIMER (
          .PCLK(PCLK)
         ,.PRESET_n(PRESET_n)
         ,.PADDR(PADDR)
         ,.PSEL(PSEL1)
         ,.PENABLE(PENABLE)
         ,.PWRITE(PWRITE)
         ,.PWDATA(PWDATA)
         ,.PSTRB(PSTRB)
         ,.PREADY(PREADY1)
         ,.PRDATA(PRDATA1));
decoder #(DATA_WIDTH,SLAVE_NUM) DECODER(
    .PSEL(PSEL),
    .PSEL0(PSEL0),
    .PSEL1(PSEL1),
    .PRDATA(PRDATA),
    .PRDATA0(PRDATA0),
    .PRDATA1(PRDATA1),
    .PREADY0(PREADY0),
    .PREADY1(PREADY1),
    .PREADY(PREADY)
);
endmodule

```

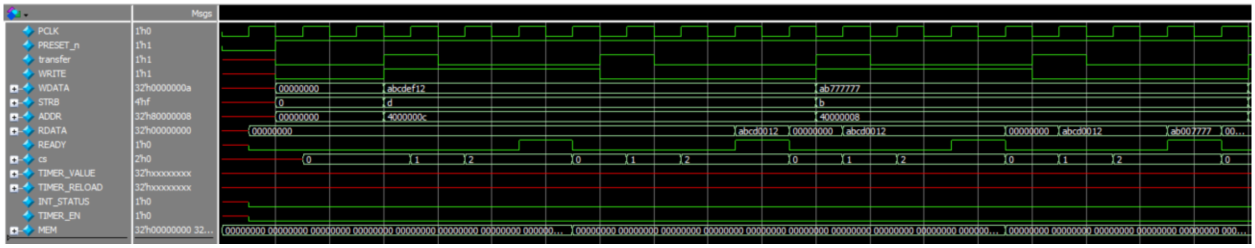
5.8.6.Testbench Code

```
module APB_tb();

reg  PCLK,PRESET_n,transfer,WRITE;
reg  [31:0] WDATA ;
reg  [3 :0] STRB  ;
reg  [31:0] ADDR  ;
wire [31:0] RDATA ;

APB_Wrapper #(32,64,2,32)DUT(PCLK,ADDR,WDATA,RDATA,PRESET_n,transfer,WRITE,READY,STRB);

initial begin
    PCLK=0;
    forever begin
        #10 PCLK=!PCLK;
    end
end

initial begin
    $readmemh("MEM.dat",DUT.MEM_SLAVE.MEM);
end

initial begin
//Reset assign
    PRESET_n=0;
    @(negedge PCLK);
    PRESET_n=1;
    {ADDR,WDATA,transfer,WRITE,STRB}='b0;
    repeat(2) @(negedge PCLK);
//Transfer and instantiating the MEM_slave
    transfer=1;
    ADDR=32'h4000000c;                              //select MEM , ADDRESS=c ---> word addr=3
    WRITE=1;                                        // Write operation
    WDATA=32'habcdef12;                             //Data to be written
    STRB=4'b1101;                                   //validate only this bytes
    repeat(1) @(negedge PCLK);                      //now we are at SETUP
    transfer=0;                                     //to make state resets after transfer
    repeat(2) @(negedge PCLK);                      //ACCESS then the transfer ends
    transfer=1;                                     //again another transfer
    WRITE=0;                                        //Read operation , same address before so we predict ab
cd0012 to be out
    repeat(1) @(negedge PCLK);                      //now we are setup
    transfer=0;                                     //to make state resets after transfer
    repeat(2) @(negedge PCLK);                      //ACCESS then the transfer ends
    transfer=1;                                     //again another transfer
    ADDR=32'h40000008;                              //select MEM , ADDRESS=8 ---> word addr=2
    WRITE=1;                                        //Write operation
    WDATA=32'hab777777;                             //write data
    STRB=4'b1011;                                   //validate only this bytes
    repeat(1) @(negedge PCLK);                      //now we are at SETUP
    transfer=0;                                     //to make state resets after transfer
    repeat(2) @(negedge PCLK);                      //ACCESS then the transfer ends
    transfer=1;                                     //transfer
    WRITE=0;                                        //Read operation
    repeat(1) @(negedge PCLK);                      //now setup
    transfer=0;                                     //to make state resets after transfer
    repeat(3) @(negedge PCLK);
    transfer=1;                                     //transfer
    ADDR=32'h80000008;                              //now the timer slave activated , address points about
reload value
    WDATA=32'h0000000a;                             //Reload value a to timer
    STRB=4'b1111;                                   //validate all the data in
    WRITE=1;                                        //write opration
    repeat(1) @(negedge PCLK);                      //setup state
    transfer=0;                                     //to make state resets after transfer
    repeat(2) @(negedge PCLK);                      //ACCESS then end transfer
    PRESET_n=0;                                     //Resets to reload the timer
    @(negedge PCLK);                                //wait pos edge clk
    PRESET_n=1;                                     //remove resets
    transfer=1;                                     //transfer
    ADDR=32'h80000000;                              //Timer slave activated ,address points to enable timer
    WDATA={28'b0,4'b1001};                          //write data to enables interrupt and and timer
```

```
    STRB=4'b1111;                               //validate data input
    WRITE=1;                                    //write operation
    repeat(1) @(negedge PCLK);                  //now setup state
    transfer=0;                                 //to make state resets after transfer
    repeat(2) @(negedge PCLK);                  //ACCESS then end transfer
    transfer=1;                                 //transfer
    ADDR=32'h80000004;                          //Timer slave activated ,address points to enable Timer value
    WDATA=32'h0000000a;                         //dumy bits
    STRB=4'b0000;                               //don't validate input
    WRITE=0;                                    //read the data
    repeat(3) @(negedge PCLK);                  //Setup access then end transfer
    transfer=1;                                 //transfer
    repeat(5) @(negedge PCLK);                  //read multiple times the timer value
    repeat(1) @(negedge PCLK);                  //Setup access then end transfer
    ADDR=32'h8000000c;                          //address points at interrupt state
    WDATA=32'h0000000a;                         //dumy bits
    STRB=4'b0000;                               //Don't validate inputs
    WRITE=0;                                    //Read data
    transfer=1;
    repeat(4) @(negedge PCLK);                  //access then end transfer
    transfer=1;
    // @(negedge PCLK);
    $stop;
end
endmodule
```

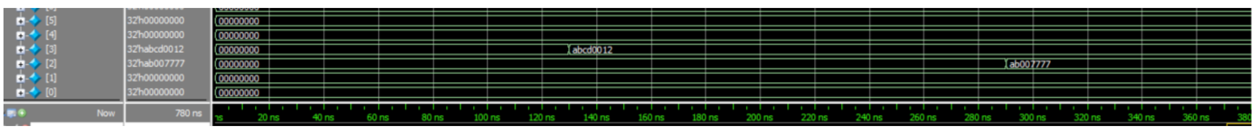
5.9.Questasim simulation

5.9.1.APB RAM Memory



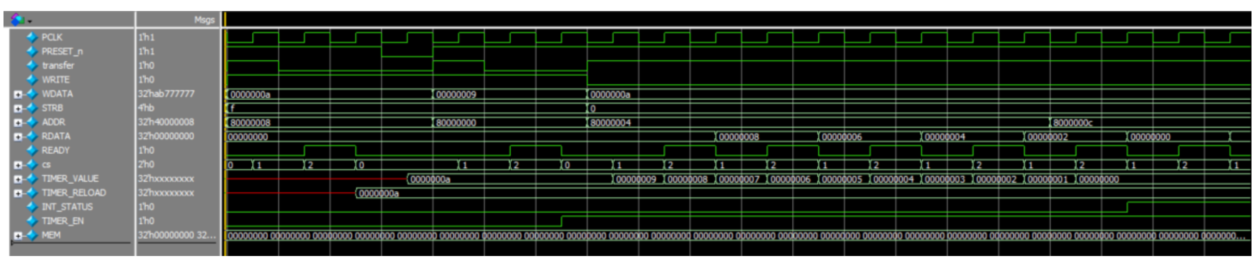
- ✓ **Reset is correct.**
- ✓ **Ready signal is correct.**

Here we should write $(\mathbf{abcd0012})_H$ in the MEM[3] and read it ,then write $(ab007777)_H$ in MEM[2].



- ✓ **Write and Read operation works well.**
- ✓ **The APB Ram works well.**

5.9.2.APB Timer

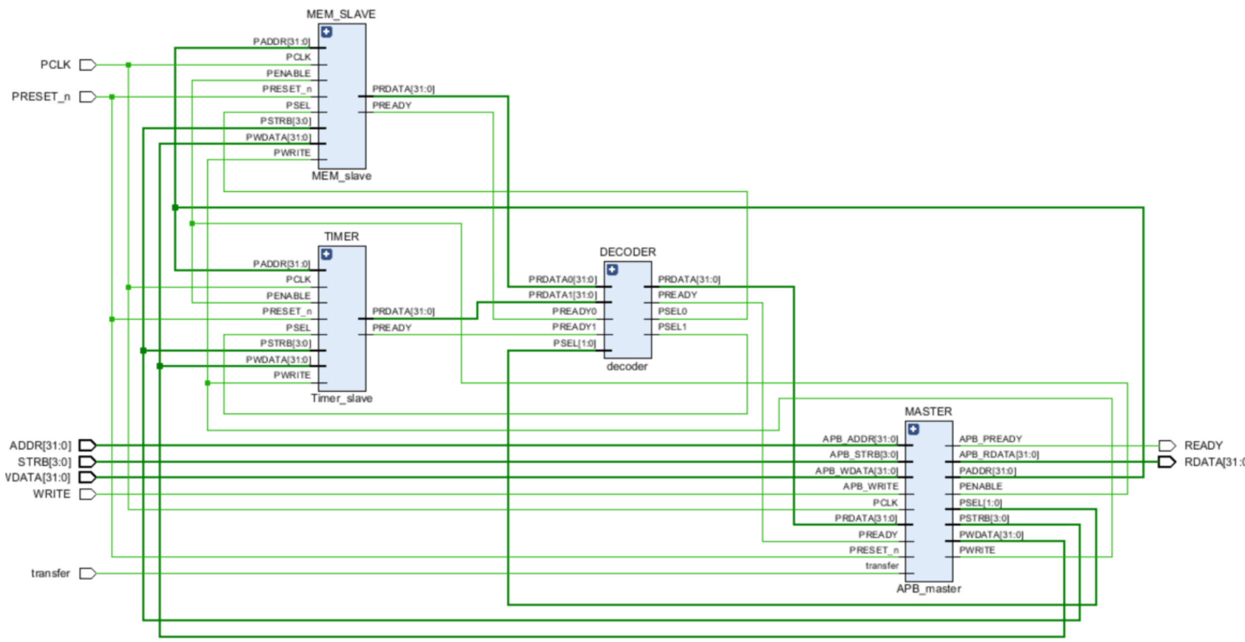


- At first, we reloaded the Timer Reload.
- Reset the timer to start at the Reload value.
- Enables the timer.
- Read the timer value multiple times.
- Read the interrupt signal at the end.

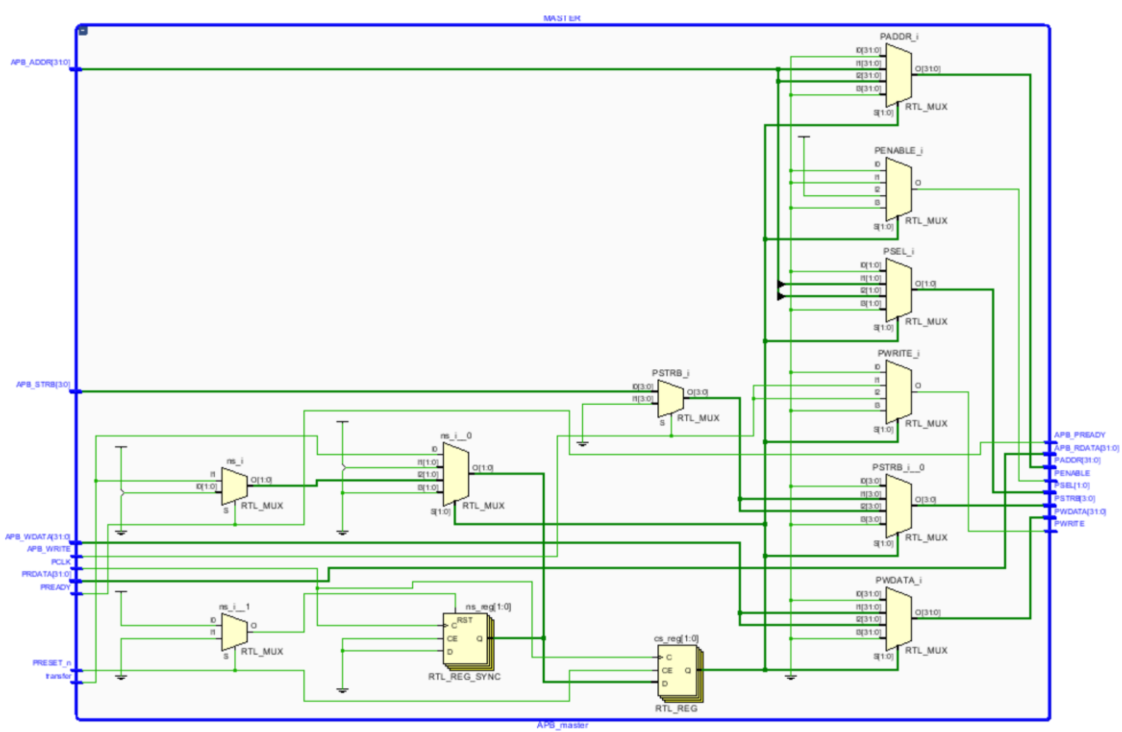
5.10.Synthesis tool

5.10.1.Elaboration

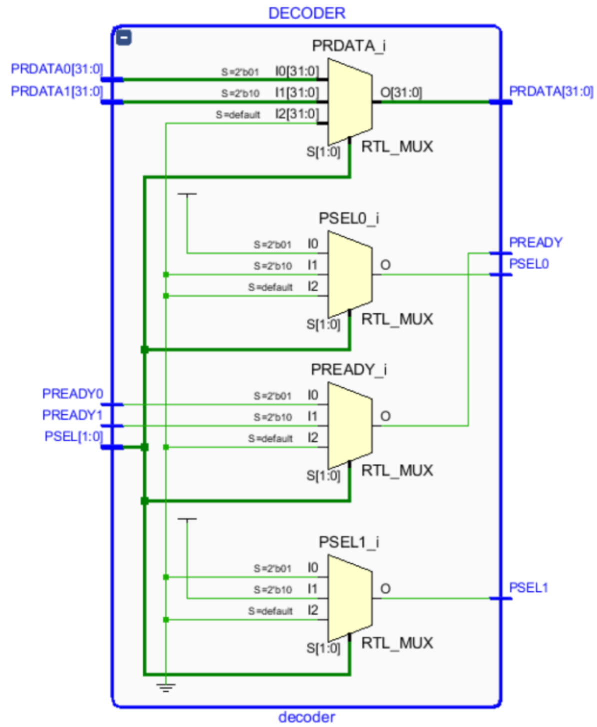
- Wrapped module Schematic



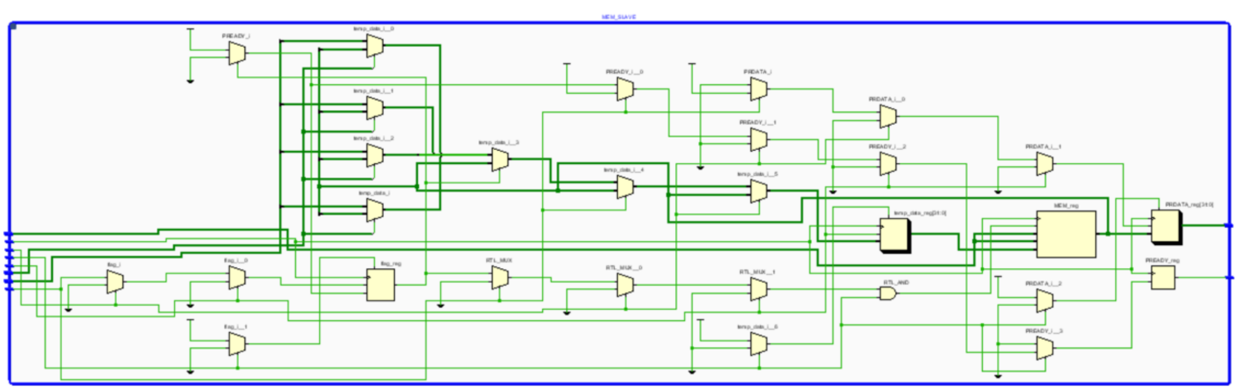
- Master module schematic



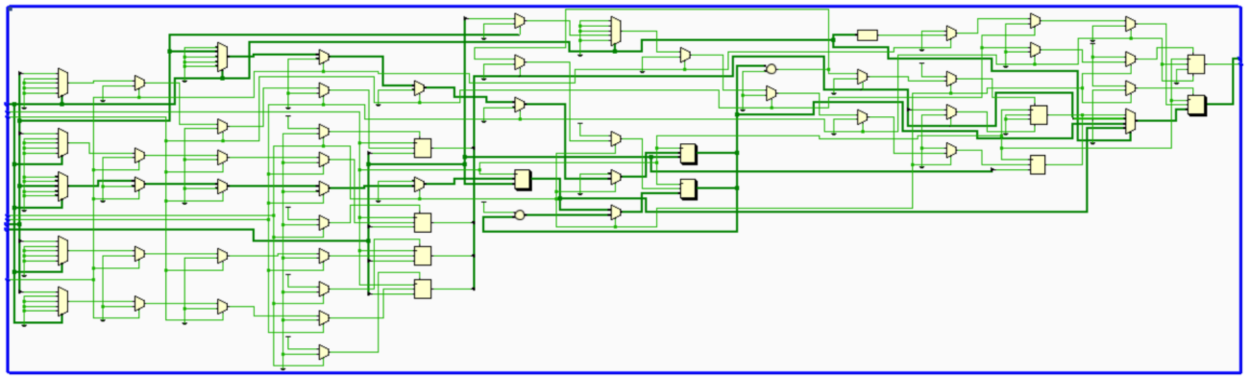
- Decoder schematic



- APB RAM memory schematic

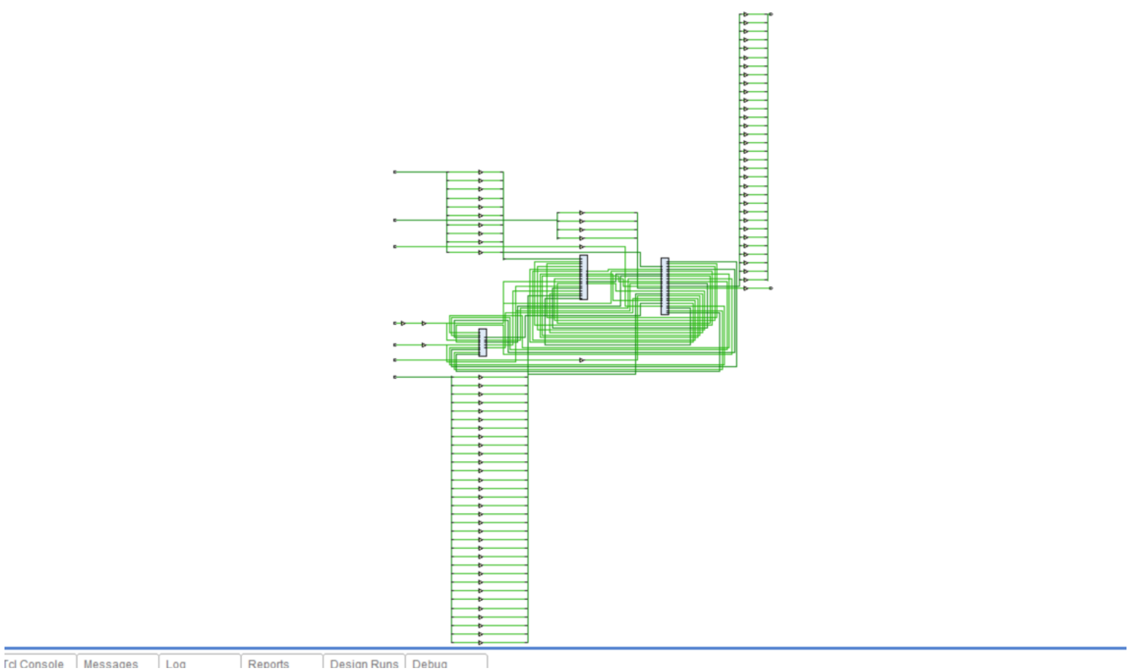


- APB Timer schematic

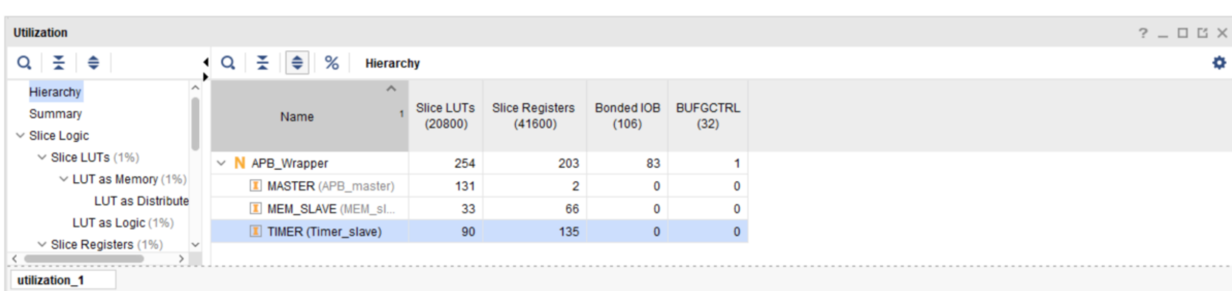


5.10.2.Synthesis

- Schematic



- Utilities Report



Name	Slice LUTs (20800)	Slice Registers (41600)	Bonded IOB (106)	BUFGCTRL (32)
APB_Wrapper	254	203	83	1
MASTER (APB_master)	131	2	0	0
MEM_SLAVE (MEM_sl...	33	66	0	0
TIMER (Timer_slave)	90	135	0	0