A Report on MAVEN SILICON Internship

AHB-APB BRIDGE DESIGN PROJECT

Submitted By:

DEVARAKONDA SURESH JAHNAVI
MS/23-24/0368, DI33
Electronics and Communication Engineering
National Institute of Technology, Andhra Pradesh

Verilog Codes:

AHB Master:

```
module ahb_master(input hclk,hresetn,hr_readyout, input[31:0]hrdata,
input[1:0]hresp, output reg hwrite,hreadyin, output reg [31:0]haddr,hwdata,
output reg[1:0]htrans);
reg [2:0]hburst, hsize;
integer i;
task single_write();
begin
@(posedge hclk)
#1;
begin
hwrite=1;
htrans=2'd2;
hsize=0;
hburst=0;
hreadyin=1;
haddr=32'h8400_0000;
end
@(posedge hclk)
#1;
begin
hwdata=32'h29;
htrans=2'd0;
end
end
endtask
task single_read();
begin
@(posedge hclk)
#1;
begin
hwrite=0;
htrans=2'd2;
hsize=0;
```

```
hburst=0;
hreadyin=1;
haddr=32'h8400_0000;
end
@(posedge hclk)
#1;
begin
htrans=2'd0;
end
end
endtask
task burst_incr4_write();
begin
@(posedge hclk)
#1;
begin
hwrite=1;
htrans=2'd2;
hsize=0;
hburst=0;
hreadyin=1;
haddr=32'h8400_0000;
end
@(posedge hclk)
#1;
begin
haddr=haddr+1;
hwdata={$random}%256;
htrans=2'd3;
end
for(i=0;i<2;i=i+1)
begin
@(posedge hclk);
#1;
begin
haddr=haddr+1;
hwdata={$random}%256;
```

```
htrans=2'd3;
end
end
@(posedge hclk)
#1;
begin
hwdata={$random}%256;
htrans=2'd0;
end
end
endtask
task burst_incr4_read();
begin
@(posedge hclk)
#1;
begin
hwrite=0;
htrans=2'd2;
hsize=0;
hburst=0;
hreadyin=1;
haddr=32'h8400_0000;
end
for(i=0;i<3;i=i+1)
begin
@(posedge hclk);
#1;
begin
haddr=haddr+1;
htrans=2'd3;
end
@(posedge hclk);
end
end
endtask
endmodule
```

AHB Slave Interface:

```
module ahb_slave_interface(input hclk,hresetn,hwrite,hreadyin,
input [31:0] hwdata,haddr,prdata,
input [1:0]htrans,
output wire [31:0] hrdata,
output reg [31:0] haddr1,haddr2,hwdata1,hwdata2,
output reg hwrite_reg,hwrite_reg1, valid,
output reg [2:0] temp_selx);
//piplining haddr,hwdata and write signal
always@(posedge hclk)
begin
if(!hresetn)
begin
haddr1<=0;
haddr2<=0;
end
else
begin
haddr1  <=haddr;
haddr2<=haddr1;
end
end
always@(posedge hclk)
begin
if(!hresetn)
begin
hwdata1 <=0;
hwdata2<=0;
end
else
begin
hwdata1  <=hwdata;
hwdata2<=hwdata1;
end
end
always@(posedge hclk)
```

```
begin
if(!hresetn)
begin
hwrite_reg<=0;
hwrite_reg1 <=0;
end
else
begin
hwrite_reg<=hwrite;
hwrite_reg1 <=hwrite_reg;
end
end
//generating valid signal
always@(*)
begin
valid=0;
if(hreadyin==1 && haddr>=32'h80000000 && haddr<32'h8c000000 &&
(htrans==2'b10||htrans==2'b11))
valid=1;
else
valid=0;
end
//generating temp_selx signal
always@(*)
begin
temp_selx=0;
if(haddr>=32'h80000000 && haddr<32'h84000000)
temp_selx=3'b001;
else if(haddr>=32'h84000000 && haddr<32'h88000000)
temp_selx=3'b010;
else if(haddr>=32'h88000000 && haddr<32'h8c000000)
temp_selx=3'b100;
else
temp_selx=0;
end
assign hrdata=prdata;
endmodule
```

APB Controller:

```
module apb_controller(input hclk,hresetn,hwrite,hwrite_reg,valid,
input [31:0]haddr,haddr1,haddr2,hwdata,hwdata1,hwdata2,prdata,
input [2:0]tempselx,
output reg pwrite,penable,hr_readyout,
output reg [2:0]psel,
output reg [31:0]paddr,pwdata);

//temporary o/p variables
reg penable_temp,pwrite_temp,hr_readyout_temp;
reg [2:0]psel_temp;
reg [31:0]paddr_temp,pwdata_temp;

//present state sequential logic
parameter
idle=3'b000,read=3'b001,renable=3'b010,wwait=3'b011,write=3'b100,wenable=3'b
101,writep= 3'b110,wenablep=3'b111;
reg [2:0]present,next;

always@(posedge hclk)
begin
if(!hresetn)
present<=idle;
else
present<=next;
end

//next state combinational logic
always@(*)
begin
next=idle;
case(present)
```

```
idle:begin
if(valid==1&&hwrite==1)
next=wwait;
else if(valid==1&&hwrite==0)
next=read;
else
next=idle;
end
read:next=renable;
renable:begin
if(valid==1&&hwrite==1)
next=wwait;
else if(valid==1&&hwrite==0)
next=read;
else
next=idle;
end
wwait:begin
if(valid==0)
next=write;
else
next=writep;
end
write:begin
if(valid==0)
next=wenable;
else
next=wenablep;
end
wenable:begin
if(valid==1&&hwrite==1)
next=wwait;
else if(valid==1&&hwrite==0)
next=read;
else
next=idle;
end
```

```
writep:next=wenablep;
wenablep:begin
if(valid==1&&hwrite_reg==1)
next=writep;
else if(valid==0&&hwrite_reg==1)
next=write;
else
next=read;
end
endcase
end

//temporaray output logic(comb)
always@(*)
begin
case(present)
idle:begin
if(valid==1&&hwrite==0)
begin
paddr_temp=haddr;
pwrite_temp=hwrite;
psel_temp=tempselx;
penable_temp=0;
hr_readyout_temp=0;
end
else if(valid==1&&hwrite==1)
begin
psel_temp=0;
penable_temp=0;
hr_readyout_temp=1;
end
else
begin
psel_temp=0;
penable_temp=0;
hr_readyout_temp=1;
```

```
end
end

read:begin
penable_temp=1;
hr_readyout_temp=1;
end

renable:begin
if(valid==1&&hwrite==0)
begin
paddr_temp=haddr;
pwrite_temp=hwrite;
psel_temp=tempselx;
penable_temp=0;
hr_readyout_temp=0;
end
else if(valid==1&&hwrite==1)
begin
psel_temp=0;
penable_temp=0;
hr_readyout_temp=1;
end
else
begin
psel_temp=0;
penable_temp=0;
hr_readyout_temp=1;
end
end

wwait:begin
paddr_temp=haddr1;
pwdata_temp=hwdata;
pwrite_temp=hwrite;
psel_temp=tempselx;
penable_temp=0;
```

```
hr_readyout_temp=0;
end

write:begin
penable_temp=1;
hr_readyout_temp=1;
end

wenable:begin
if(valid==1&&hwrite==0)
begin
hr_readyout_temp=1;
psel_temp=0;
penable_temp=0;
end
else if(valid==1&&hwrite==0)
begin
paddr_temp=haddr1;
pwrite_temp=hwrite_reg;
psel_temp=tempselx;
penable_temp=0;
hr_readyout_temp=0;
end
else
begin
hr_readyout_temp=1;
psel_temp=0;
penable_temp=0;
end
end

writep:begin
penable_temp=1;
hr_readyout_temp=1;
end
```

```
wenablep:begin
paddr_temp=haddr1;
pwdata_temp=hwdata;
pwrite_temp=hwrite;
psel_temp=tempselx;
penable_temp=0;
hr_readyout_temp=0;
end
endcase
end


//output logic(seq)
always@(posedge hclk)
begin
if(!hresetn)
begin
paddr<=0;
pwdata<=0;
pwrite<=0;
psel<=0;
penable<=0;
hr_readyout<=1;
end
else
begin
paddr<=paddr_temp;
pwdata<=pwdata_temp;
pwrite<=pwrite_temp;
psel<=psel_temp;
penable<=penable_temp;
hr_readyout<=hr_readyout_temp;
end
end
endmodule
```

APB Interface:

```
module apb_interface(input pwrite,penable, input [2:0]psel,
input [31:0]paddr,pwdata, output pwrite_out,penable_out,
output [2:0]psel_out,output [31:0]paddr_out,pwdata_out, output reg [31:0]prdata);
assign pwrite_out=pwrite;
assign penable_out=penable;
assign psel_out=psel;
assign paddr_out=paddr;
assign pwdata_out=pwdata;
always@(*)
begin
if(!pwrite&&penable)
prdata=($random)%256;
else
prdata=32'h0;
end
endmodule
```

Bridge Top:

```
module bridge_top(input hclk,hresetn,hwrite,hreadyin,
input [1:0]htrans,input [31:0]hwdata,haddr,prdata,
output penable,pwrite,hr_readyout,output [2:0]psel,
output [1:0]hresp,output [31:0]paddr,pwdata,hrdata);
wire [31:0]hwdata1,hwdata2,haddr1,hadddr2;
wire [2:0]temp_sel;
wire hwrite_reg,hwrite_reg1;
wire valid;
ahb_slave_interface A1(hclk,hresetn,hwrite,hreadyin,hwdata,haddr,prdata,
htrans,hrdata,haddr1,haddr2,hwdata1,hwdata2,
hwrite_reg,hwrite_reg1,valid,temp_selx);
apb_controller A2(hclk,hresetn,hwrite,hwrite_reg,valid,
haddr,haddr1,haddr2,hwdata,hwdata1,hwdata2,prdata,
tempselx,pwrite,penable,hr_readyout,psel,
paddr,pwdata);
endmodule
```

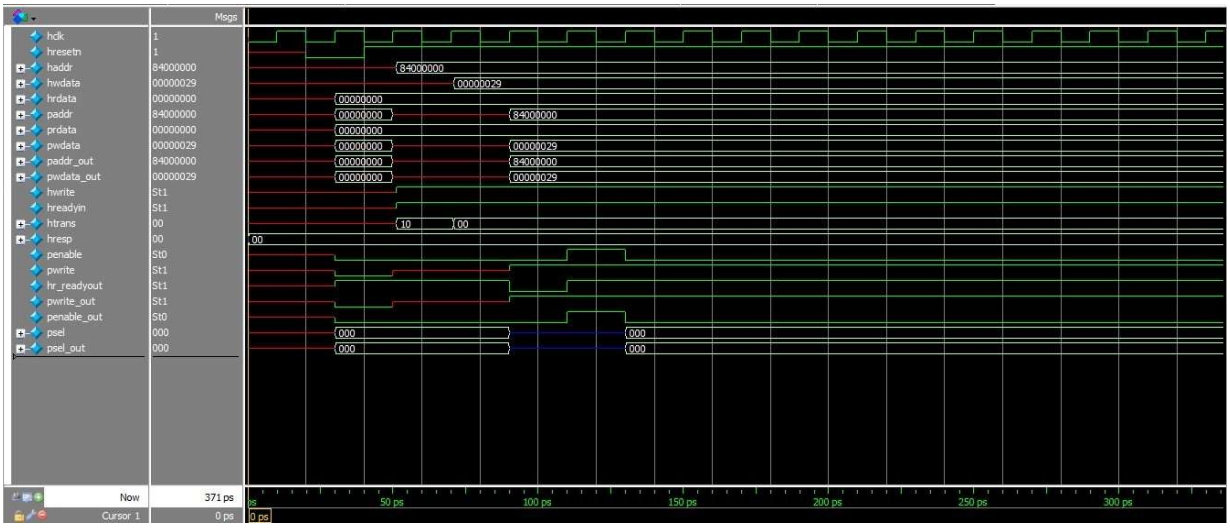
Top tb:

```
module top_tb();
reg hclk;
reg hresetn;
wire  [31:0]haddr,hwdata,hrdata,paddr,prdata,pwdata,paddr_out,pwdata_out;
wire hwrite,hreadyin;
wire [1:0]htrans;
wire [1:0]hresp;
wire penable,pwrite,hr_readyout,pwrite_out,penable_out;
wire [2:0]psel,psel_out;
ahb_master ahb(hclk,hresetn,hr_readyout,hrdata,hresp,
hwrite,hreadyin,haddr,hwdata,htrans);
bridge_top BRIDGE(hclk,hresetn,hwrite,hreadyin,htrans,
hwdata,haddr,prdata,penable,pwrite,hr_readyout,psel,
hresp,paddr,pwdata,hrdata);
apb_interface APB(pwrite,penable,psel,paddr,pwdata,
pwrite_out,penable_out,psel_out,paddr_out,pwdata_out,
prdata);
initial
begin
hclk=1'b0;
forever #10 hclk=~hclk;
end
task reset;
begin
@(negedge hclk)
hresetn=1'b0;
@(negedge hclk)
hresetn=1'b1;
end
endtask
initial
begin
reset;
//ahb.single_write(); #300;
//ahb.single_read(); #300;
```

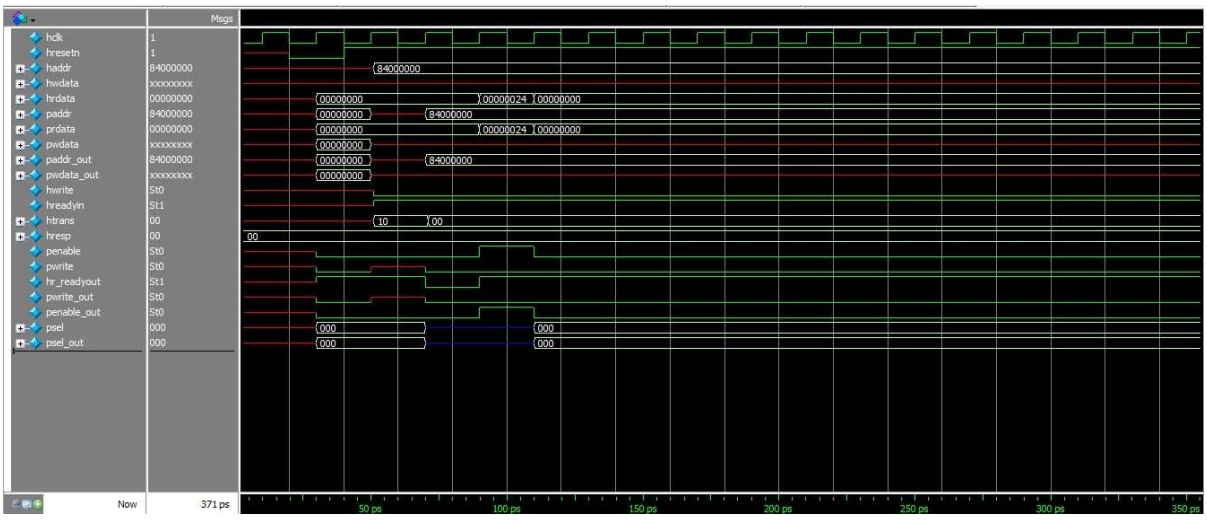
```
//ahb.burst_incr4_write(); #300;
ahb.burst_incr4_read(); #300;
$finish;
end
endmodule
```

ModelSim Waveforms:

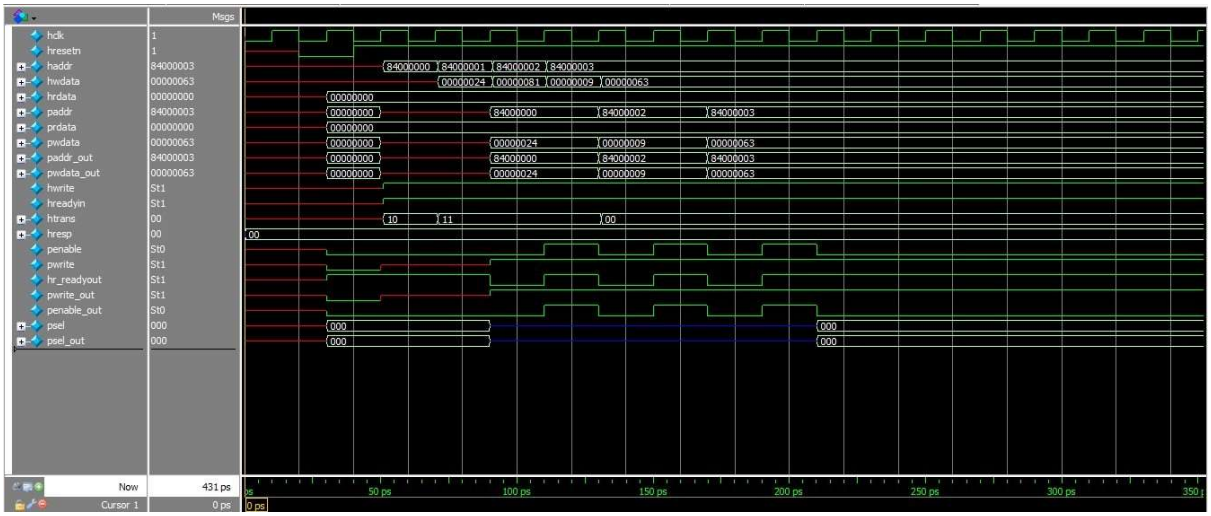
SINGLE WRITE Operation:



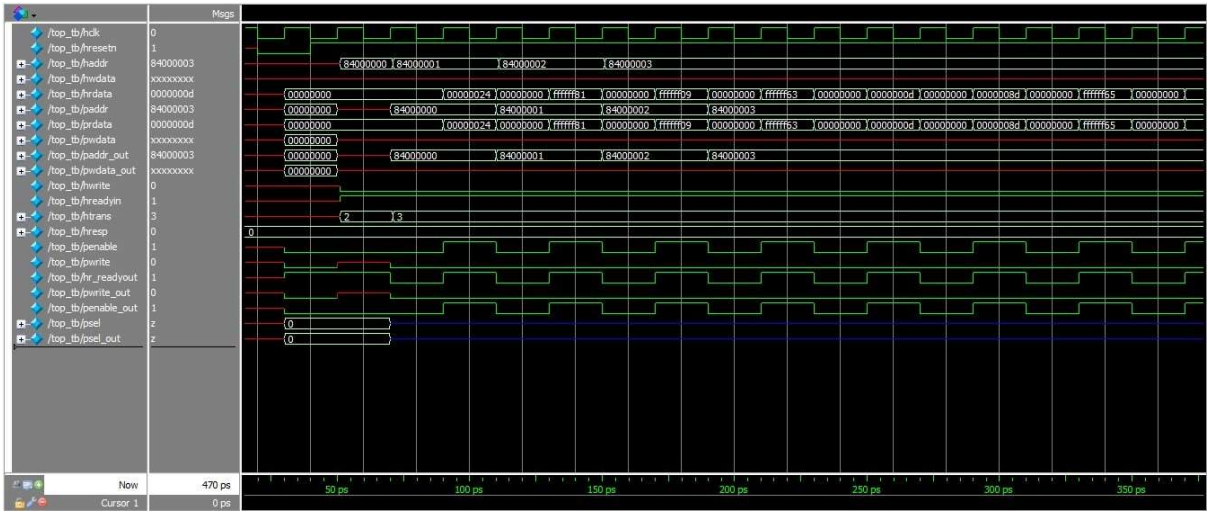
SINGLE READ Operation:



BURST WRITE with Increment 4 Burst:

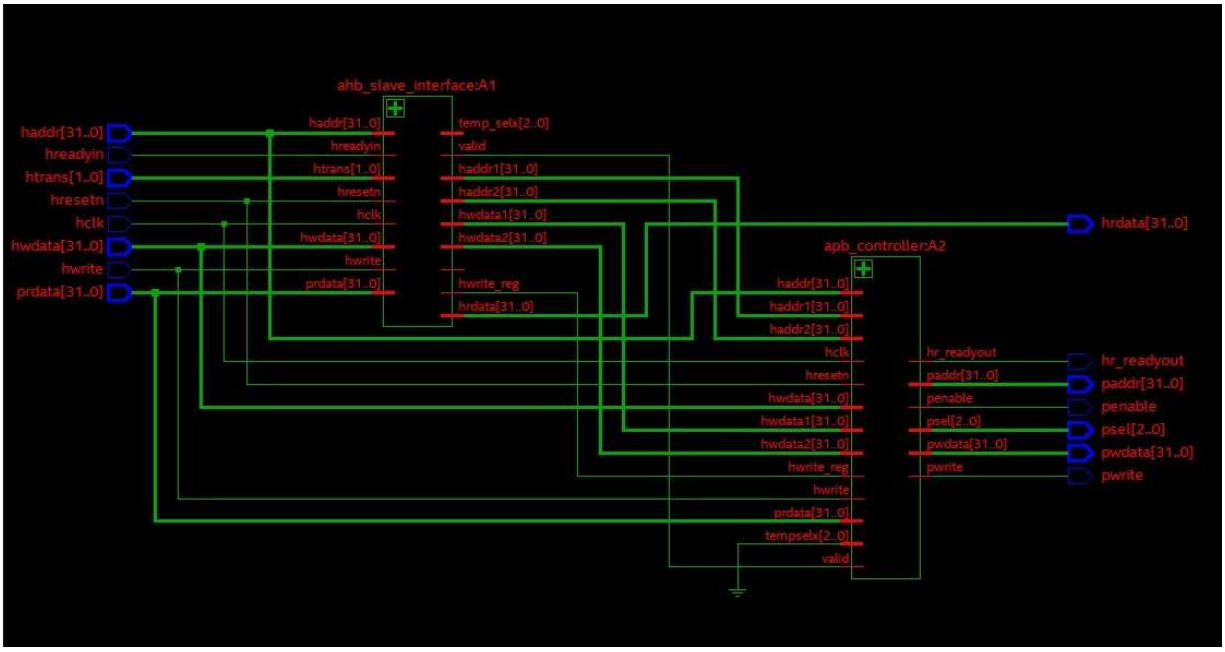


BURST READ with Increment 4 Burst:

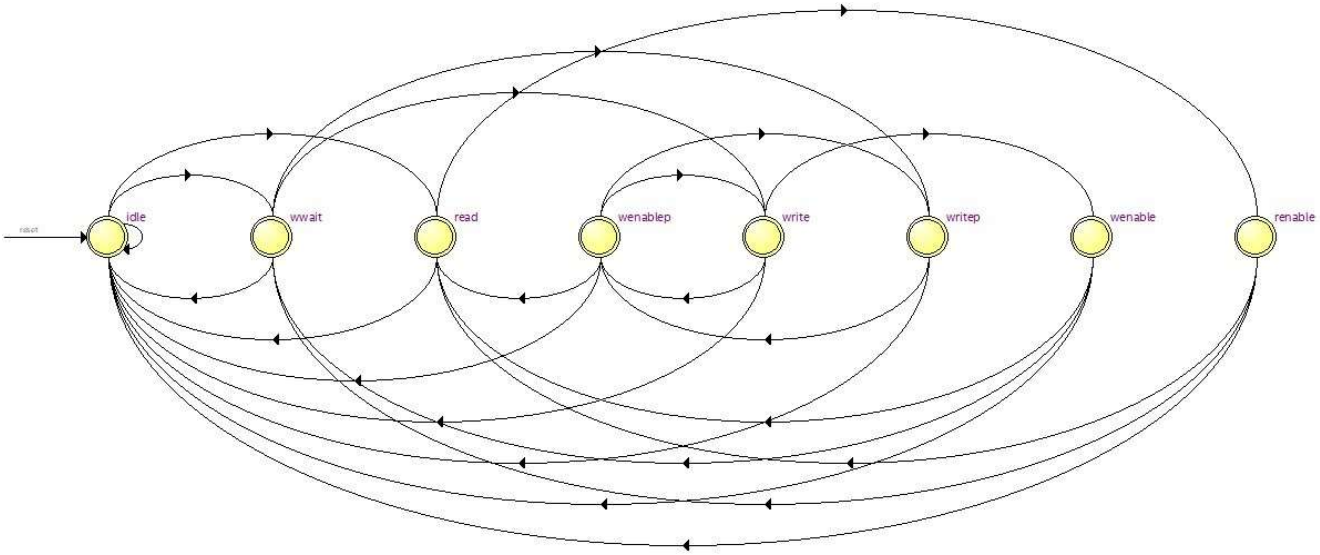


QUARTUS PRIME SCHEMATIC AND STATE DIAGRAMS:

RTL SCHEMATIC of BRIDGE:



STATE DIAGRAM of APB Controller:



STATE TABLE for APB Controller:

State Table

	Source State	Destination State	Condition
1	idle	idle	(!always2).(!hwrite) + (!always2).(hwrite).(!valid) + (!always2).(hwrite).(valid).(!hresetn) + (always2).(!hresetn)
2	idle	read	(!valid).(always2).(hresetn) + (valid).(always2).(!hwrite).(hresetn)
3	idle	wwait	(hwrite).(valid).(hresetn)
4	read	idle	(!hresetn)
5	read	renable	(hresetn)
6	renable	idle	(!always2).(!hwrite) + (!always2).(hwrite).(!valid) + (!always2).(hwrite).(valid).(!hresetn) + (always2).(!hresetn)
7	renable	read	(!valid).(always2).(hresetn) + (valid).(always2).(!hwrite).(hresetn)
8	renable	wwait	(hwrite).(valid).(hresetn)
9	wenable	idle	(!always2).(!hwrite) + (!always2).(hwrite).(!valid) + (!always2).(hwrite).(valid).(!hresetn) + (always2).(!hresetn)
10	wenable	read	(!valid).(always2).(hresetn) + (valid).(always2).(!hwrite).(hresetn)
11	wenable	wwait	(hwrite).(valid).(hresetn)
12	wenablep	idle	(!hresetn)
13	wenablep	read	(!hwrite_reg).(hresetn)
14	wenablep	write	(hwrite_reg).(!valid).(hresetn)
15	wenablep	writep	(hwrite_reg).(valid).(hresetn)
16	write	idle	(!hresetn)
17	write	wenable	(!valid).(hresetn)
18	write	wenablep	(valid).(hresetn)
19	writep	idle	(!hresetn)
20	writep	wenablep	(hresetn)
21	wwait	idle	(!hresetn)
22	wwait	write	(!valid).(hresetn)
23	wwait	writep	(valid).(hresetn)