**BÁO CÁO ĐỒ ÁN 1**

**APB AMBA 2.0 PROTOCOL**

1. **APB PROTOCOL là gì?**

Giao tiếp APB (Advanced Peripheral Bus) là một trong bộ giao tiếp AMBA (Advanced Microcontroller Bus Architecture). AMBA là một cấu trúc bus hệ thống sử dụng trong các vi điều khiển được phát triển bởi ARM. AMBA có nhiều cấu trúc bus khác nhau như AXI, AHB, ASB, APB. Trong đó, APB là bus dùng để giao tiếp với các ngoại vi không đòi hỏi tốc độ xử lý nhanh như UART, SPI, I2C, ...

1. **Các tín hiệu của APB PROTOCOL**

APB có giao tiếp đơn giản gồm các tín hiệu như sau:

* PCLK: Tín hiệu clock đồng bộ hoạt động theo cạnh lên
* PADDR: địa chỉ thanh ghi được truy xuất
* PWRITE: tín hiệu cho phép ghi
* PSEL: tín hiệu cho phép
* PENABLE: tín hiệu báo pha ENABLE của một truy xuất đọc hoặc ghi
* PWDATA: dữ liệu ghi
* PRDATA: dữ liệu đọc

A diagram of a diagram

Description automatically generated with medium confidence

Hình 1: Giao tiếp APB – Ghi (trái) và Đọc (phải)

1. **Ví dụ minh hoạ**

Master APB: thành phần lái các tín hiệu điều khiển của bus APB. Master APB thường là một bus khác có tốc độ cao hơn trong hệ thống bus, ví dụ như AXI, AHB, ... tuy nhiên nó cũng có thể là một master đầu cuối có giao tiếp trực tiếp đến bus APB.

Slave APB: thành phần nhận các tín hiệu điều khiển của bus APB. Bộ điều khiển EEPROM trong bài này là một slave APB

A diagram of a program

Description automatically generated

Hình 2: Minh hoạ về kết nối của APB bus

Hoạt động của giao tiếp APB đơn giản gồm 3 pha (trạng thái):

* IDLE: khi PSEL = 0. Lúc này, PENABLE cũng phải bằng "0" và slave APB không quan tâm đến giá trị các tín hiệu điều khiển còn lại trên bus như PWRITE, PWDATA hay PADDR
* SETUP: khi PSEL = 1 và PENABLE = 0. Lúc này, các tín hiệu điều khiển trên bus phải có giá trị hợp lệ. Các tín hiệu điều khiển bao gồm PADDR, PWRITE và PWDATA nếu là một truy cập "ghi".
* ENABLE: khi PSEL = 1 và PENABLE = 1. Lúc này, slave APB phải nhận xong các giá điều khiển và phải lái PRDATA đến giá trị hợp lệ nếu đó là một truy cập "đọc"
* Chú ý, mỗi truy cập "đọc" hoặc "ghi" từ master APB đến slave APB phải qua 2 bước là SETUP và ENABLE. Sau bước ENABLE nếu không có truy cập nào khác thì trạng thái truy cập sẽ về IDLE nhưng nếu master APB tiếp tục có truy cập tiếp theo thì có thể lái trực tiếp qua trạng thái SETUP.

A diagram of a system

Description automatically generated

Hình 3: Các trạng thái hoạt động của bus APB

**AMBA 3.0 APB**

1. **Những thay đổi so với 2.0**

A diagram of a data processing process

Description automatically generated

Hình 1: AMBA 3.0 APB

* Tín hiệu sẵn sàng PREADY, để mở rộng chuyển đổi APB
* Tín hiệu lỗi PSLVERR, để chỉ ra sự thất bại khi chuyển đổi

A close-up of a diagram

Description automatically generated

Hình 2: Giao tiếp APB – Ghi không đợi (trái) và ghi có đợi (phải)

A diagram of a power supply

Description automatically generated with medium confidence

Hình 3: Giao tiếp APB – Đọc không đợi (trái) và đọc có đợi (phải)

A diagram of a computer program

Description automatically generated with medium confidence

Hình 4: Giao tiếp APB – Ghi có lỗi (trái) và Đọc có lỗi (phải)

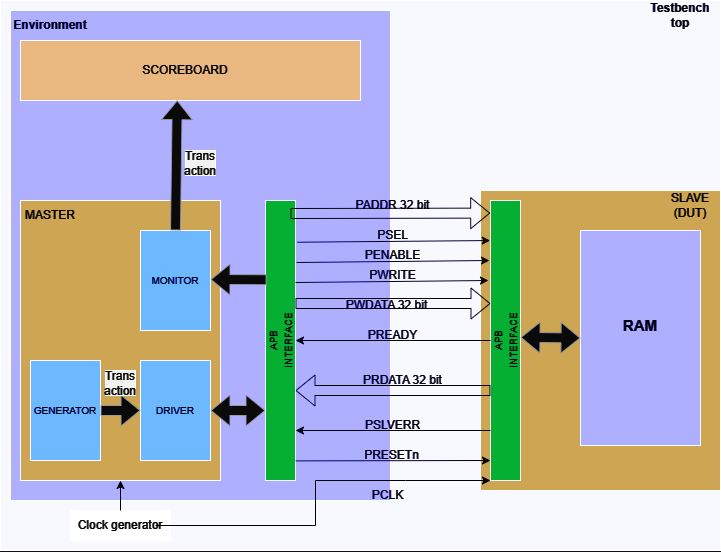
A diagram of a process

Description automatically generated

Hình 5: Các trạng thái hoạt động của AMBA 3.0 bus APB

**KIỂM TRA THIẾT KẾ AMBA 3.0 APB với THIẾT KẾ RAM**

1. **KIẾN TRÚC TESTBENCH**



Hình 1: Kiến trúc của testbench

1. **CÁC THÀNH PHẦN**
   1. Transaction

|  |
| --- |
| CODE |
| class transaction;  typedef enum int {  write = 0,  read = 1,  random = 2,  error = 3  } op\_type;  randc op\_type oper;  rand bit [31:0] paddr;  rand bit [31:0] pwdata;  rand bit psel, penable, pwrite;  bit pready, pslverr;  constraint addr\_c { //2 3 4  paddr > 1;  paddr < 5;  }  constraint data\_C {  pwdata > 1;  pwdata < 10;  }  function void display(input string tag);  $display(  "[%0s]OP:%s paddr:%0d pwdata:%0d psel:%0b penable:%0b pwrite:%0b prdata:%0d pready:%0b pslverr:%0b",  tag, oper.name(), paddr, pwdata, psel, penable, pwrite, prdata, pready, pslverr);  endfunction  function transaction copy();  copy = new();  copy.oper = this.oper;  copy.paddr = this.paddr;  copy.pwdata = this.pwdata;  copy.psel = this.psel;  copy.penable = this.penable;  copy.pwrite = this.pwrite;  copy.prdata = this.prdata;  copy.pready = this.pready;  copy.pslverr = this.pslverr;  endfunction  endclass |

* 1. Generator

|  |
| --- |
| CODE |
| class generator;  transaction tr;  mailbox #(transaction) mbx;  int count=0;  event nextdrv;//driver completed task of triggering interface  event nextsco;//scoreboard completed its objective  event done;  function new(mailbox #(transaction)mbx);  this.mbx=mbx;  tr=new();  endfunction  task run();  repeat(count)  begin  random: assert (tr.randomize())  else $error("Randomization failed!");  mbx.put(tr.copy);  tr.display("GEN");  @(nextdrv);  @(nextsco);  end  ->done;  endtask  endclass |

* 1. Driver

|  |
| --- |
| CODE |
| class driver;  virtual apb\_if vif;  mailbox #(transaction) mbx;  transaction datac;  event nextdrv;  function new(mailbox #(transaction)mbx);  this.mbx=mbx;  endfunction  task reset;  vif.presetn<=1'b0;  vif.psel<=1'b0;  vif.penable<=1'b0;  vif.pwdata<=0;  vif.paddr<=0;  vif.pwrite<=1'b0;  repeat(5)@(posedge vif.pclk);  vif.presetn<=1'b1;  repeat(5)@(posedge vif.pclk);  $display("[DRV]:RESET DONE");  endtask  task run;  forever begin  mbx.get(datac);  if(datac.oper==0)//write  begin  @(posedge vif.pclk);  vif.psel<=1'b1;  vif.penable<=1'b0;  vif.pwdata<=datac.pwdata;  vif.paddr<=datac.paddr;  vif.pwrite<=1'b1;  @(posedge vif.pclk);  vif.penable<=1'b1;  repeat(2) @(posedge vif.pclk);  vif.psel<=1'b0;  vif.penable<=1'b0;  vif.pwrite<=1'b0;  $display("[DRV]: DATA WRITE OP data: %0d and addr: %0d",datac.pwdata,datac.paddr);  end  else if (datac.oper==1)//read  begin  @(posedge vif.pclk);  vif.psel<=1'b1;  vif.penable<=1'b0;  vif.pwdata<=datac.pwdata;  vif.paddr<=datac.paddr;  vif.pwrite<=1'b0;  @(posedge vif.pclk);  vif.penable<=1'b1;  repeat(2) @(posedge vif.pclk);  vif.psel<=1'b0;  vif.penable<=1'b0;  vif.pwrite<=1'b0;  $display("[DRV]: DATA READ OP addr: %0d",datac.paddr);  end  else if (datac.oper==2) //random  begin  @(posedge vif.pclk);  vif.psel<=1'b1;  vif.penable<=1'b0;  vif.pwdata<=datac.pwdata;  vif.paddr<=datac.paddr;  vif.pwrite<=datac.pwrite;  @(posedge vif.pclk);  vif.penable<=1'b1;  repeat(2) @(posedge vif.pclk);  vif.psel<=1'b0;  vif.penable<=1'b0;  vif.pwrite<=1'b0;  $display("[DRV]: RANDOM OPERATION");  end  else if (datac.oper==3) //slave errror  begin  @(posedge vif.pclk);  vif.psel<=1'b1;  vif.penable<=1'b0;  vif.pwdata<=datac.pwdata;  vif.paddr<=$urandom\_range(32,100);  vif.pwrite<=datac.pwrite;  @(posedge vif.pclk);  vif.penable<=1'b1;  repeat(2) @(posedge vif.pclk);  vif.psel<=1'b0;  vif.penable<=1'b0;  vif.pwrite<=1'b0;  $display("[DRV]: SLV ERROR");  end  ->nextdrv;  end  endtask;  endclass |

* 1. Monitor

|  |
| --- |
| CODE |
| class monitor;  virtual apb\_if vif;  mailbox #(transaction) mbx;  transaction tr;  function new(mailbox #(transaction)mbx);  this.mbx=mbx;  endfunction  task run;  tr=new();  forever  begin  @(posedge vif.pclk);  if((vif.psel)&&(!vif.penable))  begin  @(posedge vif.pclk);  if(vif.psel && vif.penable && vif.pwrite)//write access  begin  @(posedge vif.pclk);  tr.pwdata=vif.pwdata;  tr.paddr=vif.paddr;  tr.pwrite=vif.pwrite;  tr.pslverr=vif.pslverr;  $display("[MON]: DATA WRITE data: %0d and addr: %0d write: %0b",vif.pwdata,vif.paddr,vif.pwrite);  @(posedge vif.pclk);  end  else if (vif.psel && vif.penable && !vif.pwrite)  begin  @(posedge vif.pclk);  tr.prdata=vif.prdata;  tr.paddr=vif.paddr;  tr.pwrite=vif.pwrite;  tr.pslverr=vif.pslverr;  @(posedge vif.pclk);  $display("[MON]: DATA READ data: %0d and addr: %0d write: %0b",vif.prdata,vif.paddr,vif.pwrite);  end  mbx.put(tr);  end  end  endtask  endclass |

* 1. Scoreboard

|  |
| --- |
| CODE |
| class scoreboard;  mailbox #(transaction) mbx;  transaction tr;  event nextsco;  bit [31:0] pwdata [12]='{default:0};  bit [31:0] rdata;  int index;  function new(mailbox #(transaction)mbx);  this.mbx=mbx;  endfunction  task run();  forever  begin  mbx.get(tr);  $display("[SCO]: DATA RECIVED wdata:%0d rdata:%0d addr:%0d write:%0b",tr.wdata,tr.prdata,tr.paddr,tr.pwrite);  if((tr.pwrite==1'b1)&&(tr.pslverr==1'b0))//write access  begin  pwdata[tr.paddr]=tr.pwdata;  $display("[SCO]: DATA STORED DATA: %0d ADDR: %0d",tr.pwdata,tr.paddr);  end  else if((tr.pwrite==1'b0)&&(tr.pslverr==1'b0))  begin  rdata=pwdata[tr.paddr];  if(tr.prdata==rdata);  $display("[SCO]: DATA MATCHED");  else  $display("[SCO]: DATA MISMATCHED");  end  else if(tr.pslverr==1'b1)  begin  $display("[SCO] : SLV ERROR DETECTED");  end  ->nextsco;  end  endtask  endclass |