**Hoe to use initial and always block:**

1. `timescale 1ns / 1ps
3. module tb();
5. /////global signal clk , rst
7. reg clk;
8. reg rst;
10. reg [3:0] temp;
12. //1. Initialized Global Variable
13. initial begin
14. clk = 1'b0;
15. rst = 1'b0;
16. end
17. ////2. Random signal for data/ control
19. initial begin
20. rst = 1'b1;
21. #30;
22. rst = 1'b0;
23. end

26. initial begin
27. temp = 4'b0100;
28. #10;
29. temp = 4'b1100;
30. #10;
31. temp = 4'b0011;
32. #10;
33. end

36. ///////3. System Task at the start of simulation
37. initial begin
38. $dumpfile("dump.vcd");
39. $dumpvars;
40. end
41. /////4. Analyzing Values of variable on COnsole
42. initial begin
43. $monitor("Temp : %0d at time : %0t", temp, $time);
44. end
46. ///5. Stop simulation by forcefully calling $finish
47. initial begin
48. #200;
49. $finish();
50. end
52. endmodule

**how to use the always block:**

1. `timescale 1ns / 1ps
3. module tb();

6. reg clk; //initial value = X
8. reg clk50;
9. reg clk25 = 0; ///initialize variable

12. initial begin
13. clk = 1'b0;
14. rst = 1'b0;
15. clk50 = 0;
17. end

20. always #5 clk = ~clk; //100 MHz
22. always #10 clk50 = ~clk50; ///50 MHz
24. always #20 clk25 = ~clk25; ///25 MHz




30. initial begin
31. $dumpfile("dump.vcd");
32. $dumpvars;
33. end

36. initial begin
37. #200;
38. $finish();
39. end
41. endmodule

**ALIGNING EDGES OF THE GENERATED CLOCK AND REFERENCE CLOCK:**

1. `timescale 1ns / 1ps
3. module tb();

6. reg clk; //initial value = X
8. reg clk50;
9. reg clk25 = 0; ///initialize variable

12. initial begin
13. clk = 1'b0;
14. clk50 = 0;
15. end


19. always #5 clk = ~clk;
21. always begin
22. #5;
23. clk50 = 1;
24. #10;
25. clk50 = 0;
26. #5;
27. end
29. always begin
30. #5;
31. clk25 = 1;
32. #20;
33. clk25 = 0;
34. #15;
35. end
37. initial begin
38. $dumpfile("dump.vcd");
39. $dumpvars;
40. end

43. initial begin
44. #200;
45. $finish();
46. end
48. Endmodule

**Time scale unit program:**

1. `timescale 1ns / 1ps //10^3 -> 3
3. module tb();



8. reg clk16 = 0;
9. reg clk8 = 0; ///initialize variable

12. always #31.25 clk16 = ~clk16;
13. always #62.5 clk8 = ~clk8;


17. initial begin
18. $dumpfile("dump.vcd");
19. $dumpvars;
20. end

23. initial begin
24. #200;
25. $finish();
26. end
28. endmodule

initial begin

#phase;

While (1) begin

Clk 50 =1;

#ton;

Clk50 = 0;

#toff;

End

End

**Where we want to use task and function we can efficiently use this**

1. `timescale 1ns / 1ps
3. module tb();

6. reg clk = 0;
7. reg clk50 = 0;
9. always #5 clk = ~clk; //100 MHz
11. /\*
12. real phase = 10;
13. real ton = 5;
14. real toff = 5;
15. \*/
16. /\*
17. task clkgen(input real phase, input real ton, input real toff);
18. #phase;
19. while(1) begin
20. clk50 = 1;
21. #ton;
22. clk50 = 0;
23. #toff;
24. end
25. endtask
26. \*/

29. task calc (input real freq\_hz, input real duty\_cycle, input real phase, output real pout, output real ton, output real toff);
30. pout = phase;
31. ton = (1.0 / freq\_hz) \* duty\_cycle \* 1000\_000\_000;
32. toff = (1000\_000\_000 / freq\_hz) - ton;
33. endtask

36. task clkgen(input real phase, input real ton, input real toff);
37. @(posedge clk);
38. #phase;
39. while(1) begin
40. clk50 = 1;
41. #ton;
42. clk50 = 0;
43. #toff;
44. end
45. endtask
47. real phase;
48. real ton;
49. real toff;

52. initial begin
53. calc(100\_000\_000, 0.1, 2, phase, ton, toff);
54. clkgen(phase, ton, toff);
55. end


59. initial begin
60. #200;
61. $finish();
62. end
64. endmodule

Write a code to generate a 9MHz square waveform for the signal sclk. Assume timescale with 1nsec time-unit and 3 digit precision

`timescale 1ns / 1ps //1ns/1ps = 1000 = 10^3 = 3 Digits Precision

module tb();

  reg clk;

  initial begin

    clk = 1'b0;

  end

  always #55.556 clk = ~clk;

  initial begin

    $dumpfile("dump.vcd");

    $dumpvars;

  end

  initial begin

    #500;

    $finish;

  end

endmodule

`timescale 1ns/1ps

module tb;

reg clk9 = 1'b0;

always #55.555

clk9 = ~clk9;

//9MHz

Clk

Endmodule

1. **Write a code to print the values of all the variables after 12 nSec.**
2. module tb();
3. reg[7:0] a,b;
4. integer c,d;
5. initial
6. begin
7. a=8`d12;
8. b=8`d34;
9. c=67;
10. d=255;
11. #12
12. $display("print all values a=%d b=%d c= %d d= %d",a,b,c,d);
13. end
14. endmodule

**how to copy the content to other arr?**

module tb;

int arr1[5];

int arr2[5];

initial begin

for(int i=0;i<5;i++)

begin

arr1[i] = 5\*i;

end

arr2 = arr1;

$display("arr2:%0p",arr2);

end

endmodule

**How to use the condtructor in a class?**

1. class first;
3. reg [2:0] data;
4. reg [1:0] data2;

7. endclass

10. module tb;
12. first f;
14. initial begin
15. f = new();
16. #1;
17. $display("Value of data : %0d and data2 : %0d",f.data, f.data2);
18. end


22. endmodule

**Create a Class consisting of 3 data members each of unsigned integer type. Initialize them to 45,78, and 90. Use the display function to print the values on the console.**

**class simple;**

bit [31:0] data1;

bit [31:0] data2;

bit [31:0] data3;

endclass

module tb;

simple s;

initial begin

s = new();

s.data1 = 32'd45;

s.data2 = 32'd78;

s.data3 = 32'd90;

#1;

$display (" the value of data1: %0d , data2: %0d and the data3 : %0d " , s.data1 , s.data2,s.data3);

end

endmodule

**hoe to write a simple task function:**

**module tb;**

**function bit [4:0] add(input bit [3:0]a,b);**

**return a+b;**

**endfunction**

**bit [4:0] res=0;**

**initial begin**

**res=add(4'b0100,4'b0010);**

**$display("value of addition : %0d",res);**

**end**

**endmodule**

**other usage of function**

module tb;

bit [4:0] res=0;

bit [3:0] ain=4'b0100;

bit [3:0] bin=4'b0010;

function bit[4:0] add();

return ain+bin;

endfunction

function void display\_ain\_bin();

$display("value of ain:%od andbin : %od, ain,bin);

endfunction

initial begin

res=add();

display\_ain\_bin();

end

endmodule

**hoe to use task**

**module tb;**

**bit [4:0] res=0;**

**bit [3:0] ain=4'b0100;**

**bit [3:0] bin=4'b0010;**

**function bit[4:0] add();**

**return ain+bin;**

**endfunction**

**function void display\_ain\_bin();**

**$display("value of ain:%od andbin : %od, ain,bin);**

**endfunction**

**initial begin**

**res=add();**

**display\_ain\_bin();**

**end**

**endmodule**

**how to use the task with timing constraint?**

1. module tb;


5. ///default direction : input
6. /\*
7. task add (input bit [3:0] a, input bit [3:0] b, output bit [4:0] y);
8. y = a + b;
9. endtask
10. \*/

13. bit [3:0] a,b;
14. bit [4:0] y;
16. bit clk = 0;
18. always #5 clk = ~clk; ///20 ns --> 50 Mhz
20. task add ();
21. y = a + b;
22. $display("a : %0d and b : %0d and y : %0d",a,b,y);
23. endtask
25. task stim\_a\_b();
26. a = 1;
27. b = 3;
28. add();
29. #10;
30. a = 5;
31. b = 6;
32. add();
33. #10;
34. a = 7;
35. b = 8;
36. add();
37. #10;
38. endtask

41. task stim\_clk ();
42. @(posedge clk); // wait
43. a =$urandom();
44. b = $urandom();
45. add();
46. endtask


50. initial begin
51. #110;
52. $finish();
53. end


57. initial begin
58. // stim\_a\_b();
59. for(int i = 0; i< 11 ; i++) begin
60. stim\_clk();
61. end
63. end


67. endmodule

**PASS BY REFERENCE:**

1. module tb;
3. //////pass by value
5. task swap ( input bit [1:0] a, [1:0] b);
6. bit [1:0] temp;
7. temp = a;
8. a = b;
9. b = temp;
10. $display("Value of a : %0d and b : %0d", a,b);
11. endtask





18. //////pass by reference
20. task automatic swap ( ref bit [1:0] a, [1:0] b); /// function automatic bit [1:0] add (arguments);
21. bit [1:0] temp;
22. temp = a;
23. a = b;
24. b = temp;
26. $display("Value of a : %0d and b : %0d", a,b);
27. endtask

30. ////restrict access to variables
31. task automatic swap (const ref bit [1:0] a, ref bit [1:0] b); /// function automatic bit [1:0] add (arguments);
32. bit [1:0] temp;
34. temp = a;
35. // a = b;
36. b = temp;
38. $display("Value of a : %0d and b : %0d", a,b);
39. endtask
41. bit [1:0] a;
42. bit [1:0] b;
44. initial begin
45. a = 1;
46. b = 2;
48. swap(a,b);
50. $display("Value of a : %0d and b : %0d", a,b);
51. end

54. endmodule

**the stragery for the array how its work**

1. module tb;
3. bit [3:0] res[16];
5. function automatic void init\_arr (ref bit [3:0] a[16]);
6. for(int i =0; i <= 15; i++) begin
7. a[i] = i;
8. end
9. endfunction
11. initial begin
12. init\_arr(res);
14. for(int i =0; i <= 15; i++) begin
15. $display("res[%0d] : %0d", i, res[i]);
16. end


20. end

23. endmodule

**User defined function**

1. class first;
3. int data;
5. function new(input int datain = 0);
6. data = datain;
7. endfunction

10. endclass

13. module tb;
15. first f1;
17. initial begin
18. f1 = new(23);
19. $display("Data : %0d", f1.data);
20. end

23. endmodule

**constructor using with is accessing its position**

**class first;**

**int data1;**

**bit [7:0] data2;**

**shortint data3;**

**function new(input int data1 =0,input bit [7:0] data2=8'h00,input shortint data3=0);**

**this.data1=data1;**

**this.data2=data2;**

**this.data3=data3;**

**endfunction**

**endclass**

**module tb;**

**first f1;**

**initial begin**

**f1=new(23,4,35);//following position**

**$display("data : %0d data2:%0d data3:%0d",f1.data1,f1.data2,f3.data);**

**end**

**endmodule**

accessing with data with dot operator.

**class first;**

**int data1;**

**bit [7:0] data2;**

**shortint data3;**

**function new(input int data1 =0,input bit [7:0] data2=8'h00,input shortint data3=0);**

**this.data1=data1;**

**this.data2=data2;**

**this.data3=data3;**

**endfunction**

**endclass**

**module tb;**

**first f1;**

**initial begin**

**//f1=new(23,4,35);//following position**

**f1=new(.data2(4),.data3(5),.data1(23));**

**$display("data : %0d, data2:%0d and data3:%0d",f1.data1,f1.data2,f1.data3);**

**end**

**endmodule**

**how to use task and function in inside a class?**

1. class first;
3. int data1;
4. bit [7:0] data2;
5. shortint data3;
7. function new(input int data1 = 0, input bit[7:0] data2 = 8'h00, input shortint data3 = 0);
8. this.data1 = data1;
9. this.data2 = data2;
10. this.data3 = data3;
11. endfunction
13. task display();
14. $display("Value of Data1 : %0d , Data2 : %0d and Data3 : %0d", data1, data2, data3);
15. endtask

18. endclass

21. module tb;
23. first f1;
25. initial begin
27. f1 = new( .data2(4), .data3(5), .data1(23));
28. f1.display();
29. end

32. endmodule

Assume class consists of three data members a, b, and c each of size 4-bit. Create a task inside the class that returns the result of the addition of data members. The task must also be capable of sending the value of a, b, c, and result to the console. Verify code for a = 1, b = 2, and c = 4.

// assign 15

class datamember;

  bit [2:0] a,b,c;

  bit [4:0] result;

  function new(input bit[2:0] a=0,b=0,c=0);

    this.a=a;

    this.b=b;

    this.c=c;

  endfunction

  task display();

    result=a+b+c;

    $display("result after addition of data member is",result);

  endtask

endclass

module tb;

  datamember f1;

  initial begin

    f1= new(.a(1), .b(2),.c(4));

    f1.display();

  end

endmodule

**how to access the class inside a class?**

1. class first;
3. int data = 34;
5. task display();
6. $display("Value of data : %0d",data);
7. endtask
9. endclass
11. class second;
13. first f1;
15. function new();
16. f1 = new();
17. endfunction
19. endclass

22. module tb;
24. second s;
26. initial begin
27. s = new();
28. $display("Value of Data : %0d",s.f1.data);
29. s.f1.display();
31. s.f1.data = 45;
32. s.f1.display();

35. end


39. endmodule

**SCOPE of the DATA?**

1. class first;
3. local int data = 34;
5. task set(input int data);
6. this.data = data;
7. endtask
9. function int get();
10. return data;
11. endfunction
13. task display();
14. $display("Value of data : %0d",data);
15. endtask
17. endclass
19. class second;
21. first f1;
23. function new();
24. f1 = new();
25. endfunction
27. endclass

30. module tb;
32. second s;
34. initial begin
35. s = new();
36. s.f1.set(123);
37. $display("Value of data : %0d", s.f1.get());
39. end


43. endmodule

**COPY OF THE DATA?**

1. class first;
3. int data;
5. endclass
7. module tb;
9. first f1;
10. first p1;

13. initial begin
14. f1 = new(); ///1. constructor
15. ///////////////
16. f1.data = 24; ///2. processing
17. /////////////////
19. p1 = new f1; /// 3. copying data from f1 to p1
21. /////////////
23. $display("Value of data member : %0d", p1.data); //4. processing
25. p1.data = 12;
27. $display("Value of data member : %0d", f1.data); //4. processing

30. end


34. endmodule

**strategies to copy object.**

**class first;**

**int data=12;**

**endclass**

**class second;**

**int ds=34;**

**first f1;**

**function new();**

**f1=new();**

**endfunction**

**endclass**

**module tb;**

**second s1,s2;**

**initial begin**

**s1=new();**

**s1.ds=45;**

**s2=new s1;**

**$display("value of ds:%0d",s2.ds);**

**s2.ds=78;**

**$display("value of ds:%0d",s1.ds);**

**s2.f1.data=56;**

**$display("value of data :%0d",s1.f1.data);**

**end**

**endmodule**

**SHALLOW COPY CODE?**

1. class first;
3. int data = 12;
5. endclass
7. class second;
9. int ds = 34;
11. first f1;
13. function new();
14. f1 = new();
15. endfunction

18. endclass

21. module tb;
23. second s1, s2;
25. initial begin
26. s1 = new();
28. s1.ds = 45;
30. s2 = new s1;
32. $display("Value of ds: %0d", s2.ds);
34. s2.ds = 78;
36. $display("Value of ds: %0d", s1.ds);
38. s2.f1.data = 56;
40. $display("Value of data: %0d", s1.f1.data);


44. end


48. endmodule

**DEEP COPY?**

1. class first;

4. int data = 12;
6. function first copy();
7. copy = new();
8. copy.data = data;
9. endfunction

12. endclass
14. class second;
16. int ds = 34;
18. first f1;
20. function new();
21. f1 = new();
22. endfunction
24. function second copy();
25. copy = new();
26. copy.ds = ds;
27. copy.f1 = f1.copy;
28. endfunction


32. endclass

35. module tb;
37. second s1, s2;
39. initial begin
40. s1 = new();
41. s2 = new();
43. s1.ds = 45;
45. s2 = s1.copy();
47. $display("Value of s2 : %0d", s2.ds);
49. s2.ds = 78;
51. $display("Value of s1 : %0d", s1.ds);
53. s2.f1.data = 98;
55. $display("Value of s1 : %0d", s1.f1.data);



60. end


64. endmodule

EXTENDS THE CLASS?

class first;

int data = 12;

function void display();

$display("value of data: %0d",data);

endfunction

endclass

///extended class

class second extends first;

//it will access to contain all the data member and function

int temp=34;

function void add();

$display("value after processing :%0d",temp+4);

endfunction

endclass

module tb;

//instance of second class

second s;

initial begin

s= new();

$display("value of data : %0d",s.data);

s.display();

$display("value of temp : %0d",s.temp);

s.add();

end

endmodule

**polymorphism?**

1. class first; ///parent class
3. int data = 12;
5. virtual function void display();
6. $display("FIRST : Value of data : %0d", data);
7. endfunction


11. endclass

14. class second extends first; //child class
16. int temp = 34;
18. function void add();
19. $display("secomd:Value after process : %0d", temp+4);
20. endfunction

23. function void display();
24. $display("SECOND : Value of data : %0d", data);
25. endfunction



30. endclass

33. module tb;
35. first f;
36. second s;

39. initial begin
40. f = new();
41. s = new();
43. f = s;
44. f.display();
46. end


50. endmodule

**SUPER KEYWORD?**

1. class first; ////////////parent class
2. int data;
4. function new(input int data);
5. this.data = data;
6. endfunction

9. endclass
11. class second extends first;
12. int temp;
14. function new(int data, int temp);
15. super.new(data);
16. this.temp = temp;
17. endfunction
19. endclass
21. module tb;
22. second s;
24. initial begin
25. s = new(67, 45);
26. $display("Value of data : %0d and Temp : %0d", s.data, s.temp);
27. end
29. Endmodule

**How to do randmaization with some values are repetition?**

class generator;

rand bit [3:0] a,b;// rand or randc—without repetition of same value.

bit [3:0] y;

endclass

module tb;

generator g;int i=0;

initial begin

g=new();

for(i=0;i<10;i++)//generating multiple stimulus

begin

g.randomize();

$display("value of a:%0d and b:%0d",g.a,g.b);

#10;

end

end

endmodule

**checking randomization is successful by using IF else and Assert propriety**

1. // Code your testbench here
2. //////////transaction
3. //////////////// generator
5. class generator;
7. randc bit [3:0] a, b; ////////////rand or randc
8. bit [3:0] y;

11. //constraint data {a > 15;}


15. endclass
17. module tb;
18. generator g;
19. int i = 0;
20. int status = 0;
22. initial begin
23. g = new();
25. for(i=0;i<10;i++) begin
26. /\*
27. if(!g.randomize()) begin
28. $display("Randomization Failed at %0t", $time);
29. $finish();
30. end
31. \*/
33. assert(g.randomize())else
34. begin
35. $display("Randomization Failed at %0t", $time);
36. $finish();
37. end
39. $display("Value of a :%0d and b: %0d", g.a,g.b);
40. #10;
41. end
43. end

46. endmodule

**WORKIGN WITH MULTIPLE STIMULI?**

1. class generator;
3. rand bit [3:0] a, b; ////////////rand or randc
4. bit [3:0] y;

7. endclass
9. module tb;
10. generator g;
11. int i = 0;
12. int status = 0;
14. initial begin

17. for(i=0;i<10;i++) begin
18. g = new();
19. g.randomize();
20. $display("Value of a :%0d and b: %0d", g.a,g.b);
21. #10;
22. end
24. end

27. endmodule

**SIMPLE CONSTRAINT?**

1. class generator;
3. randc bit [3:0] a, b; ////////////rand or randc
4. bit [3:0] y;
5. /\*
6. constraint data\_a {a > 3; a < 7;}
8. constraint data\_b {b == 3;}
9. \*/
11. constraint data {a > 3; a < 7 ; b > 0;}

14. endclass
16. module tb;
17. generator g;
18. int i = 0;
19. int status = 0;
21. initial begin

24. for(i=0;i<10;i++) begin
25. g = new();
26. g.randomize();
27. $display("Value of a :%0d and b: %0d", g.a,g.b);
28. #10;
29. end
31. end

34. endmodule

**RANGE OF INTERFACE: inside interface need to be used**

1. class generator;
3. randc bit [3:0] a, b; ////////////rand or randc
4. bit [3:0] y;
5. /\*
6. constraint data\_a {a > 3; a < 7;}
8. constraint data\_b {b == 3;}
9. \*/
11. //constraint data {a > 3; a < 7 ; b > 0;}
13. /\*
14. constraint data {
15. a inside {[0:8],[10:11],15} ; //0 1 2 3 4 5 6 7 8 10 11 15
16. b inside {[3:11]} ; // 3 4 5 6 7 8 9 10 11
18. }
19. \*/

22. constraint data {
23. !(a inside {[3:7]});
24. !(b inside {[5:9]});
26. }


30. ///// a = 3:7, b = 5:9
32. endclass
34. module tb;
35. generator g;
36. int i = 0;
37. int status = 0;
39. initial begin
40. g = new();
42. for(i=0;i<15;i++) begin
44. assert(g.randomize()) else $display("Randomization Failed");
45. $display("Value of a :%0d and b: %0d", g.a,g.b);
46. #10;
47. end
49. end

52. Endmodule

**How to use the constraint out the class?**

1. class generator;
3. randc bit [3:0] a, b; ////////////rand or randc
4. bit [3:0] y;
6. extern constraint data;
8. extern function void display();
10. endclass

13. constraint generator::data {
14. a inside {[0:3]};
15. b inside {[3:5]};
16. };
18. function void generator::display();
19. $display("Value of a : %0d and b : %0d", a,b);
20. endfunction



25. module tb;
26. generator g;
27. int i = 0;
28. int status = 0;
30. initial begin
31. g = new();
33. for(i=0;i<15;i++) begin
34. assert(g.randomize()) else $display("Randomization Failed");
35. g.display();
36. #10;
37. end
39. end

42. endmodule

**the function of finding probability := vs :/**

1. class first;
3. rand bit wr; // :=
4. rand bit rd; // :/
6. rand bit [1:0] var1;
7. rand bit [1:0] var2;
9. constraint data {
10. var1 dist {0 := 30, [1:3] := 90}; //0 = 30/300 , 1,2,3 = 90/300
11. var2 dist {0 :/ 30 , [1:3] :/ 90}; //0,1,2,3, = 30/120 = 0.25
13. }

16. constraint cntrl {
18. wr dist {0 := 30 , 1 := 70};
20. rd dist {0 :/ 30 , 1 :/ 70};
22. }



27. endclass

30. module tb;
32. first f;
34. initial begin
35. f = new();
37. for(int i = 0; i < 15; i++) begin
38. f.randomize();
39. $display("Value of Var1(:=) : %0d and Var2(:/) : %0d", f.var1, f.var2);
40. end
42. end

45. endmodule

**IMPLICATION OPERATOR:**

1. class generator;
3. randc bit [3:0] a;
4. rand bit ce;
5. rand bit rst;
7. constraint control\_rst {
8. rst dist {0:= 80, 1 := 20};
9. }

12. constraint control\_ce {
13. ce dist {1:= 80, 0 := 20};
14. }
16. constraint control\_rst\_ce {
17. ( rst == 0 ) <-> (ce == 1);
18. }









29. endclass
31. module tb;
33. generator g;
35. initial begin
36. g = new();
38. for (int i = 0; i<10 ; i++) begin
39. assert(g.randomize()) else $display("Randomization Failed");
40. $display("Value of rst : %0b and ce : %0b", g.rst, g.ce);
41. end
43. end

46. endmodule

**EQUIVALENCE Operator**

1. class generator;
3. randc bit [3:0] raddr, waddr;
4. rand bit wr; ///write to mem
5. rand bit oe; ///output enable
7. constraint wr\_c {
8. wr dist {0:= 50, 1 := 50};
9. }

12. constraint oe\_c {
13. oe dist {1:= 50, 0 := 50};
14. }
16. constraint wr\_oe\_c {
17. ( wr == 1 ) <-> (oe == 0);
18. }


22. endclass
24. module tb;
26. generator g;
28. initial begin
29. g = new();
31. for (int i = 0; i<10 ; i++) begin
32. assert(g.randomize()) else $display("Randomization Failed");
33. end
35. end

38. endmodule

**IF Else operator**

1. class generator;
3. randc bit [3:0] raddr, waddr;
4. rand bit wr; ///write to mem
5. rand bit oe; ///output enable
7. constraint wr\_c {
8. wr dist {0:= 50, 1 := 50};
9. }

12. constraint oe\_c {
13. oe dist {1:= 50, 0 := 50};
14. }
16. constraint wr\_oe\_c {
17. ( wr == 1 ) <-> (oe == 0);
18. }
20. constraint write\_read {
22. if(wr == 1)
23. {
24. waddr inside {[11:15]};
25. raddr == 0;
26. }
27. else
28. {
29. waddr == 0;
30. raddr inside {[11:15]};
31. }

34. }

37. endclass
39. module tb;
41. generator g;
43. initial begin
44. g = new();
46. for (int i = 0; i<15 ; i++) begin
47. assert(g.randomize()) else $display("Randomization Failed");
48. $display("Value of wr : %0b | oe : %0b | raddr : %0d | waddr : %0d |", g.wr, g.oe,g.raddr, g.waddr);
49. end
51. end

54. endmodule

**ON AND OFF CONSTRAINT:**

1. class generator;
3. randc bit [3:0] raddr, waddr;
4. rand bit wr; ///write to mem
5. rand bit oe; ///output enable
7. constraint wr\_c {
8. wr dist {0:= 50, 1 := 50};
9. }

12. constraint oe\_c {
13. oe dist {1:= 50, 0 := 50};
14. }
16. constraint wr\_oe\_c {
17. ( wr == 1 ) -> (oe == 0);
18. }


22. endclass
24. module tb;
26. generator g;
28. initial begin
29. g = new();
31. g.wr\_oe\_c.constraint\_mode(0); ///1 -> COnstraint is on // 0-> constraint is off
32. $display("Constraint Status oe\_c : %0d",g.wr\_oe\_c.constraint\_mode());
33. for (int i = 0; i<20 ; i++) begin
34. assert(g.randomize()) else $display("Randomization Failed");
35. $display("Value of wr : %0b | oe : %0b | ", g.wr, g.oe);
36. end
38. end

41. Endmodule

Generate values between 0 to 7 for addr signal when wr is high and values between 8 to 15 when wr is low. Generator code is mentioned in the Instruction tab. Verify your code for 20 iterations by sending values of both wr and addr on a console.

class generator;

randc bit [3:0] addr;

rand bit wr;

constraint wr\_c {

wr dist {0:= 50, 1 := 50};

}

constraint address\_write {

if(wr == 1)

{

addr inside {[0:7]};

}

else

{

addr inside {[8:15]};

}

}

endclass

module tb;

generator g;

initial begin

g = new();

for (int i = 0; i<20 ; i++) begin

assert(g.randomize()) else $display("Randomization Failed");

$display("Value of wr : %0d ,addr : %0d |", g.wr, g.addr);

end

end

endmodule

**how to create the testbench with respect to input of the design?**

**DESIGN:**

**module fifo(**

**input wreq,rreq,**

**input clk,**

**input rst,**

**input [7:0] wdata;**

**output [7:0] rdata;**

**output f,e);**

**endmodule**

**testbench written**

**class transaction;**

**bit clk;**

**bit rst;**

**rand bit wreq,rreq; //active high**

**rand bit [7:0] wdata;**

**bit [7:0] rdata;**

**bit empty;**

**bit full;**

**constraint ctrl\_wr{**

**wreq dist {0 :=30; 1:=70};**

**}**

**constraint ctrl\_rd{**

**rreq dist{0:= 30; 1:=70;};**

**}**

**constraint wr\_rd{**

**wreq != rreq;**

**}**

**Endclass**

**Event triggering:**

**module tb;**

**event a1,a2;**

**initial begin**

**->a1;**

**->a2;**

**end**

**initial begin**

**wait(a1.triggered);**

**$display("Event A1 Trigger");**

**wait(a2.triggered);**

**$display("Event A2 Trigger");**

**end**

**endmodule**

**HOW TO DESIGN THE GENERATOR AND DRIVER AND TO FIND THE WHEATER THE PACKET SENT BY THE GENERATOR RECEIVED BY THE DRIVER**

1. module tb;
3. int data1,data2;
4. event done;
6. int i =0;
8. ////Generator
9. initial begin
10. for(i =0; i< 10; i++) begin
11. data1 = $urandom();
12. $display("Data Sent : %0d", data1);
13. #10;
14. end
15. -> done;
16. end
18. ///////Driver
20. initial begin
21. forever begin
22. #10;
23. data2 = data1;
24. $display("Data RCVD : %0d", data2);
25. end
26. end
27. /////////////
29. initial begin
30. wait(done.triggered);
31. $finish();
32. end




38. endmodule

**GENERATING AND SENDING THE STIMULUS BY USING BY USING FORK AND JOIN USING IN DRIVER AND GENERATOR;**

1. module tb;
2. int i =0;
3. bit [7:0] data1,data2;
4. event done;
5. event next;
7. task generator();
8. for(i = 0; i<10; i++) begin
9. data1 = $urandom();
10. $display("Data Sent : %0d", data1);
11. #10;
13. wait(next.triggered);
15. end
17. -> done;
18. endtask


22. task receiver();
23. forever begin
24. #10;
25. data2 = data1;
26. $display("Data RCVD : %0d",data2);
27. ->next;
28. end
30. endtask



35. task wait\_event();
36. wait(done.triggered);
37. $display("Completed Sending all Stimulus");
38. $finish();
39. endtask


43. initial begin
44. fork
45. generator();
46. receiver();
47. wait\_event();
48. join
50. ///////
52. end

55. endmodule

**HOW TO WORK WITH FROAK ADD JOIN**

1. module tb;
3. task first();
4. $display("Task 1 Started at %0t",$time);
5. #20;
6. $display("Task 1 Completed at %0t",$time);
7. endtask


11. task second();
12. $display("Task 2 Started at %0t",$time);
13. #30;
14. $display("Task 2 Completed at %0t",$time);
15. endtask


19. task third();
20. $display("Reached next to Join at %0t",$time);
21. endtask

24. initial begin



29. fork
31. first();
32. second();
34. join\_none

37. third();

40. end


44. endmodule

**HOW TO WORK WITH SEMAPHORE?**

**semaphore sem;**

**first f;**

**second s;**

**int data;**

**int i = 0;**

**task send\_first();**

**sem.get(1);**

**for(i = 0; i<10; i++) begin**

**f.randomize();**

**data = f.data;**

**$display("First access Semaphore and Data sent : %0d", f.data);**

**#10;**

**end**

**sem.put(1);**

**$display("Semaphore Unoccupied");**

**endtask**

**task send\_second();**

**sem.get(1);**

**for(i = 0; i<10; i++) begin**

**s.randomize();**

**data = s.data;**

**$display("Second access Semaphore and Data sent : %0d", s.data);**

**#10;**

**end**

**sem.put(1);**

**$display("Semaphore Unoccupied");**

**endtask**

**task run();**

**sem = new(1);**

**f = new();**

**s = new();**

**fork**

**send\_first();**

**send\_second();**

**join**

**endtask**

**endclass**

**module tb;**

**main m;**

**initial begin**

**m = new();**

**m.run();**

**end**

**initial begin**

**#250;**

**$finish();**

**end**

**endmodule**

**HOW TO COMMUNICATE GENERATIOR AND DRIVER WITH MAILBOX:**

1. class generator;
3. int data = 12;
4. mailbox mbx;

7. task run();
8. mbx.put(data);
9. $display("[GEN] : Data Send from Gen : %0d ",data);
10. endtask
12. endclass
14. class driver;
15. mailbox mbx;
16. int data;
18. task run();
19. mbx.get(data);
20. $display("[DRV] : DATA rcvd : %0d",data);
21. endtask

24. endclass

27. module tb;
28. generator gen;
29. driver drv;
30. mailbox mbx;
32. initial begin
33. gen = new();
34. drv = new();
35. mbx = new();
37. gen.mbx = mbx;
38. drv.mbx = mbx;
40. gen.run();
41. drv.run();
42. end

45. endmodule

**CREATING SPECIFYING MAILBOX WITH CUSTOM CONSTRUCTOR:**

1. class generator;
3. int data = 56;
5. mailbox mbx; ///gen2drv
7. function new(mailbox mbx);
8. this.mbx = mbx;
9. endfunction
11. task run();
12. mbx.put(data);
13. $display("[GEN] : SENT DATA : %0d", data);
14. endtask

17. endclass

20. class driver;
21. int datac = 0;
22. mailbox mbx;


26. function new(mailbox mbx);
27. this.mbx = mbx;
28. endfunction

31. task run();
32. mbx.get(datac);
33. $display("[DRV] : RCVD Data : %0d", datac);
34. endtask

37. endclass

40. module tb;
42. generator gen;
43. driver drv;
44. mailbox mbx;

47. initial begin
48. mbx = new();
50. gen = new(mbx);
51. drv = new(mbx);


55. gen.run();
56. drv.run();

59. end


63. endmodule

**HOW TO USE TRANCTION CLASS IN GENERATOR AND DRIVER:**

class transaction;

rand bit [3:0] din1;

rand bit [3:0] din2;

bit [4:0] dout;

endclass

class generator;

transaction t;//handler

mailbox mbx;

function new(mailbox mbx);

this.mbx = mbx;

endfunction

task main();

for(int i = 0; i < 10; i++) begin

t = new();

assert(t.randomize) else $display("Randomization Failed");

$display("[GEN] : DATA SENT : din1 : %0d and din2 : %0d", t.din1, t.din2);

mbx.put(t);

#10;

end

endtask

endclass

class driver;

transaction dc;

mailbox mbx;

function new(mailbox mbx);

this.mbx = mbx;

endfunction

task main();

forever begin

mbx.get(dc);

$display("[DRV] : DATA RCVD : din1 : %0d and din2 : %0d", dc.din1, dc.din2);

#10;

end

endtask

endclass

module tb;

generator g;

driver d;

mailbox mbx;

initial begin

mbx = new();

g = new(mbx);

d = new(mbx);

fork

g.main();

d.main();

join

end

endmodule

**how to work with parameterized class in generator and driver class**

1. class transaction;
3. bit [7:0] data;
5. endclass

8. class generator;
10. int data = 12;
11. transaction t;
13. mailbox #(transaction) mbx;
15. logic [7:0] temp = 3;
17. function new(mailbox #(transaction) mbx);
18. this.mbx = mbx;
19. endfunction

22. task run();
23. t = new();
24. t.data = 45;
25. mbx.put(temp);
26. $display("[GEN] : Data Send from Gen : %0d ",t.data);
27. endtask
29. endclass
31. class driver;
32. mailbox #(transaction) mbx;
33. transaction data;
35. function new(mailbox #(transaction) mbx);
36. this.mbx = mbx;
37. endfunction


41. task run();
42. mbx.get(data);
43. $display("[DRV] : DATA rcvd : %0d",data.data);
44. endtask

47. endclass

50. module tb;
51. generator gen;
52. driver drv;
53. mailbox #(transaction) mbx;
55. initial begin
57. mbx = new();
58. gen = new(mbx);
59. drv = new(mbx);
61. gen.run();
62. drv.run();
63. end

66. endmodule

Code for transaction class is mentioned in the Instruction tab. Write a code to send transaction data between generator and driver. Also, verify the data by printing the value of data members of Generator and Driver.

class transaction;

bit [7:0] addr = 7'h12;

bit [3:0] data = 4'h4;

bit we = 1'b1;

bit rst = 1'b0;

endclass

class generator;

  transaction t;

  mailbox #(transaction)  mbx;

  //logic [7:0] temp = 3;

  function new(mailbox #(transaction) mbx);

    this.mbx = mbx;

  endfunction

  task run();

    t = new();

    //t.data = 45;

    $display("[GEN] : Addr Send from Gen : %0d ",t.addr);

    $display("[GEN] : data Send from Gen : %0d ",t.data);

    $display("[GEN] : we Send from Gen : %0d ",t.we);

    $display("[GEN] : rst Send from Gen : %0d ",t.rst);

    mbx.put(t);

    #10;

  endtask

endclass

class driver;

  mailbox #(transaction) mbx;

  transaction dc;

  function new(mailbox #(transaction)  mbx);

    this.mbx = mbx;

  endfunction

  task run();

    mbx.get(dc);

    $display("[DRV] : addr rcvd : %0d",dc.addr);

    $display("[DRV] : data rcvd : %0d",dc.data);

  endtask

endclass

module tb;

  generator gen;

  driver drv;

  mailbox #(transaction) mbx;

  initial begin

    mbx = new();

    gen = new(mbx);

    drv = new(mbx);

    gen.run();

    drv.run();

  end

endmodule

**ADDING THE INTERFACE WITH SIMPLE DUT**

**DESIGN:**

1. module and4 (
2. input [3:0] a,
3. input [3:0] b,
4. output [3:0] y
6. );
8. assign y = a & b;

11. endmodule

**Interface**

1. interface and\_if;
2. logic [3:0] a;
3. logic [3:0] b;
4. logic [3:0] y;
6. endinterface

9. module tb;
11. and\_if aif();
13. and4 dut (.a(aif.a), .b(aif.b), .y(aif.y));
15. initial begin
16. aif.a = 4'b0100;
17. aif.b = 4'b1100;
18. #10;
19. $display("a : %b , b : %b and y : %b",aif.a, aif.b, aif.y );
20. end
22. endmodule

**ADDING DRIVER TO THE DUT**

**DESIGN:**

1. module add
2. (
3. input [3:0] a,b,
4. output reg [4:0] sum,
5. input clk
6. );

9. always@(posedge clk)
10. begin
11. sum <= a + b;
12. end

15. endmodule

**TEST BENCH**

1. interface add\_if;
2. logic [3:0] a;
3. logic [3:0] b;
4. logic [4:0] sum;
5. logic clk;
6. endinterface

9. class driver;
11. virtual add\_if aif;
13. task run();
14. forever begin
15. @(posedge aif.clk);
16. aif.a <= 2;
17. aif.b <= 3;
18. $display("[DRV] : Interface Trigger");
19. end
20. endtask

23. endclass


27. module tb;
29. add\_if aif();
30. driver drv;
32. add dut (aif.a, aif.b, aif.sum, aif.clk );

35. initial begin
36. aif.clk <= 0;
37. end
39. always #10 aif.clk <= ~aif.clk;
41. initial begin
42. drv = new();
43. drv.aif = aif;
44. drv.run();
46. end
48. initial begin
49. $dumpfile("dump.vcd");
50. $dumpvars;
51. #100;
52. $finish();
53. end
55. endmodule

**add transaction constructor in generator custom constructor**

**send deep copy of transaction between generator and driver …**

1. class transaction;
3. randc bit [3:0] a;
4. randc bit [3:0] b;
6. function void display();
7. $display("a : %0d \t b: %0d ", a,b);
8. endfunction
10. function transaction copy();
11. copy = new();
12. copy.a = this.a;
13. copy.b = this.b;
14. endfunction
16. endclass


20. class generator;
22. transaction trans;
23. mailbox #(transaction) mbx;
24. int i = 0;
26. function new(mailbox #(transaction) mbx);
27. this.mbx = mbx;
28. trans = new();
29. endfunction
31. task run();
32. for(i = 0; i<20; i++) begin
33. assert(trans.randomize()) else $display("Randomization Failed");
34. $display("[GEN] : DATA SENT TO DRIVER");
35. trans.display();
36. mbx.put(trans.copy);
37. end
39. endtask

42. endclass


46. module tb;
48. generator gen;
49. mailbox #(transaction) mbx;


53. initial begin
54. mbx = new();
55. gen = new(mbx);
56. gen.run();
57. end

60. endmodule

How to generate non repeated random value and also get the generated value is received to generator to driver by using event triggering:

1. class transaction;
2. randc bit [3:0] a;
3. randc bit [3:0] b;
4. bit [4:0] sum;
6. function void display();
7. $display("a : %0d \t b: %0d \t sum : %0d",a,b,sum);
8. endfunction
10. function transaction copy();
11. copy = new();
12. copy.a = this.a;
13. copy.b = this.b;
14. copy.sum = this.sum;
15. endfunction
17. endclass

20. class generator;
22. transaction trans;
23. mailbox #(transaction) mbx;
24. event done;
26. function new(mailbox #(transaction) mbx);
27. this.mbx = mbx;
28. trans = new();
29. endfunction

32. task run();
33. for(int i = 0; i<10; i++) begin
34. trans.randomize();
35. mbx.put(trans.copy);
36. $display("[GEN] : DATA SENT TO DRIVER");
37. trans.display();
38. #20;
39. end
40. -> done;
41. endtask
43. endclass




49. interface add\_if;
50. logic [3:0] a;
51. logic [3:0] b;
52. logic [4:0] sum;
53. logic clk;
54. endinterface

57. class driver;
59. virtual add\_if aif;
60. mailbox #(transaction) mbx;
61. transaction data;
62. event next;
64. function new(mailbox #(transaction) mbx);
65. this.mbx = mbx;
66. endfunction

69. task run();
70. forever begin
71. mbx.get(data);
72. @(posedge aif.clk);
73. aif.a <= data.a;
74. aif.b <= data.b;
75. $display("[DRV] : Interface Trigger");
76. data.display();
77. end
78. endtask

81. endclass


85. module tb;
87. add\_if aif();
88. driver drv;
89. generator gen;
90. event done;

93. mailbox #(transaction) mbx;
95. add dut (aif.a, aif.b, aif.sum, aif.clk );

98. initial begin
99. aif.clk <= 0;
100. end
102. always #10 aif.clk <= ~aif.clk;
104. initial begin
105. mbx = new();
106. drv = new(mbx);
107. gen = new(mbx);
108. drv.aif = aif;
109. done = gen.done;
110. end
112. initial begin
113. fork
114. gen.run();
115. drv.run();
116. join\_none
117. wait(done.triggered);
118. $finish();
119. end

122. initial begin
123. $dumpfile("dump.vcd");
124. $dumpvars;
125. end
127. endmodule

**How to inject the error in class**

1. class transaction;
2. randc bit [3:0] a;
3. randc bit [3:0] b;
4. bit [4:0] sum;

7. function void display();
8. $display("a : %0d \t b: %0d \t sum : %0d",a,b,sum);
9. endfunction

12. endclass
14. class error extends transaction;
16. constraint data\_c {a == 0; b == 0;}
18. endclass



23. class generator;
25. transaction trans;
26. mailbox #(transaction) mbx;
27. event done;

30. function new(mailbox #(transaction) mbx);
31. this.mbx = mbx;
32. endfunction

35. task run();
36. trans = new();
37. for(int i = 0; i<10; i++) begin
38. trans.randomize();
39. mbx.put(trans);
40. $display("[GEN] : DATA SENT TO DRIVER");
41. trans.display();
42. #20;
43. end
44. -> done;
45. endtask
47. endclass







56. interface add\_if;
57. logic [3:0] a;
58. logic [3:0] b;
59. logic [4:0] sum;
60. logic clk;
62. modport DRV (input a,b, input sum,clk);


66. endinterface

69. class driver;
71. virtual add\_if aif;
73. mailbox #(transaction) mbx;
75. transaction data;
77. function new(mailbox #(transaction) mbx);
78. this.mbx = mbx;
79. endfunction

82. task run();
83. forever begin
84. mbx.get(data);
85. @(posedge aif.clk);
86. aif.a <= data.a;
87. aif.b <= data.b;
88. $display("[DRV] : Interface Trigger");
89. data.display();
90. end
91. endtask

94. endclass



99. module tb;
101. add\_if aif();
102. driver drv;
103. generator gen;
104. error err;

107. event done;

110. mailbox #(transaction) mbx;
112. add dut (aif.a, aif.b, aif.sum, aif.clk );

115. initial begin
116. aif.clk <= 0;
117. end
119. always #10 aif.clk <= ~aif.clk;
121. initial begin
122. mbx = new();
123. err = new();
124. drv = new(mbx);
125. gen = new(mbx);
127. gen.trans = err;
129. drv.aif = aif;
130. done = gen.done;


134. end
136. initial begin
137. fork
138. gen.run();
139. drv.run();
140. join\_none
141. wait(done.triggered);
142. $finish();
143. end

146. initial begin
147. $dumpfile("dump.vcd");
148. $dumpvars;
149. end
151. endmodule

**similar to the above code**

1. class transaction;
2. randc bit [3:0] a;
3. randc bit [3:0] b;
4. bit [4:0] sum;

7. function void display();
8. $display("a : %0d \t b: %0d \t sum : %0d",a,b,sum);
9. endfunction
11. virtual function transaction copy();
12. copy = new();
13. copy.a = this.a;
14. copy.b = this.b;
15. copy.sum = this.sum;
16. endfunction
18. endclass
20. class error extends transaction;
22. constraint data\_c {a == 0 ; b == 0;}
24. function transaction copy();
25. copy = new();
26. copy.a = 12;
27. copy.b = 12;
28. copy.sum = this.sum;
29. endfunction
31. endclass

34. class generator;
36. transaction trans;
37. mailbox #(transaction) mbx;
38. event done;

41. function new(mailbox #(transaction) mbx);
42. this.mbx = mbx;
43. trans = new();
44. endfunction

47. task run();
48. for(int i = 0; i<10; i++) begin
49. trans.randomize();
50. mbx.put(trans.copy);
51. $display("[GEN] : DATA SENT TO DRIVER");
52. trans.display();
53. #20;
54. end
55. -> done;
56. endtask
58. endclass




64. interface add\_if;
65. logic [3:0] a;
66. logic [3:0] b;
67. logic [4:0] sum;
68. logic clk;
69. endinterface

72. class driver;
74. virtual add\_if aif;
75. mailbox #(transaction) mbx;
76. transaction data;
78. function new(mailbox #(transaction) mbx);
79. this.mbx = mbx;
80. endfunction

83. task run();
84. forever begin
85. mbx.get(data);
86. @(posedge aif.clk);
87. aif.a <= data.a;
88. aif.b <= data.b;
89. $display("[DRV] : Interface Trigger");
90. data.display();
91. end
92. endtask

95. endclass


99. module tb;
101. add\_if aif();
102. driver drv;
103. generator gen;
104. event done;
105. event next;
106. error er;

109. mailbox #(transaction) mbx;
111. add dut (aif.a, aif.b, aif.sum, aif.clk );

114. initial begin
115. aif.clk <= 0;
116. end
118. always #10 aif.clk <= ~aif.clk;
120. initial begin
121. mbx = new();
122. drv = new(mbx);
123. gen = new(mbx);
124. drv.aif = aif;
125. er = new();
126. gen.trans = er;
127. done = gen.done;
129. end
131. initial begin
132. fork
133. gen.run();
134. drv.run();
135. join\_none
136. wait(done.triggered);
137. $finish();
138. end

141. initial begin
142. $dumpfile("dump.vcd");
143. $dumpvars;
144. end
146. endmodule

**how to connect monitor and scoreboard through put and get method**

1. class transaction;
2. randc bit [3:0] a;
3. randc bit [3:0] b;
4. bit [4:0] sum;

7. function void display();
8. $display("a : %0d \t b: %0d \t sum : %0d",a,b,sum);
9. endfunction

12. endclass
14. interface add\_if;
15. logic [3:0] a;
16. logic [3:0] b;
17. logic [4:0] sum;

logic clk;

1. endinterface

4. class monitor;
6. mailbox #(transaction) mbx;
7. transaction trans;
8. virtual add\_if aif;
10. function new(mailbox #(transaction) mbx);
11. this.mbx = mbx;
12. endfunction
14. task run();
15. trans = new();
16. forever begin
17. @(posedge aif.clk);
18. trans.a = aif.a;
19. trans.b = aif.b;
20. trans.sum = aif.sum;
21. $display("[MON] : DATA SENT TO SCOREBOARD");
22. trans.display();
23. mbx.put(trans);
24. end
25. endtask

28. endclass

31. class scoreboard;
33. mailbox #(transaction) mbx;
34. transaction trans;
36. function new(mailbox #(transaction) mbx);
37. this.mbx = mbx;
38. endfunction
40. task run();
41. forever begin
42. mbx.get(trans);
43. $display("[SCO] : DATA RCVD FROM MONITOR");
44. trans.display();
45. #20;
46. end
47. endtask

50. endclass







59. module tb;
61. add\_if aif();
62. monitor mon;
63. scoreboard sco;
64. mailbox #(transaction) mbx;




70. add dut (aif.a, aif.b, aif.sum, aif.clk );

73. initial begin
74. aif.clk <= 0;
75. end
77. always #10 aif.clk <= ~aif.clk;
79. initial begin
80. for(int i = 0; i < 20 ; i++) begin
81. @(posedge aif.clk);
82. aif.a <= $urandom\_range(0,15);
83. aif.b <= $urandom\_range(0,15);
84. end
85. end
87. initial begin
88. mbx = new();
89. mon = new(mbx);
90. sco = new(mbx);
91. mon.aif = aif;
93. end
95. initial begin
96. fork
97. mon.run();
98. sco.run();
99. join
100. end
101. initial begin
102. $dumpfile("dump.vcd");
103. $dumpvars;
104. #450;
105. $finish();
106. end

1. endmodule

**to get the relevant output along with input like little bit delay**

1. class transaction;
2. randc bit [3:0] a;
3. randc bit [3:0] b;
4. bit [4:0] sum;

7. function void display();
8. $display("a : %0d \t b: %0d \t sum : %0d",a,b,sum);
9. endfunction

12. endclass
14. interface add\_if;
15. logic [3:0] a;
16. logic [3:0] b;
17. logic [4:0] sum;
18. logic clk;
19. endinterface

22. class monitor;
24. mailbox #(transaction) mbx;
25. transaction trans;
26. virtual add\_if aif;
28. function new(mailbox #(transaction) mbx);
29. this.mbx = mbx;
30. endfunction
32. task run();
33. trans = new();
34. forever begin
35. repeat(2) @(posedge aif.clk);
36. trans.a = aif.a;
37. trans.b = aif.b;
38. trans.sum = aif.sum;
39. $display("[MON] : DATA SENT TO SCOREBOARD");
40. trans.display();
41. mbx.put(trans);
42. end
43. endtask

46. endclass

49. class scoreboard;
51. mailbox #(transaction) mbx;
52. transaction trans;
54. function new(mailbox #(transaction) mbx);
55. this.mbx = mbx;
56. endfunction
58. task run();
59. forever begin
60. mbx.get(trans);
61. $display("[SCO] : DATA RCVD FROM MONITOR");
62. trans.display();
63. #40;
64. end
65. endtask

68. endclass







77. module tb;
79. add\_if aif();
80. monitor mon;
81. scoreboard sco;
82. mailbox #(transaction) mbx;




88. add dut (aif.a, aif.b, aif.sum, aif.clk );

91. initial begin
92. aif.clk <= 0;
93. end
95. always #10 aif.clk <= ~aif.clk;
97. initial begin
98. for(int i = 0; i < 20 ; i++) begin
99. repeat(2) @(posedge aif.clk);
100. aif.a <= $urandom\_range(0,15);
101. aif.b <= $urandom\_range(0,15);
102. end
103. end
105. initial begin
106. mbx = new();
107. mon = new(mbx);
108. sco = new(mbx);
109. mon.aif = aif;
111. end
113. initial begin
114. fork
115. mon.run();
116. sco.run();
117. join
118. end
119. initial begin
120. $dumpfile("dump.vcd");
121. $dumpvars;
122. #450;
123. $finish();
124. end
126. endmodule

**SIMPLE ADDER TESTBENCH**

**DESIGN CODE:**

module add

(

input [3:0] a,b;

output reg[4:0] sum,

input clk);

always@(posedge clk)

begin

sum<=a+b;

end

endmodule

1. class transaction;
2. randc bit [3:0] a;
3. randc bit [3:0] b;
4. bit [4:0] sum;

7. function void display();
8. $display("a : %0d \t b: %0d \t sum : %0d",a,b,sum);
9. endfunction

12. endclass
14. interface add\_if;
15. logic [3:0] a;
16. logic [3:0] b;
17. logic [4:0] sum;
18. logic clk;
19. endinterface

22. class monitor;
24. mailbox #(transaction) mbx;
25. transaction trans;
26. virtual add\_if aif;
28. function new(mailbox #(transaction) mbx);
29. this.mbx = mbx;
30. endfunction
32. task run();
33. trans = new();
34. forever begin
35. repeat(2) @(posedge aif.clk);
36. trans.a = aif.a;
37. trans.b = aif.b;
38. trans.sum = aif.sum;
39. $display("-------------------------");
40. $display("[MON] : DATA SENT TO SCOREBOARD");
41. trans.display();
42. mbx.put(trans);
43. end
44. endtask

47. endclass

50. class scoreboard;
52. mailbox #(transaction) mbx;
53. transaction trans;
55. function new(mailbox #(transaction) mbx);
56. this.mbx = mbx;
57. endfunction
59. task compare(input transaction trans);
60. if((trans.sum) == (trans.a + trans.b))
61. $display("[SCO] : Sum Result Matched");
62. else
63. $error("[SCO] : Result Mismatched");
64. endtask
66. task run();
67. forever begin
68. mbx.get(trans);
69. $display("[SCO] : DATA RCVD FROM MONITOR");
70. trans.display();
71. compare(trans);
72. $display("-------------------------");
73. #40;
74. end
75. endtask

78. endclass







87. module tb;
89. add\_if aif();
90. monitor mon;
91. scoreboard sco;
92. mailbox #(transaction) mbx;




98. add dut (aif.a, aif.b, aif.sum, aif.clk );

101. initial begin
102. aif.clk <= 0;
103. end
105. always #10 aif.clk <= ~aif.clk;
107. initial begin
108. for(int i = 0; i < 20 ; i++) begin
109. repeat(2) @(posedge aif.clk);
110. aif.a <= $urandom\_range(0,15);
111. aif.b <= $urandom\_range(0,15);
112. end
113. end
115. initial begin
116. mbx = new();
117. mon = new(mbx);
118. sco = new(mbx);
119. mon.aif = aif;
121. end
123. initial begin
124. fork
125. mon.run();
126. sco.run();
127. join
128. end
129. initial begin
130. $dumpfile("dump.vcd");
131. $dumpvars;
132. #450;
133. $finish();
134. end
136. endmodule

Systemverilog projects:

DD FF

1. module dff (dff\_if vif);
3. // Always block triggered on the positive edge of the clock signal
4. always @(posedge vif.clk)
5. begin
6. // Check if the reset signal is asserted
7. if (vif.rst == 1'b1)
8. // If reset is active, set the output to 0
9. vif.dout <= 1'b0;
10. else
11. // If reset is not active, pass the input value to the output
12. vif.dout <= vif.din;
13. end
15. endmodule
17. // Define an interface "dff\_if" with the following signals
18. interface dff\_if;
19. logic clk; // Clock signal
20. logic rst; // Reset signal
21. logic din; // Data input
22. logic dout; // Data output
24. endinterface
25. class transaction;
26. rand bit din; // Define a random input bit "din"
27. bit dout; // Define an output bit "dout"
29. function transaction copy();
30. copy = new(); // Create a new transaction object
31. copy.din = this.din; // Copy the input value
32. copy.dout = this.dout; // Copy the output value
33. endfunction
35. function void display(input string tag);
36. $display("[%0s] : DIN : %0b DOUT : %0b", tag, din, dout); // Display transaction information
37. endfunction
39. endclass
41. //////////////////////////////////////////////////
43. class generator;
44. transaction tr; // Define a transaction object
45. mailbox #(transaction) mbx; // Create a mailbox to send data to the driver
46. mailbox #(transaction) mbxref; // Create a mailbox to send data to the scoreboard for comparison/golden data
47. event sconext; // Event to sense the completion of scoreboard work
48. event done; // Event to trigger when the requested number of stimuli is applied
49. int count; // Stimulus count
51. function new(mailbox #(transaction) mbx, mailbox #(transaction) mbxref);
52. this.mbx = mbx; // Initialize the mailbox for the driver
53. this.mbxref = mbxref; // Initialize the mailbox for the scoreboard
54. tr = new(); // Create a new transaction object
55. endfunction
57. task run();
58. repeat(count) begin
59. assert(tr.randomize) else $error("[GEN] : RANDOMIZATION FAILED");
60. mbx.put(tr.copy); // Put a copy of the transaction into the driver mailbox
61. mbxref.put(tr.copy); // Put a copy of the transaction into the scoreboard mailbox
62. tr.display("GEN"); // Display transaction information
63. @(sconext); // Wait for the scoreboard's completion signal
64. end
65. ->done; // Trigger "done" event when all stimuli are applied
66. endtask
68. endclass
70. //////////////////////////////////////////////////////////
72. class driver;
73. transaction tr; // Define a transaction object
74. mailbox #(transaction) mbx; // Create a mailbox to receive data from the generator
75. virtual dff\_if vif; // Virtual interface for DUT
77. function new(mailbox #(transaction) mbx);
78. this.mbx = mbx; // Initialize the mailbox for receiving data
79. endfunction
81. task reset();
82. vif.rst <= 1'b1; // Assert reset signal
83. repeat(5) @(posedge vif.clk); // Wait for 5 clock cycles
84. vif.rst <= 1'b0; // Deassert reset signal
85. @(posedge vif.clk); // Wait for one more clock cycle
86. $display("[DRV] : RESET DONE"); // Display reset completion message
87. endtask
89. task run();
90. forever begin
91. mbx.get(tr); // Get a transaction from the generator
92. vif.din <= tr.din; // Set DUT input from the transaction
93. @(posedge vif.clk); // Wait for the rising edge of the clock
94. tr.display("DRV"); // Display transaction information
95. vif.din <= 1'b0; // Set DUT input to 0
96. @(posedge vif.clk); // Wait for the rising edge of the clock
97. end
98. endtask
100. endclass
102. //////////////////////////////////////////////////////
104. class monitor;
105. transaction tr; // Define a transaction object
106. mailbox #(transaction) mbx; // Create a mailbox to send data to the scoreboard
107. virtual dff\_if vif; // Virtual interface for DUT
109. function new(mailbox #(transaction) mbx);
110. this.mbx = mbx; // Initialize the mailbox for sending data to the scoreboard
111. endfunction
113. task run();
114. tr = new(); // Create a new transaction
115. forever begin
116. repeat(2) @(posedge vif.clk); // Wait for two rising edges of the clock
117. tr.dout = vif.dout; // Capture DUT output
118. mbx.put(tr); // Send the captured data to the scoreboard
119. tr.display("MON"); // Display transaction information
120. end
121. endtask
123. endclass
125. ////////////////////////////////////////////////////
127. class scoreboard;
128. transaction tr; // Define a transaction object
129. transaction trref; // Define a reference transaction object for comparison
130. mailbox #(transaction) mbx; // Create a mailbox to receive data from the driver
131. mailbox #(transaction) mbxref; // Create a mailbox to receive reference data from the generator
132. event sconext; // Event to signal completion of scoreboard work
134. function new(mailbox #(transaction) mbx, mailbox #(transaction) mbxref);
135. this.mbx = mbx; // Initialize the mailbox for receiving data from the driver
136. this.mbxref = mbxref; // Initialize the mailbox for receiving reference data from the generator
137. endfunction
139. task run();
140. forever begin
141. mbx.get(tr); // Get a transaction from the driver
142. mbxref.get(trref); // Get a reference transaction from the generator
143. tr.display("SCO"); // Display the driver's transaction information
144. trref.display("REF"); // Display the reference transaction information
145. if (tr.dout == trref.din)
146. $display("[SCO] : DATA MATCHED"); // Compare data and display the result
147. else
148. $display("[SCO] : DATA MISMATCHED");
149. $display("-------------------------------------------------");
150. ->sconext; // Signal completion of scoreboard work
151. end
152. endtask
154. endclass
156. ////////////////////////////////////////////////////////
158. class environment;
159. generator gen; // Generator instance
160. driver drv; // Driver instance
161. monitor mon; // Monitor instance
162. scoreboard sco; // Scoreboard instance
163. event next; // Event to signal communication between generator and scoreboard
165. mailbox #(transaction) gdmbx; // Mailbox for communication between generator and driver
166. mailbox #(transaction) msmbx; // Mailbox for communication between monitor and scoreboard
167. mailbox #(transaction) mbxref; // Mailbox for communication between generator and scoreboard
169. virtual dff\_if vif; // Virtual interface for DUT
171. function new(virtual dff\_if vif);
172. gdmbx = new(); // Create a mailbox for generator-driver communication
173. mbxref = new(); // Create a mailbox for generator-scoreboard reference data
174. gen = new(gdmbx, mbxref); // Initialize the generator
175. drv = new(gdmbx); // Initialize the driver
176. msmbx = new(); // Create a mailbox for monitor-scoreboard communication
177. mon = new(msmbx); // Initialize the monitor
178. sco = new(msmbx, mbxref); // Initialize the scoreboard
179. this.vif = vif; // Set the virtual interface for DUT
180. drv.vif = this.vif; // Connect the virtual interface to the driver
181. mon.vif = this.vif; // Connect the virtual interface to the monitor
182. gen.sconext = next; // Set the communication event between generator and scoreboard
183. sco.sconext = next; // Set the communication event between scoreboard and generator
184. endfunction
186. task pre\_test();
187. drv.reset(); // Perform the driver reset
188. endtask
190. task test();
191. fork
192. gen.run(); // Start generator
193. drv.run(); // Start driver
194. mon.run(); // Start monitor
195. sco.run(); // Start scoreboard
196. join\_any
197. endtask
199. task post\_test();
200. wait(gen.done.triggered); // Wait for generator to complete
201. $finish(); // Finish simulation
202. endtask
204. task run();
205. pre\_test(); // Run pre-test setup
206. test(); // Run the test
207. post\_test(); // Run post-test cleanup
208. endtask
209. endclass
211. /////////////////////////////////////////////////////
213. module tb;
214. dff\_if vif(); // Create DUT interface
216. dff dut(vif); // Instantiate DUT
218. initial begin
219. vif.clk <= 0; // Initialize clock signal
220. end
222. always #10 vif.clk <= ~vif.clk; // Toggle the clock every 10 time units
224. environment env; // Create environment instance
226. initial begin
227. env = new(vif); // Initialize the environment with the DUT interface
228. env.gen.count = 30; // Set the generator's stimulus count
229. env.run(); // Run the environment
230. end
232. initial begin
233. $dumpfile("dump.vcd"); // Specify the VCD dump file
234. $dumpvars; // Dump all variables
235. end
236. endmodule

FIFO code:

1. module FIFO(input clk, rst, wr, rd,
2. input [7:0] din, output reg [7:0] dout,
3. output empty, full);
5. // Pointers for write and read operations
6. reg [3:0] wptr = 0, rptr = 0;
8. // Counter for tracking the number of elements in the FIFO
9. reg [4:0] cnt = 0;
11. // Memory array to store data
12. reg [7:0] mem [15:0];
14. always @(posedge clk)
15. begin
16. if (rst == 1'b1)
17. begin
18. // Reset the pointers and counter when the reset signal is asserted
19. wptr <= 0;
20. rptr <= 0;
21. cnt <= 0;
22. end
23. else if (wr && !full)
24. begin
25. // Write data to the FIFO if it's not full
26. mem[wptr] <= din;
27. wptr <= wptr + 1;
28. cnt <= cnt + 1;
29. end
30. else if (rd && !empty)
31. begin
32. // Read data from the FIFO if it's not empty
33. dout <= mem[rptr];
34. rptr <= rptr + 1;
35. cnt <= cnt - 1;
36. end
37. end
39. // Determine if the FIFO is empty or full
40. assign empty = (cnt == 0) ? 1'b1 : 1'b0;
41. assign full = (cnt == 16) ? 1'b1 : 1'b0;
43. endmodule
45. //////////////////////////////////////
47. // Define an interface for the FIFO
48. interface fifo\_if;
50. logic clock, rd, wr; // Clock, read, and write signals
51. logic full, empty; // Flags indicating FIFO status
52. logic [7:0] data\_in; // Data input
53. logic [7:0] data\_out; // Data output
54. logic rst; // Reset signal
56. endinterface

TB:

1. class transaction;
3. rand bit oper; // Randomized bit for operation control (1 or 0)
4. bit rd, wr; // Read and write control bits
5. bit [7:0] data\_in; // 8-bit data input
6. bit full, empty; // Flags for full and empty status
7. bit [7:0] data\_out; // 8-bit data output
9. constraint oper\_ctrl {
10. oper dist {1 :/ 50 , 0 :/ 50}; // Constraint to randomize 'oper' with 50% probability of 1 and 50% probability of 0
11. }
13. endclass
15. ///////////////////////////////////////////////////
17. class generator;
19. transaction tr; // Transaction object to generate and send
20. mailbox #(transaction) mbx; // Mailbox for communication
21. int count = 0; // Number of transactions to generate
22. int i = 0; // Iteration counter
24. event next; // Event to signal when to send the next transaction
25. event done; // Event to convey completion of requested number of transactions
27. function new(mailbox #(transaction) mbx);
28. this.mbx = mbx;
29. tr = new();
30. endfunction;
32. task run();
33. repeat (count) begin
34. assert (tr.randomize) else $error("Randomization failed");
35. i++;
36. mbx.put(tr);
37. $display("[GEN] : Oper : %0d iteration : %0d", tr.oper, i);
38. @(next);
39. end -> done;
40. endtask
42. endclass
43. ////////////////////////////////////////////
45. class driver;
47. virtual fifo\_if fif; // Virtual interface to the FIFO
48. mailbox #(transaction) mbx; // Mailbox for communication
49. transaction datac; // Transaction object for communication
51. function new(mailbox #(transaction) mbx);
52. this.mbx = mbx;
53. endfunction;
55. // Reset the DUT
56. task reset();
57. fif.rst <= 1'b1;
58. fif.rd <= 1'b0;
59. fif.wr <= 1'b0;
60. fif.data\_in <= 0;
61. repeat (5) @(posedge fif.clock);
62. fif.rst <= 1'b0;
63. $display("[DRV] : DUT Reset Done");
64. $display("------------------------------------------");
65. endtask
67. // Write data to the FIFO
68. task write();
69. @(posedge fif.clock);
70. fif.rst <= 1'b0;
71. fif.rd <= 1'b0;
72. fif.wr <= 1'b1;
73. fif.data\_in <= $urandom\_range(1, 10);
74. @(posedge fif.clock);
75. fif.wr <= 1'b0;
76. $display("[DRV] : DATA WRITE data : %0d", fif.data\_in);
77. @(posedge fif.clock);
78. endtask
80. // Read data from the FIFO
81. task read();
82. @(posedge fif.clock);
83. fif.rst <= 1'b0;
84. fif.rd <= 1'b1;
85. fif.wr <= 1'b0;
86. @(posedge fif.clock);
87. fif.rd <= 1'b0;
88. $display("[DRV] : DATA READ");
89. @(posedge fif.clock);
90. endtask
92. // Apply random stimulus to the DUT
93. task run();
94. forever begin
95. mbx.get(datac);
96. if (datac.oper == 1'b1)
97. write();
98. else
99. read();
100. end
101. endtask
103. endclass
105. ///////////////////////////////////////////////////////
107. class monitor;
109. virtual fifo\_if fif; // Virtual interface to the FIFO
110. mailbox #(transaction) mbx; // Mailbox for communication
111. transaction tr; // Transaction object for monitoring
113. function new(mailbox #(transaction) mbx);
114. this.mbx = mbx;
115. endfunction;
117. task run();
118. tr = new();
120. forever begin
121. repeat (2) @(posedge fif.clock);
122. tr.wr = fif.wr;
123. tr.rd = fif.rd;
124. tr.data\_in = fif.data\_in;
125. tr.full = fif.full;
126. tr.empty = fif.empty;
127. @(posedge fif.clock);
128. tr.data\_out = fif.data\_out;
130. mbx.put(tr);
131. $display("[MON] : Wr:%0d rd:%0d din:%0d dout:%0d full:%0d empty:%0d", tr.wr, tr.rd, tr.data\_in, tr.data\_out, tr.full, tr.empty);
132. end
134. endtask
136. endclass
138. /////////////////////////////////////////////////////
140. class scoreboard;
142. mailbox #(transaction) mbx; // Mailbox for communication
143. transaction tr; // Transaction object for monitoring
144. event next;
145. bit [7:0] din[$]; // Array to store written data
146. bit [7:0] temp; // Temporary data storage
147. int err = 0; // Error count
149. function new(mailbox #(transaction) mbx);
150. this.mbx = mbx;
151. endfunction;
153. task run();
154. forever begin
155. mbx.get(tr);
156. $display("[SCO] : Wr:%0d rd:%0d din:%0d dout:%0d full:%0d empty:%0d", tr.wr, tr.rd, tr.data\_in, tr.data\_out, tr.full, tr.empty);
158. if (tr.wr == 1'b1) begin
159. if (tr.full == 1'b0) begin
160. din.push\_front(tr.data\_in);
161. $display("[SCO] : DATA STORED IN QUEUE :%0d", tr.data\_in);
162. end
163. else begin
164. $display("[SCO] : FIFO is full");
165. end
166. $display("--------------------------------------");
167. end
169. if (tr.rd == 1'b1) begin
170. if (tr.empty == 1'b0) begin
171. temp = din.pop\_back();
173. if (tr.data\_out == temp)
174. $display("[SCO] : DATA MATCH");
175. else begin
176. $error("[SCO] : DATA MISMATCH");
177. err++;
178. end
179. end
180. else begin
181. $display("[SCO] : FIFO IS EMPTY");
182. end
184. $display("--------------------------------------");
185. end
187. -> next;
188. end
189. endtask
191. endclass
193. ///////////////////////////////////////////////////////
195. class environment;
197. generator gen;
198. driver drv;
199. monitor mon;
200. scoreboard sco;
201. mailbox #(transaction) gdmbx; // Generator + Driver mailbox
202. mailbox #(transaction) msmbx; // Monitor + Scoreboard mailbox
203. event nextgs;
204. virtual fifo\_if fif;
206. function new(virtual fifo\_if fif);
207. gdmbx = new();
208. gen = new(gdmbx);
209. drv = new(gdmbx);
210. msmbx = new();
211. mon = new(msmbx);
212. sco = new(msmbx);
213. this.fif = fif;
214. drv.fif = this.fif;
215. mon.fif = this.fif;
216. gen.next = nextgs;
217. sco.next = nextgs;
218. endfunction
220. task pre\_test();
221. drv.reset();
222. endtask
224. task test();
225. fork
226. gen.run();
227. drv.run();
228. mon.run();
229. sco.run();
230. join\_any
231. endtask
233. task post\_test();
234. wait(gen.done.triggered);
235. $display("---------------------------------------------");
236. $display("Error Count :%0d", sco.err);
237. $display("---------------------------------------------");
238. $finish();
239. endtask
241. task run();
242. pre\_test();
243. test();
244. post\_test();
245. endtask
247. endclass
249. ///////////////////////////////////////////////////////
251. module tb;
253. fifo\_if fif();
254. FIFO dut (fif.clock, fif.rst, fif.wr, fif.rd, fif.data\_in, fif.data\_out, fif.empty, fif.full);
256. initial begin
257. fif.clock <= 0;
258. end
260. always #10 fif.clock <= ~fif.clock;
262. environment env;
264. initial begin
265. env = new(fif);
266. env.gen.count = 10;
267. env.run();
268. end
270. initial begin
271. $dumpfile("dump.vcd");
272. $dumpvars;
273. end
275. endmodule

SPI

DESIGN CODE:

1. module spi\_master(
2. input clk, newd,rst,
3. input [11:0] din,
4. output reg sclk,cs,mosi
5. );
7. typedef enum bit [1:0] {idle = 2'b00, enable = 2'b01, send = 2'b10, comp = 2'b11 } state\_type;
8. state\_type state = idle;
10. int countc = 0;
11. int count = 0;
13. ////////////////////
14. /////generation of sclk
15. always@(posedge clk)
16. begin
17. if(rst == 1'b1) begin
18. countc <= 0;
19. sclk <= 1'b0;
20. end
21. else begin
22. if(countc < 10 )
23. countc <= countc + 1;
24. else
25. begin
26. countc <= 0;
27. sclk <= ~sclk;
28. end
29. end
30. end
32. //////////////////state machine
33. reg [11:0] temp;

36. always@(posedge sclk)
37. begin
38. if(rst == 1'b1) begin
39. cs <= 1'b1;
40. mosi <= 1'b0;
41. end
42. else begin
43. case(state)
44. idle:
45. begin
46. if(newd == 1'b1) begin
47. state <= send;
48. temp <= din;
49. cs <= 1'b0;
50. end
51. else begin
52. state <= idle;
53. temp <= 8'h00;
54. end
55. end

58. send : begin
59. if(count <= 11) begin
60. mosi <= temp[count]; /////sending lsb first
61. count <= count + 1;
62. end
63. else
64. begin
65. count <= 0;
66. state <= idle;
67. cs <= 1'b1;
68. mosi <= 1'b0;
69. end
70. end

73. default : state <= idle;
75. endcase
76. end
77. end
79. endmodule
80. ///////////////////////////
82. module spi\_slave (
83. input sclk, cs, mosi,
84. output [11:0] dout,
85. output reg done
86. );
88. typedef enum bit {detect\_start = 1'b0, read\_data = 1'b1} state\_type;
89. state\_type state = detect\_start;
91. reg [11:0] temp = 12'h000;
92. int count = 0;
94. always@(posedge sclk)
95. begin
97. case(state)
98. detect\_start:
99. begin
100. done <= 1'b0;
101. if(cs == 1'b0)
102. state <= read\_data;
103. else
104. state <= detect\_start;
105. end
107. read\_data : begin
108. if(count <= 11)
109. begin
110. count <= count + 1;
111. temp <= { mosi, temp[11:1]};
112. end
113. else
114. begin
115. count <= 0;
116. done <= 1'b1;
117. state <= detect\_start;
118. end
120. end
122. endcase
123. end
124. assign dout = temp;
126. endmodule


130. ///////////////////////////////
131. module top (
132. input clk, rst, newd,
133. input [11:0] din,
134. output [11:0] dout,
135. output done
136. );
138. wire sclk, cs, mosi;
140. spi\_master m1 (clk, newd, rst, din, sclk, cs, mosi);
141. spi\_slave s1 (sclk, cs, mosi, dout, done);

144. endmodule

VERILOG TB:

1. `timescale 1ns / 1ps
2. //////////////////////////////////////////////////////////////////////////////////
4. module tb( );
6. reg clk = 0, rst = 0, newd = 0;
7. reg [11:0] din = 0;
8. wire [11:0] dout;
9. wire done;
11. always #10 clk = ~clk;
13. top dut (clk, rst, newd, din, dout, done);
15. initial begin
16. rst = 1;
17. repeat(5) @(posedge clk);
18. rst = 0;
20. for(int i = 0; i < 10; i++)
21. begin
22. newd = 1;
23. din = $urandom;
24. @(posedge dut.s1.sclk);
25. newd = 0;
26. @(posedge done);
27. end

30. end

33. endmodule

SV

1. ////////////////Transaction Class
2. class transaction;
4. bit newd; // Flag for new transaction
5. rand bit [11:0] din; // Random 12-bit data input
6. bit [11:0] dout; // 12-bit data output
8. function transaction copy();
9. copy = new(); // Create a copy of the transaction
10. copy.newd = this.newd; // Copy the newd flag
11. copy.din = this.din; // Copy the data input
12. copy.dout = this.dout; // Copy the data output
13. endfunction
15. endclass
17. ////////////////Generator Class
18. class generator;
20. transaction tr; // Transaction object
21. mailbox #(transaction) mbx; // Mailbox for transactions
22. event done; // Done event
23. int count = 0; // Transaction count
24. event drvnext; // Event to synchronize with driver
25. event sconext; // Event to synchronize with scoreboard
27. function new(mailbox #(transaction) mbx);
28. this.mbx = mbx; // Initialize mailbox
29. tr = new(); // Create a new transaction
30. endfunction
32. task run();
33. repeat(count) begin
34. assert(tr.randomize) else $error("[GEN] :Randomization Failed");
35. mbx.put(tr.copy); // Put a copy of the transaction in the mailbox
36. $display("[GEN] : din : %0d", tr.din);
37. @(sconext); // Wait for the scoreboard synchronization event
38. end
39. -> done; // Signal when done
40. endtask
42. endclass
44. ////////////////Driver Class
45. class driver;
47. virtual spi\_if vif; // Virtual interface
48. transaction tr; // Transaction object
49. mailbox #(transaction) mbx; // Mailbox for transactions
50. mailbox #(bit [11:0]) mbxds; // Mailbox for data output to monitor
51. event drvnext; // Event to synchronize with generator
53. bit [11:0] din; // Data input
55. function new(mailbox #(bit [11:0]) mbxds, mailbox #(transaction) mbx);
56. this.mbx = mbx; // Initialize mailboxes
57. this.mbxds = mbxds;
58. endfunction
60. task reset();
61. vif.rst <= 1'b1; // Set reset signal
62. vif.newd <= 1'b0; // Clear new data flag
63. vif.din <= 1'b0; // Clear data input
64. repeat(10) @(posedge vif.clk);
65. vif.rst <= 1'b0; // Clear reset signal
66. repeat(5) @(posedge vif.clk);
68. $display("[DRV] : RESET DONE");
69. $display("-----------------------------------------");
70. endtask
72. task run();
73. forever begin
74. mbx.get(tr); // Get a transaction from the mailbox
75. vif.newd <= 1'b1; // Set new data flag
76. vif.din <= tr.din; // Set data input
77. mbxds.put(tr.din); // Put data in the mailbox for the monitor
78. @(posedge vif.sclk);
79. vif.newd <= 1'b0; // Clear new data flag
80. @(posedge vif.done);
81. $display("[DRV] : DATA SENT TO DAC : %0d",tr.din);
82. @(posedge vif.sclk);
83. end
85. endtask
87. endclass
89. ////////////////Monitor Class
90. class monitor;
91. transaction tr; // Transaction object
92. mailbox #(bit [11:0]) mbx; // Mailbox for data output
94. virtual spi\_if vif; // Virtual interface
96. function new(mailbox #(bit [11:0]) mbx);
97. this.mbx = mbx; // Initialize the mailbox
98. endfunction
100. task run();
101. tr = new(); // Create a new transaction
102. forever begin
103. @(posedge vif.sclk);
104. @(posedge vif.done);
105. tr.dout = vif.dout; // Record data output
106. @(posedge vif.sclk);
107. $display("[MON] : DATA SENT : %0d", tr.dout);
108. mbx.put(tr.dout); // Put data in the mailbox
109. end
111. endtask
113. endclass
115. ////////////////Scoreboard Class
116. class scoreboard;
117. mailbox #(bit [11:0]) mbxds, mbxms; // Mailboxes for data from driver and monitor
118. bit [11:0] ds; // Data from driver
119. bit [11:0] ms; // Data from monitor
120. event sconext; // Event to synchronize with environment
122. function new(mailbox #(bit [11:0]) mbxds, mailbox #(bit [11:0]) mbxms);
123. this.mbxds = mbxds; // Initialize mailboxes
124. this.mbxms = mbxms;
125. endfunction
127. task run();
128. forever begin
129. mbxds.get(ds); // Get data from driver
130. mbxms.get(ms); // Get data from monitor
131. $display("[SCO] : DRV : %0d MON : %0d", ds, ms);
133. if(ds == ms)
134. $display("[SCO] : DATA MATCHED");
135. else
136. $display("[SCO] : DATA MISMATCHED");
138. $display("-----------------------------------------");
139. ->sconext; // Synchronize with the environment
140. end
142. endtask
144. endclass
146. ////////////////Environment Class
147. class environment;
148. generator gen; // Generator object
149. driver drv; // Driver object
150. monitor mon; // Monitor object
151. scoreboard sco; // Scoreboard object
153. event nextgd; // Event for generator to driver communication
154. event nextgs; // Event for generator to scoreboard communication
156. mailbox #(transaction) mbxgd; // Mailbox for generator to driver communication
157. mailbox #(bit [11:0]) mbxds; // Mailbox for driver to monitor communication
158. mailbox #(bit [11:0]) mbxms; // Mailbox for monitor to scoreboard communication
160. virtual spi\_if vif; // Virtual interface
162. function new(virtual spi\_if vif);
164. mbxgd = new(); // Initialize mailboxes
165. mbxms = new();
166. mbxds = new();
167. gen = new(mbxgd); // Initialize generator
168. drv = new(mbxds,mbxgd); // Initialize driver
169. mon = new(mbxms); // Initialize monitor
170. sco = new(mbxds, mbxms); // Initialize scoreboard
172. this.vif = vif;
173. drv.vif = this.vif;
174. mon.vif = this.vif;
176. gen.sconext = nextgs; // Set synchronization events
177. sco.sconext = nextgs;
179. gen.drvnext = nextgd;
180. drv.drvnext = nextgd;
181. endfunction
183. task pre\_test();
184. drv.reset(); // Perform driver reset
185. endtask
187. task test();
188. fork
189. gen.run(); // Run generator
190. drv.run(); // Run driver
191. mon.run(); // Run monitor
192. sco.run(); // Run scoreboard
193. join\_any
194. endtask
196. task post\_test();
197. wait(gen.done.triggered); // Wait for generator to finish
198. $finish();
199. endtask
201. task run();
202. pre\_test();
203. test();
204. post\_test();
205. endtask
206. endclass
208. ////////////////Testbench Top
209. module tb;
210. spi\_if vif(); // Virtual interface instance
212. top dut(vif.clk,vif.rst,vif.newd,vif.din,vif.dout,vif.done);
214. initial begin
215. vif.clk <= 0;
216. end
218. always #10 vif.clk <= ~vif.clk;
220. environment env;
222. assign vif.sclk = dut.m1.sclk;
224. initial begin
225. env = new(vif);
226. env.gen.count = 4;
227. env.run();
228. end
230. initial begin
231. $dumpfile("dump.vcd");
232. $dumpvars;
233. end
234. endmodule

UART

1. `timescale 1ns / 1ps
3. module uart\_top
4. #(
5. parameter clk\_freq = 1000000,
6. parameter baud\_rate = 9600
7. )
8. (
9. input clk,rst,
10. input rx,
11. input [7:0] dintx,
12. input newd,
13. output tx,
14. output [7:0] doutrx,
15. output donetx,
16. output donerx
17. );
19. uarttx
20. #(clk\_freq, baud\_rate)
21. utx
22. (clk, rst, newd, dintx, tx, donetx);
24. uartrx
25. #(clk\_freq, baud\_rate)
26. rtx
27. (clk, rst, rx, donerx, doutrx);

30. endmodule

33. //////////////////////////////////////////////////////////////////
35. module uarttx
36. #(
37. parameter clk\_freq = 1000000,
38. parameter baud\_rate = 9600
39. )
40. (
41. input clk,rst,
42. input newd,
43. input [7:0] tx\_data,
44. output reg tx,
45. output reg donetx
46. );
48. localparam clkcount = (clk\_freq/baud\_rate); ///x
50. integer count = 0;
51. integer counts = 0;
53. reg uclk = 0;
55. enum bit[1:0] {idle = 2'b00, start = 2'b01, transfer = 2'b10, done = 2'b11} state;
57. ///////////uart\_clock\_gen
58. always@(posedge clk)
59. begin
60. if(count < clkcount/2)
61. count <= count + 1;
62. else begin
63. count <= 0;
64. uclk <= ~uclk;
65. end
66. end

69. reg [7:0] din;
70. ////////////////////Reset decoder

73. always@(posedge uclk)
74. begin
75. if(rst)
76. begin
77. state <= idle;
78. end
79. else
80. begin
81. case(state)
82. idle:
83. begin
84. counts <= 0;
85. tx <= 1'b1;
86. donetx <= 1'b0;
88. if(newd)
89. begin
90. state <= transfer;
91. din <= tx\_data;
92. tx <= 1'b0;
93. end
94. else
95. state <= idle;
96. end


100. transfer: begin
101. if(counts <= 7) begin
102. counts <= counts + 1;
103. tx <= din[counts];
104. state <= transfer;
105. end
106. else
107. begin
108. counts <= 0;
109. tx <= 1'b1;
110. state <= idle;
111. donetx <= 1'b1;
112. end
113. end



118. default : state <= idle;
119. endcase
120. end
121. end
123. endmodule


127. ////////////////////////////////////////////////////////////////////



132. module uartrx
133. #(
134. parameter clk\_freq = 1000000, //MHz
135. parameter baud\_rate = 9600
136. )
137. (
138. input clk,
139. input rst,
140. input rx,
141. output reg done,
142. output reg [7:0] rxdata
143. );
145. localparam clkcount = (clk\_freq/baud\_rate);
147. integer count = 0;
148. integer counts = 0;
150. reg uclk = 0;

153. enum bit[1:0] {idle = 2'b00, start = 2'b01} state;
155. ///////////uart\_clock\_gen
156. always@(posedge clk)
157. begin
158. if(count < clkcount/2)
159. count <= count + 1;
160. else begin
161. count <= 0;
162. uclk <= ~uclk;
163. end
164. end


168. always@(posedge uclk)
169. begin
170. if(rst)
171. begin
172. rxdata <= 8'h00;
173. counts <= 0;
174. done <= 1'b0;
175. end
176. else
177. begin
178. case(state)
180. idle :
181. begin
182. rxdata <= 8'h00;
183. counts <= 0;
184. done <= 1'b0;
186. if(rx == 1'b0)
187. state <= start;
188. else
189. state <= idle;
190. end
192. start:
193. begin
194. if(counts <= 7)
195. begin
196. counts <= counts + 1;
197. rxdata <= {rx, rxdata[7:1]};
198. end
199. else
200. begin
201. counts <= 0;
202. done <= 1'b1;
203. state <= idle;
204. end
205. end

208. default : state <= idle;
210. endcase
212. end
214. end
216. endmodule

219. ///////////////////////////////////////////////////////////////////
221. interface uart\_if;
222. logic clk;
223. logic uclktx;
224. logic uclkrx;
225. logic rst;
226. logic rx;
227. logic [7:0] dintx;
228. logic newd;
229. logic tx;
230. logic [7:0] doutrx;
231. logic donetx;
232. logic donerx;
234. endinterface

TB CODE:

1. class transaction;
3. typedef enum bit {write = 1'b0 , read = 1'b1} oper\_type;
5. randc oper\_type oper;
7. bit rx;
9. rand bit [7:0] dintx;
11. bit newd;
12. bit tx;
14. bit [7:0] doutrx;
15. bit donetx;
16. bit donerx;


20. function transaction copy();
21. copy = new();
22. copy.rx = this.rx;
23. copy.dintx = this.dintx;
24. copy.newd = this.newd;
25. copy.tx = this.tx;
26. copy.doutrx = this.doutrx;
27. copy.donetx = this.donetx;
28. copy.donerx = this.donerx;
29. copy.oper = this.oper;
30. endfunction
32. endclass

35. /\*
36. module tb;
37. transaction tr;

40. initial begin
41. tr = new();
42. tr.display("TOP");
44. end

47. endmodule
49. \*/

52. class generator;
54. transaction tr;
56. mailbox #(transaction) mbx;
58. event done;
60. int count = 0;
62. event drvnext;
63. event sconext;

66. function new(mailbox #(transaction) mbx);
67. this.mbx = mbx;
68. tr = new();
69. endfunction

72. task run();
74. repeat(count) begin
75. assert(tr.randomize) else $error("[GEN] :Randomization Failed");
76. mbx.put(tr.copy);
77. $display("[GEN]: Oper : %0s Din : %0d",tr.oper.name(), tr.dintx);
78. @(drvnext);
79. @(sconext);
80. end
82. -> done;
83. endtask

86. endclass
88. /////////////////////////////////////////////////////////
90. class driver;

93. virtual uart\_if vif;
95. transaction tr;
97. mailbox #(transaction) mbx;
99. mailbox #(bit [7:0]) mbxds;

102. event drvnext;
104. bit [7:0] din;

107. bit wr = 0; ///random operation read / write
108. bit [7:0] datarx; ///data rcvd during read




114. function new(mailbox #(bit [7:0]) mbxds, mailbox #(transaction) mbx);
115. this.mbx = mbx;
116. this.mbxds = mbxds;
117. endfunction


121. task reset();
122. vif.rst <= 1'b1;
123. vif.dintx <= 0;
124. vif.newd <= 0;
125. vif.rx <= 1'b1;
127. repeat(5) @(posedge vif.uclktx);
128. vif.rst <= 1'b0;
129. @(posedge vif.uclktx);
130. $display("[DRV] : RESET DONE");
131. $display("----------------------------------------");
132. endtask


136. task run();
138. forever begin
139. mbx.get(tr);
141. if(tr.oper == 1'b0) ////data transmission
142. begin
143. //
144. @(posedge vif.uclktx);
145. vif.rst <= 1'b0;
146. vif.newd <= 1'b1; ///start data sending op
147. vif.rx <= 1'b1;
148. vif.dintx = tr.dintx;
149. @(posedge vif.uclktx);
150. vif.newd <= 1'b0;
151. ////wait for completion
152. //repeat(9) @(posedge vif.uclktx);
153. mbxds.put(tr.dintx);
154. $display("[DRV]: Data Sent : %0d", tr.dintx);
155. wait(vif.donetx == 1'b1);
156. ->drvnext;
157. end
159. else if (tr.oper == 1'b1)
160. begin
162. @(posedge vif.uclkrx);
163. vif.rst <= 1'b0;
164. vif.rx <= 1'b0;
165. vif.newd <= 1'b0;
166. @(posedge vif.uclkrx);
168. for(int i=0; i<=7; i++)
169. begin
170. @(posedge vif.uclkrx);
171. vif.rx <= $urandom;
172. datarx[i] = vif.rx;
173. end

176. mbxds.put(datarx);
178. $display("[DRV]: Data RCVD : %0d", datarx);
179. wait(vif.donerx == 1'b1);
180. vif.rx <= 1'b1;
181. ->drvnext;

184. end


188. end
190. endtask
192. endclass
194. /\*
196. module tb;
197. generator gen;
198. driver drv;
199. event next;
200. event done;
201. mailbox #(transaction) mbx;
202. mailbox #(bit [7:0]) mbxt;
204. uart\_if vif();
206. uart\_top #(1000000, 9600) dut (vif.clk,vif.rst,vif.rx,vif.dintx,vif.send,vif.tx,vif.doutrx,vif.donetx, vif.donerx);


210. initial begin
211. vif.clk <= 0;
212. end
214. always #10 vif.clk <= ~vif.clk;


218. initial begin
219. mbx = new();
220. mbxt = new();
221. gen = new(mbx);
222. drv = new(mbxt,mbx);
223. gen.count = 20;
224. drv.vif = vif;
226. gen.drvnext = next;
228. drv.drvnext = next;
229. end
231. initial begin
233. fork
234. gen.run();
235. drv.run();
236. join\_none
237. wait(gen.done.triggered);
238. $finish();
239. end
241. initial begin
242. $dumpfile("dump.vcd");
243. $dumpvars;
244. end


248. assign vif.uclktx = dut.utx.uclk;
249. assign vif.uclkrx = dut.rtx.uclk;


253. endmodule
255. \*/
257. class monitor;
259. transaction tr;
261. mailbox #(bit [7:0]) mbx;
263. bit [7:0] srx; //////send
264. bit [7:0] rrx; ///// recv


268. virtual uart\_if vif;

271. function new(mailbox #(bit [7:0]) mbx);
272. this.mbx = mbx;
273. endfunction
275. task run();
277. forever begin
279. @(posedge vif.uclktx);
280. if ( (vif.newd== 1'b1) && (vif.rx == 1'b1) )
281. begin
283. @(posedge vif.uclktx); ////start collecting tx data from next clock tick
285. for(int i = 0; i<= 7; i++)
286. begin
287. @(posedge vif.uclktx);
288. srx[i] = vif.tx;
290. end

293. $display("[MON] : DATA SEND on UART TX %0d", srx);
295. //////////wait for done tx before proceeding next transaction
296. @(posedge vif.uclktx); //
297. mbx.put(srx);
299. end
301. else if ((vif.rx == 1'b0) && (vif.newd == 1'b0) )
302. begin
303. wait(vif.donerx == 1);
304. rrx = vif.doutrx;
305. $display("[MON] : DATA RCVD RX %0d", rrx);
306. @(posedge vif.uclktx);
307. mbx.put(rrx);
308. end
309. end
310. endtask

313. endclass
315. ////////////////////////////////////////////////////////
317. /\*
318. module tb;
319. generator gen;
320. driver drv;
321. monitor mon;
323. event sconext;
324. event drvnext;
326. event done;
328. mailbox #(transaction) mbx;
329. mailbox #(bit [7:0]) mbxds;
330. mailbox #(bit [7:0]) mbxms;
332. uart\_if vif();
334. uart\_top #(1000000, 9600) dut (vif.clk,vif.rst,vif.rx,vif.dintx,vif.send,vif.tx,vif.doutrx,vif.donetx, vif.donerx);


338. initial begin
339. vif.clk <= 0;
340. end
342. always #10 vif.clk <= ~vif.clk;


346. initial begin
347. mbx = new();
348. mbxds = new();
349. mbxms = new();

352. gen = new(mbx);
353. drv = new(mbxds,mbx);
354. mon= new(mbxms);
356. gen.count = 10;
357. drv.vif = vif;
358. mon.vif = vif;
360. gen.drvnext = drvnext;
362. drv.drvnext = drvnext;
364. gen.sconext = sconext;
366. mon.sconext = sconext;
367. end
369. initial begin
371. fork
372. gen.run();
373. drv.run();
374. mon.run();
375. join\_none
376. wait(gen.done.triggered);
377. $finish();
378. end
380. initial begin
381. $dumpfile("dump.vcd");
382. $dumpvars;
383. end


387. assign vif.uclktx = dut.utx.uclk;
388. assign vif.uclkrx = dut.rtx.uclk;


392. endmodule
393. \*/
395. /////////////////////////////////////////////////////////////////////////


399. class scoreboard;
400. mailbox #(bit [7:0]) mbxds, mbxms;
402. bit [7:0] ds;
403. bit [7:0] ms;
405. event sconext;
407. function new(mailbox #(bit [7:0]) mbxds, mailbox #(bit [7:0]) mbxms);
408. this.mbxds = mbxds;
409. this.mbxms = mbxms;
410. endfunction
412. task run();
413. forever begin
415. mbxds.get(ds);
416. mbxms.get(ms);
418. $display("[SCO] : DRV : %0d MON : %0d", ds, ms);
419. if(ds == ms)
420. $display("DATA MATCHED");
421. else
422. $display("DATA MISMATCHED");
424. $display("----------------------------------------");
426. ->sconext;
427. end
428. endtask

431. endclass
433. ///////////////////////////////
435. class environment;
437. generator gen;
438. driver drv;
439. monitor mon;
440. scoreboard sco;


444. event nextgd; ///gen -> drv
446. event nextgs; /// gen -> sco
448. mailbox #(transaction) mbxgd; ///gen - drv
450. mailbox #(bit [7:0]) mbxds; /// drv - sco

453. mailbox #(bit [7:0]) mbxms; /// mon - sco
455. virtual uart\_if vif;

458. function new(virtual uart\_if vif);
460. mbxgd = new();
461. mbxms = new();
462. mbxds = new();
464. gen = new(mbxgd);
465. drv = new(mbxds,mbxgd);


469. mon = new(mbxms);
470. sco = new(mbxds, mbxms);
472. this.vif = vif;
473. drv.vif = this.vif;
474. mon.vif = this.vif;
476. gen.sconext = nextgs;
477. sco.sconext = nextgs;
479. gen.drvnext = nextgd;
480. drv.drvnext = nextgd;
482. endfunction
484. task pre\_test();
485. drv.reset();
486. endtask
488. task test();
489. fork
490. gen.run();
491. drv.run();
492. mon.run();
493. sco.run();
494. join\_any
495. endtask
497. task post\_test();
498. wait(gen.done.triggered);
499. $finish();
500. endtask
502. task run();
503. pre\_test();
504. test();
505. post\_test();
506. endtask


510. endclass
512. ///////////////////////////////////////////

515. module tb;
517. uart\_if vif();
519. uart\_top #(1000000, 9600) dut (vif.clk,vif.rst,vif.rx,vif.dintx,vif.newd,vif.tx,vif.doutrx,vif.donetx, vif.donerx);


523. initial begin
524. vif.clk <= 0;
525. end
527. always #10 vif.clk <= ~vif.clk;
529. environment env;


533. initial begin
534. env = new(vif);
535. env.gen.count = 5;
536. env.run();
537. end

540. initial begin
541. $dumpfile("dump.vcd");
542. $dumpvars;
543. end
545. assign vif.uclktx = dut.utx.uclk;
546. assign vif.uclkrx = dut.rtx.uclk;
548. endmodule


552. ////////////////////////////////////////

I2C:

1. module eeprom\_top
2. (
3. input clk,
4. input rst,
5. input newd,
6. input ack,
7. input wr,
8. output scl,
9. inout sda,
10. input [7:0] wdata,
11. input [6:0] addr, /////8-bit 7-bit : addr 1-bit : mode
12. output reg [7:0] rdata,
13. output reg done
14. );
16. reg sda\_en = 0;
18. reg sclt, sdat, donet;
19. reg [7:0] rdatat;
20. reg [7:0] addrt;

23. typedef enum {new\_data = 0, check\_wr = 1, wstart = 2, wsend\_addr = 3, waddr\_ack = 4,
24. wsend\_data = 5, wdata\_ack = 6, wstop = 7, rsend\_addr = 8, raddr\_ack = 9, rsend\_data = 10,
25. rdata\_ack = 11, rstop = 12 } state\_type;
26. state\_type state;




32. reg sclk\_ref = 0;
33. int count = 0;
34. int i = 0;
36. ////100 M / 400 K = N
37. ///N/2

40. always@(posedge clk)
41. begin
42. if(count <= 9)
43. begin
44. count <= count + 1;
45. end
46. else
47. begin
48. count <= 0;
49. sclk\_ref <= ~sclk\_ref;
50. end
51. end


55. always@(posedge sclk\_ref, posedge rst)
56. begin
57. if(rst == 1'b1)
58. begin
59. sclt <= 1'b0;
60. sdat <= 1'b0;
61. donet <= 1'b0;
62. end
63. else begin
64. case(state)
65. new\_data :
66. begin
67. sdat <= 1'b0;
68. done <= 1'b0;
69. sda\_en <= 1'b1;
70. sclt <= 1'b1;
71. sdat <= 1'b1;
72. if(newd == 1'b1)
73. state <= wstart;
74. else
75. state <= new\_data;
76. end
78. wstart:
79. begin
80. sdat <= 1'b0;
81. sclt <= 1'b1;
82. state <= check\_wr;
83. addrt <= {addr,wr};
84. end


88. check\_wr: begin
89. ///addr remain same for both write and read
90. if(wr == 1'b1)
91. begin
92. state <= wsend\_addr;
93. sdat <= addrt[0];
94. i <= 1;
95. end
96. else
97. begin
98. state <= rsend\_addr;
99. sdat <= addrt[0];
100. i <= 1;
101. end
102. end





109. wsend\_addr : begin
110. if(i <= 7) begin
111. sdat <= addrt[i];
112. i <= i + 1;
113. end
114. else
115. begin
116. i <= 0;
117. state <= waddr\_ack;
118. end
119. end

122. waddr\_ack : begin
123. if(ack == 1'b1) begin
124. state <= wsend\_data;
125. sdat <= wdata[0];
126. i <= i + 1;
127. end
128. else
129. state <= waddr\_ack;
130. end
132. wsend\_data : begin
133. if(i <= 7) begin
134. i <= i + 1;
135. sdat <= wdata[i];
136. end
137. else begin
138. i <= 0;
139. state <= wdata\_ack;
140. end
141. end
143. wdata\_ack : begin
144. if(ack == 1'b1) begin
145. state <= wstop;
146. sdat <= 1'b0;
147. sclt <= 1'b1;
148. end
149. else begin
150. state <= wdata\_ack;
151. end
152. end


156. wstop: begin
157. sdat <= 1'b1;
158. state <= new\_data;
159. done <= 1'b1;
160. end
162. ///////////////////////read state

165. rsend\_addr : begin
166. if(i <= 7) begin
167. sdat <= addrt[i];
168. i <= i + 1;
169. end
170. else
171. begin
172. i <= 0;
173. state <= raddr\_ack;
174. end
175. end

178. raddr\_ack : begin
179. if(ack == 1'b1) begin
180. state <= rsend\_data;
181. sda\_en <= 1'b0;
182. end
183. else
184. state <= raddr\_ack;
185. end
187. rsend\_data : begin
188. if(i <= 7) begin
189. i <= i + 1;
190. state <= rsend\_data;
191. rdata[i] <= sda;
192. end
193. else
194. begin
195. i <= 0;
196. state <= rstop;
197. sclt <= 1'b1;
198. sdat <= 1'b0;
199. end
200. end



205. rstop: begin
206. sdat <= 1'b1;
207. state <= new\_data;
208. done <= 1'b1;
209. end

212. default : state <= new\_data;
214. endcase
215. end
216. end
218. assign scl = (( state == wstart) || ( state == wstop) || ( state == rstop)) ? sclt : sclk\_ref;
219. assign sda = (sda\_en == 1'b1) ? sdat : 1'bz;
220. endmodule


224. /////////////////////////////module i2c\_memory
226. module i2cmem\_top (
227. input clk, rst,
228. input scl,
229. inout sda,
230. output reg ack
231. );
233. reg [7:0] mem[128];
234. reg [7:0] addrin;
235. reg [7:0] datain;
236. reg [7:0] temprd;
237. reg sda\_en = 0;
238. reg sdar = 0;
240. int i = 0;
241. int count = 0;
243. reg sclk\_ref = 0;
245. always@(posedge clk)
246. begin
247. if(count <= 9)
248. begin
249. count <= count + 1;
250. end
251. else
252. begin
253. count <= 0;
254. sclk\_ref <= ~sclk\_ref;
255. end
256. end


260. typedef enum {start = 0, store\_addr = 1, ack\_addr = 2, store\_data = 3, ack\_data = 4, stop = 5, send\_data = 6} state\_type;
261. state\_type state;

264. always@(posedge sclk\_ref, posedge rst)
265. begin
266. if(rst == 1'b1) begin
267. for(int j =0; j < 127 ; j++) begin
268. mem[j] <= 8'h91;
269. end
270. sda\_en <= 1'b1;
271. end
272. else begin
273. case(state)
274. start: begin
275. sda\_en <= 1'b1; ///read data
276. if ((scl == 1'b1) && (sda == 1'b0)) begin
277. state <= store\_addr;
278. end
279. else
280. state <= start;
281. end
283. store\_addr: begin
284. sda\_en <= 1'b1; ///read data
285. if(i <= 7) begin
286. i <= i + 1;
287. addrin[i] <= sda;
288. end
289. else begin
290. state <= ack\_addr;
291. temprd <= mem[addrin[7:1]];
292. ack <= 1'b1;
293. i <= 0;
294. end
295. end
297. ack\_addr: begin
298. ack <= 1'b0;
300. if(addrin[0] == 1'b1) begin
301. state <= store\_data;
302. sda\_en <= 1'b1;
303. end
304. else begin
305. state <= send\_data;
306. i <= 1;
307. sda\_en <= 1'b0; ///write data
308. sdar <= temprd[0];
309. end
310. end
312. store\_data :
313. begin
315. if(i <= 7) begin
316. i <= i + 1;
317. datain[i] <= sda;
318. end
319. else begin
320. state <= ack\_data;
321. ack <= 1'b1;
322. i <= 0;
323. end
324. end
326. ack\_data : begin
327. ack <= 1'b0;
328. mem[addrin[7:1]] <= datain;
329. state <= stop;
330. end
332. stop: begin
333. sda\_en <= 1'b1;
334. if( (scl == 1'b1)&&(sda == 1'b1) )
335. state <= start;
336. else
337. state <= stop;
338. end
340. send\_data : begin
341. sda\_en <= 1'b0;
342. if(i <= 7) begin
343. i <= i + 1;
344. sdar <= temprd[i];
345. end
346. else begin
347. state <= stop;
348. i <= 0;
349. sda\_en <= 1'b1;
350. end
351. end
353. default : state <= start;
354. endcase
355. end
356. end
358. assign sda = (sda\_en == 1'b1) ? 1'bz : sdar;
360. endmodule


364. ///////////////////////////////////////
366. module i2c\_top(
367. input clk,
368. input rst,
369. input newd,
370. input wr,
371. input [7:0] wdata,
372. input [6:0] addr,
373. output [7:0] rdata,
374. output done
375. );
377. wire sdac;
378. wire sclc;
379. wire ackc;
381. eeprom\_top e1 (clk,rst,newd, ackc,wr,sclc, sdac, wdata, addr, rdata,done);
383. i2cmem\_top m1 (clk,rst, sclc, sdac, ackc);
385. endmodule
387. //////////////////////////////////////////////////

390. interface i2c\_if;
391. logic clk;
392. logic rst;
393. logic newd;
394. logic wr;
395. logic [7:0] wdata;
396. logic [6:0] addr;
397. logic [7:0] rdata;
398. logic done;
399. logic sclk\_ref;

402. endinterface

**Testbench Code:**

1. /////////////////////////Transaction
2. class transaction;
4. bit newd;
5. rand bit wr;
6. rand bit [7:0] wdata;
7. rand bit [6:0] addr;
8. bit [7:0] rdata;
9. bit done;
11. constraint addr\_c { addr > 0; addr < 5; }
13. constraint rd\_wr\_c {
14. wr dist {1 :/ 50 , 0 :/ 50};
15. }
17. function void display( input string tag);
18. $display("[%0s] : WR : %0b WDATA : %0d ADDR : %0d RDATA : %0d DONE : %0b ",tag, wr, wdata, addr, rdata, done);
19. endfunction
21. function transaction copy();
22. copy = new();
23. copy.newd = this.newd;
24. copy.wr = this.wr;
25. copy.wdata = this.wdata;
26. copy.addr = this.addr;
27. copy.rdata = this.rdata;
28. copy.done = this.done;
29. endfunction
31. endclass
33. /////////////////////////Generator
35. class generator;
37. transaction tr;
38. mailbox #(transaction) mbxgd;
39. event done; ///gen completed sending requested no. of transaction
40. event drvnext; /// dr complete its wor;
41. event sconext; ///scoreboard complete its work
43. int count = 0;
45. function new( mailbox #(transaction) mbxgd);
46. this.mbxgd = mbxgd;
47. tr =new();
48. endfunction
50. task run();
52. repeat(count) begin
53. assert(tr.randomize) else $error("Randomization Failed");
54. mbxgd.put(tr.copy);
55. tr.display("GEN");
56. @(drvnext);
57. @(sconext);
58. end
59. -> done;
60. endtask

63. endclass

66. ///////////////////////////////Driver

69. class driver;
71. virtual i2c\_if vif;
73. transaction tr;
75. event drvnext;
77. mailbox #(transaction) mbxgd;

80. function new( mailbox #(transaction) mbxgd );
81. this.mbxgd = mbxgd;
82. endfunction
84. //////////////////Resetting System
85. task reset();
86. vif.rst <= 1'b1;
87. vif.newd <= 1'b0;
88. vif.wr <= 1'b0;
89. vif.wdata <= 0;
90. vif.addr <= 0;
91. repeat(10) @(posedge vif.clk);
92. vif.rst <= 1'b0;
93. repeat(5) @(posedge vif.clk);
94. $display("[DRV] : RESET DONE");
95. endtask

98. task run();
100. forever begin
102. mbxgd.get(tr);

105. @(posedge vif.sclk\_ref);
106. vif.rst <= 1'b0;
107. vif.newd <= 1'b1;
108. vif.wr <= tr.wr;
109. vif.wdata <= tr.wdata;
110. vif.addr <= tr.addr;

113. @(posedge vif.sclk\_ref);
114. vif.newd <= 1'b0;

117. wait(vif.done == 1'b1);
118. @(posedge vif.sclk\_ref);
119. wait(vif.done == 1'b0);

122. $display("[DRV] : wr:%0b wdata :%0d waddr : %0d rdata : %0d", vif.wr, vif.wdata, vif.addr, vif.rdata);
124. ->drvnext;
125. end
126. endtask


130. endclass


134. /////////////////////////////////////

137. /\* module tb;
139. generator gen;
140. driver drv;
141. event next;
142. event done;
144. mailbox #(transaction) mbxgd;
146. i2c\_if vif();
147. i2c\_top dut (vif.clk, vif.rst, vif.newd, vif.wr, vif.wdata, vif.addr, vif.rdata, vif.done);
149. initial begin
150. vif.clk <= 0;
151. end
153. always #5 vif.clk <= ~vif.clk;
155. initial begin
157. mbxgd = new();
158. gen = new(mbxgd);
159. drv = new(mbxgd);
160. gen.count = 10;
161. drv.vif = vif;
163. drv.drvnext = next;
164. gen.drvnext = next;
166. end
168. initial begin
169. fork
170. drv.reset();
171. gen.run();
172. drv.run();
173. join\_none
174. wait(gen.done.triggered);
175. $finish();
176. end
178. initial begin
179. $dumpfile("dump.vcd");
180. $dumpvars;
181. end
183. assign vif.sclk\_ref = dut.e1.sclk\_ref;

186. endmodule
188. \*/
190. //////////////////////////////////////monitor

193. class monitor;
195. virtual i2c\_if vif;
197. transaction tr;
199. mailbox #(transaction) mbxms;




205. function new( mailbox #(transaction) mbxms );
206. this.mbxms = mbxms;
207. endfunction

210. task run();
212. tr = new();
214. forever begin
216. @(posedge vif.sclk\_ref);
218. if(vif.newd == 1'b1) begin
220. if(vif.wr == 1'b0)
221. begin
222. tr.wr = vif.wr;
223. tr.wdata = vif.wdata;
224. tr.addr = vif.addr;
225. @(posedge vif.sclk\_ref);
227. wait(vif.done == 1'b1);
228. tr.rdata = vif.rdata;
230. repeat(2) @(posedge vif.sclk\_ref);
232. $display("[MON] : DATA READ -> waddr : %0d rdata : %0d", tr.addr, tr.rdata);
233. end
235. else
237. begin
238. tr.wr = vif.wr;
239. tr.wdata = vif.wdata;
240. tr.addr = vif.addr;
242. @(posedge vif.sclk\_ref);
244. wait(vif.done == 1'b1);
246. tr.rdata = vif.rdata;
248. repeat(2) @(posedge vif.sclk\_ref);
250. $display("[MON] : DATA WRITE -> wdata :%0d waddr : %0d", tr.wdata, tr.addr);
251. end

254. mbxms.put(tr);
256. end
258. end


262. endtask
264. endclass
265. ///////////////////////////////////////////////////////////////
267. class scoreboard;
269. transaction tr;
271. mailbox #(transaction) mbxms;
273. event sconext;
275. bit [7:0] temp;
277. bit [7:0] data[128] = '{default:0};


281. function new( mailbox #(transaction) mbxms );
282. this.mbxms = mbxms;
283. endfunction

286. task run();
288. forever begin
290. mbxms.get(tr);
292. tr.display("SCO");
294. if(tr.wr == 1'b1)
295. begin
297. data[tr.addr] = tr.wdata;
299. $display("[SCO]: DATA STORED -> ADDR : %0d DATA : %0d", tr.addr, tr.wdata);
300. end
301. else
302. begin
303. temp = data[tr.addr];
305. if( (tr.rdata == temp) || (tr.rdata == 145) )
306. $display("[SCO] :DATA READ -> Data Matched");
307. else
308. $display("[SCO] :DATA READ -> DATA MISMATCHED");
309. end

312. ->sconext;
313. end
314. endtask

317. endclass




323. module tb;
325. generator gen;
326. driver drv;
327. monitor mon;
328. scoreboard sco;

331. event nextgd;
332. event nextgs;

335. mailbox #(transaction) mbxgd, mbxms;

338. i2c\_if vif();
340. i2c\_top dut (vif.clk, vif.rst, vif.newd, vif.wr, vif.wdata, vif.addr, vif.rdata, vif.done);
342. initial begin
343. vif.clk <= 0;
344. end
346. always #5 vif.clk <= ~vif.clk;
348. initial begin

351. mbxgd = new();
352. mbxms = new();
354. gen = new(mbxgd);
355. drv = new(mbxgd);
357. mon = new(mbxms);
358. sco = new(mbxms);
360. gen.count = 20;
362. drv.vif = vif;
363. mon.vif = vif;
365. gen.drvnext = nextgd;
366. drv.drvnext = nextgd;
368. gen.sconext = nextgs;
369. sco.sconext = nextgs;
371. end
373. task pre\_test;
374. drv.reset();
375. endtask
377. task test;
378. fork
379. gen.run();
380. drv.run();
381. mon.run();
382. sco.run();
383. join\_any
384. endtask

387. task post\_test;
388. wait(gen.done.triggered);
389. $finish();
390. endtask
392. task run();
393. pre\_test;
394. test;
395. post\_test;
396. endtask
398. initial begin
399. run();
400. end
402. initial begin
403. $dumpfile("dump.vcd");
404. $dumpvars(1,tb);
405. end
407. assign vif.sclk\_ref = dut.e1.sclk\_ref;
409. endmodule

APB\_RAM:

1. module apb\_ram (
2. input presetn,
3. input pclk,
4. input psel,
5. input penable,
6. input pwrite,
7. input [31:0] paddr, pwdata,
8. output reg [31:0] prdata,
9. output reg pready, pslverr
10. );
12. reg [31:0] mem [32];
14. typedef enum {idle = 0, setup = 1, access = 2, transfer = 3} state\_type;
16. state\_type state = idle, next\_state = idle;
18. always@(posedge pclk)
19. begin
20. if(presetn == 1'b0)
21. begin
22. state <= idle;
23. prdata <= 32'h00000000;
24. pready <= 1'b0;
25. pslverr <= 1'b0;
26. for(int i = 0; i < 32; i++) begin
27. mem[i] <= 0;
28. end
29. end
30. else
31. begin
33. case(state)
34. idle :
35. begin
36. prdata <= 32'h00000000;
37. pready <= 1'b0;
38. pslverr <= 1'b0;
40. if((psel == 1'b0) && (penable == 1'b0))
41. begin
42. state <= setup;
43. end
44. end
46. setup: ///start of transaction
47. begin
48. if((psel == 1'b1) && (penable == 1'b0)) begin
49. if(paddr < 32) begin
50. state <= access;
51. pready <= 1'b1;
52. end
53. else
54. begin
55. state <= access;
56. pready <= 1'b0;
57. end
58. end
59. else
60. state <= setup;
61. end
63. access:
64. begin
65. if(psel && pwrite && penable)
66. begin
67. if(paddr < 32)
68. begin
69. mem[paddr] <= pwdata;
70. state <= transfer;
71. pslverr <= 1'b0;
72. end
73. else
74. begin
75. state <= transfer;
76. pready <= 1'b1;
77. pslverr <= 1'b1;
78. end
79. end
80. else if ( psel && !pwrite && penable)
81. begin
82. if(paddr < 32)
83. begin
84. prdata <= mem[paddr];
85. state <= transfer;
86. pready <= 1'b1;
87. pslverr <= 1'b0;
88. end
89. else
90. begin
91. state <= transfer;
92. pready <= 1'b1;
93. pslverr <= 1'b1;
94. prdata <= 32'hxxxxxxxx;
95. end
96. end
97. end
98. transfer: begin
99. state <= setup;
100. pready <= 1'b0;
101. pslverr <= 1'b0;
102. end


106. default : state <= idle;
108. endcase
110. end
111. end

114. endmodule

///////////////////////////////////Interface Code

1. interface abp\_if;
3. logic presetn;
4. logic pclk;
5. logic psel;
6. logic penable;
7. logic pwrite;
8. logic [31:0] paddr, pwdata;
9. logic [31:0] prdata;
10. logic pready, pslverr;

13. endinterface

////////////////////////////Testbench Code

1. class transaction;
3. typedef enum int {write = 0, read = 1, random = 2, error = 3} op\_type;
5. randc op\_type oper ;
6. rand bit [31:0] paddr;
7. rand bit [31:0] pwdata;
8. rand bit psel;
9. rand bit penable;
10. rand bit pwrite;
11. bit [31:0] prdata;
12. bit pready;
13. bit pslverr;


17. constraint addr\_c {
18. paddr > 1; paddr < 5;////2 3 4
19. }
21. constraint data\_c {
22. pwdata > 1; pwdata < 10; /// 2-9
23. }
25. function void display(input string tag);
26. $display("[%0s] : OP:%0s 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);
27. endfunction
29. function transaction copy();
30. copy = new();
31. copy.oper = this.oper;
32. copy.paddr = this.paddr;
33. copy.pwdata = this.pwdata;
34. copy.psel = this.psel;
35. copy.penable = this.penable;
36. copy.pwrite = this.pwrite;
37. copy.prdata = this.prdata;
38. copy.pready = this.pready;
39. copy.pslverr = this.pslverr;
40. endfunction

43. endclass
44. /////////////////////////////////////////
45. /\*
47. module tb;
49. transaction tr;
51. initial begin
52. tr = new();
53. tr.display("TOP");
54. end
56. endmodule
58. \*/









69. /////////////////////////////////////////
70. class generator;
72. transaction tr;
74. mailbox #(transaction) mbx;
76. int count = 0;
78. event nextdrv; ///driver completed task of triggering interface
80. event nextsco; ///scoreboard completed its objective
82. event done;

85. function new(mailbox #(transaction) mbx);
86. this.mbx = mbx;
87. tr=new();
88. endfunction;
90. task run();
92. repeat(count)
93. begin
94. assert(tr.randomize()) else $error("Randomization failed");
95. mbx.put(tr.copy);
96. tr.display("GEN");
97. @(nextdrv);
98. @(nextsco);
99. end
100. ->done;
101. endtask

104. endclass
106. /////////////////////////////////////////////////////
107. /\*
108. module tb;
110. generator gen;
111. mailbox #(transaction) mbx;


115. initial begin
116. mbx = new();
117. gen = new(mbx);
118. gen.count = 20;
119. gen.run();

122. end

125. endmodule
127. \*/



132. ///////////////////////////////////////////////////////
134. class driver;
136. virtual abp\_if vif;
138. mailbox #(transaction) mbx;
140. transaction datac;
142. event nextdrv;
144. function new(mailbox #(transaction) mbx);
145. this.mbx = mbx;
146. endfunction;

149. task reset();
150. vif.presetn <= 1'b0;
151. vif.psel <= 1'b0;
152. vif.penable <= 1'b0;
153. vif.pwdata <= 0;
154. vif.paddr <= 0;
155. vif.pwrite <= 1'b0;
156. repeat(5) @(posedge vif.pclk);
157. vif.presetn <= 1'b1;
158. repeat(5) @(posedge vif.pclk);
159. $display("[DRV] : RESET DONE");
160. endtask
162. task run();
163. forever begin
165. mbx.get(datac);
167. if(datac.oper == 0) ///write
168. begin
169. @(posedge vif.pclk);
170. vif.psel <= 1'b1;
171. vif.penable <= 1'b0;
172. vif.pwdata <= datac.pwdata;
173. vif.paddr <= datac.paddr;
174. vif.pwrite <= 1'b1;
175. @(posedge vif.pclk);
176. vif.penable <= 1'b1;
177. repeat(2) @(posedge vif.pclk);
178. vif.psel <= 1'b0;
179. vif.penable <= 1'b0;
180. vif.pwrite <= 1'b0;
181. $display("[DRV] : DATA WRITE OP data :%0d and addr : %0d", datac.pwdata, datac.paddr);
182. end
183. else if (datac.oper == 1) //read
184. begin
185. @(posedge vif.pclk);
186. vif.psel <= 1'b1;
187. vif.penable <= 1'b0;
188. vif.pwdata <= datac.pwdata;
189. vif.paddr <= datac.paddr;
190. vif.pwrite <= 1'b0;
191. @(posedge vif.pclk);
192. vif.penable <= 1'b1;
193. repeat(2) @(posedge vif.pclk);
194. vif.psel <= 1'b0;
195. vif.penable <= 1'b0;
196. vif.pwrite <= 1'b0;
198. $display("[DRV] : DATA READ OP addr : %0d", datac.paddr);
199. end
200. else if (datac.oper== 2) ///random
201. begin
202. @(posedge vif.pclk);
203. vif.psel <= 1;
204. vif.penable <= 0;
205. vif.pwdata <= datac.pwdata;
206. vif.paddr <= datac.paddr;
207. vif.pwrite <= datac.pwrite;
208. @(posedge vif.pclk);
209. vif.penable <= 1;
210. repeat(2) @(posedge vif.pclk);
211. vif.psel <= 1'b0;
212. vif.penable <= 1'b0;
213. vif.pwrite <= 1'b0;
214. $display("[DRV] : RANDOM OPERATION");
215. end
216. else if (datac.oper == 3) ///slv error
217. begin
218. @(posedge vif.pclk);
219. vif.psel <= 1;
220. vif.penable <= 0;
221. vif.pwdata <= datac.pwdata;
222. vif.paddr <= $urandom\_range(32,100);
223. vif.pwrite <= datac.pwrite;
224. @(posedge vif.pclk);
225. vif.penable <= 1;
226. repeat(2) @(posedge vif.pclk);
227. vif.psel <= 1'b0;
228. vif.penable <= 1'b0;
229. vif.pwrite <= 1'b0;
230. $display("[DRV] : SLV ERROR");
231. end

234. ->nextdrv;
235. end
236. endtask

239. endclass


243. /\*
244. module tb;
246. generator gen;
247. driver drv;
248. mailbox #(transaction) mbx;
249. event next;
251. abp\_if vif();
252. apb\_ram dut (vif.presetn, vif.pclk, vif.psel, vif.penable, vif.pwrite, vif.paddr, vif.pwdata, vif.prdata, vif.pready,vif.pslverr);
254. initial begin
255. vif.pclk <= 0;
256. end
258. always #10 vif.pclk <= ~vif.pclk;
260. initial begin
261. mbx = new();
263. gen = new(mbx);
265. gen.count = 20;
267. drv = new(mbx);
269. drv.vif = vif;
271. gen.nextdrv = next;
272. drv.nextdrv = next;
274. fork
275. gen.run();
276. drv.run();
277. join\_none
278. wait(gen.done.triggered);
279. $finish();
280. end
282. initial begin
284. end
286. initial begin
287. $dumpfile("dump.vcd");
288. $dumpvars;
289. end

292. endmodule
294. \*/


298. //////////////////////////////////
300. class monitor;
302. virtual abp\_if vif;
304. mailbox #(transaction) mbx;
306. transaction tr;



311. function new(mailbox #(transaction) mbx);
312. this.mbx = mbx;
313. endfunction;
315. task run();
316. tr = new();
317. forever begin
318. @(posedge vif.pclk);
319. if((vif.psel) && (!vif.penable))
320. begin
322. @(posedge vif.pclk);
323. if(vif.psel && vif.pwrite && vif.penable) ////write access
324. begin
325. @(posedge vif.pclk);
326. tr.pwdata = vif.pwdata;
327. tr.paddr = vif.paddr;
328. tr.pwrite = vif.pwrite;
329. tr.pslverr = vif.pslverr;
330. $display("[MON] : DATA WRITE data : %0d and addr : %0d write :%0b", vif.pwdata, vif.paddr, vif.pwrite);
331. @(posedge vif.pclk);
332. end ////write access
333. else if (vif.psel && !vif.pwrite && vif.penable)
334. begin
335. @(posedge vif.pclk);
336. tr.prdata = vif.prdata;
337. tr.pwrite = vif.pwrite;
338. tr.paddr = vif.paddr;
339. tr.pslverr = vif.pslverr;
340. @(posedge vif.pclk);
342. $display("[MON] : DATA READ data : %0d and addr : %0d write:%0b", vif.prdata,vif.paddr, vif.pwrite);


346. end


350. mbx.put(tr);
351. end
352. end
353. endtask


357. endclass
359. ///////////////////////////////////////////////
361. class scoreboard;
363. mailbox #(transaction) mbx;
364. transaction tr;
365. event nextsco;
367. bit [31:0] pwdata[12] = '{default:0};
368. bit [31:0] rdata;
369. int index;
371. function new(mailbox #(transaction) mbx);
372. this.mbx = mbx;
373. endfunction;
375. task run();
376. forever begin
378. mbx.get(tr);
379. $display("[SCO] : DATA RCVD wdata:%0d rdata:%0d addr:%0d write:%0b", tr.pwdata, tr.prdata, tr.paddr, tr.pwrite);
381. if( (tr.pwrite == 1'b1) && (tr.pslverr == 1'b0)) ///write access
382. begin
383. pwdata[tr.paddr] = tr.pwdata;
384. $display("[SCO] : DATA STORED DATA : %0d ADDR: %0d",tr.pwdata, tr.paddr);
385. end
386. else if((tr.pwrite == 1'b0) && (tr.pslverr == 1'b0)) ///read access
387. begin
388. rdata = pwdata[tr.paddr];
389. if( tr.prdata == rdata)
390. $display("[SCO] : Data Matched");
391. else
392. $display("[SCO] : Data Mismatched");
393. end
394. else if(tr.pslverr == 1'b1)
395. begin
396. $display("[SCO] : SLV ERROR DETECTED");
397. end
398. ->nextsco;
399. end
400. endtask

403. endclass
405. //////////////////////////////////////////////////////////
407. class environment;
409. generator gen;
410. driver drv;
411. monitor mon;
412. scoreboard sco;


416. event nextgd; ///gen -> drv
418. event nextgs; /// gen -> sco
420. mailbox #(transaction) gdmbx; ///gen - drv

423. mailbox #(transaction) msmbx; /// mon - sco
425. virtual abp\_if vif;

428. function new(virtual abp\_if vif);
430. gdmbx = new();
431. gen = new(gdmbx);
432. drv = new(gdmbx);

435. msmbx = new();
436. mon = new(msmbx);
437. sco = new(msmbx);
439. this.vif = vif;
440. drv.vif = this.vif;
441. mon.vif = this.vif;
443. gen.nextsco = nextgs;
444. sco.nextsco = nextgs;
446. gen.nextdrv = nextgd;
447. drv.nextdrv = nextgd;
449. endfunction
451. task pre\_test();
452. drv.reset();
453. endtask
455. task test();
456. fork
457. gen.run();
458. drv.run();
459. mon.run();
460. sco.run();
461. join\_any
462. endtask
464. task post\_test();
465. wait(gen.done.triggered);
466. $finish();
467. endtask
469. task run();
470. pre\_test();
471. test();
472. post\_test();
473. endtask


477. endclass

480. //////////////////////////////////////////////////
481. module tb;
483. abp\_if vif();
485. apb\_ram dut (vif.presetn, vif.pclk, vif.psel, vif.penable, vif.pwrite, vif.paddr, vif.pwdata, vif.prdata, vif.pready,vif.pslverr);
487. initial begin
488. vif.pclk <= 0;
489. end
491. always #10 vif.pclk <= ~vif.pclk;
493. environment env;


497. initial begin
498. env = new(vif);
499. env.gen.count = 30;
500. env.run();
501. end

504. initial begin
505. $dumpfile("dump.vcd");
506. $dumpvars;
507. end

510. endmodule

AXI MEMORY:

1. interface axi\_if();
3. ////////write address channel (aw)
5. logic awvalid; /// master is sending new address
6. logic awready; /// slave is ready to accept request
7. logic [3:0] awid; ////// unique ID for each transaction
8. logic [3:0] awlen; ////// burst length AXI3 : 1 to 16, AXI4 : 1 to 256
9. logic [2:0] awsize; ////unique transaction size : 1,2,4,8,16 ...128 bytes
10. logic [31:0] awaddr; ////write adress of transaction
11. logic [1:0] awburst; ////burst type : fixed , INCR , WRAP

14. //////////write data channel (w)
15. logic wvalid; //// master is sending new data
16. logic wready; //// slave is ready to accept new data
17. logic [3:0] wid; /// unique id for transaction
18. logic [31:0] wdata; //// data
19. logic [3:0] wstrb; //// lane having valid data
20. logic wlast; //// last transfer in write burst

23. //////////write response channel (b)
24. logic bready; ///master is ready to accept response
25. logic bvalid; //// slave has valid response
26. logic [3:0] bid; ////unique id for transaction
27. logic [1:0] bresp; /// status of write transaction
29. ///////////////read address channel (ar)
31. logic arvalid; /// master is sending new address
32. logic arready; /// slave is ready to accept request
33. logic [3:0] arid; ////// unique ID for each transaction
34. logic [3:0] arlen; ////// burst length AXI3 : 1 to 16, AXI4 : 1 to 256
35. logic [2:0] arsize; ////unique transaction size : 1,2,4,8,16 ...128 bytes
36. logic [31:0] araddr; ////write adress of transaction
37. logic [1:0] arburst; ////burst type : fixed , INCR , WRAP
39. /////////// read data channel (r)
41. logic rvalid; //// master is sending new data
42. logic rready; //// slave is ready to accept new data
43. logic [3:0] rid; /// unique id for transaction
44. logic [31:0] rdata; //// data
45. logic [3:0] rstrb; //// lane having valid data
46. logic rlast; //// last transfer in write burst
47. logic [1:0] rresp; ///status of read transfer
49. ////////////////
51. logic clk;
52. logic resetn;
54. //////////////////
55. logic [31:0] addr\_wrapwr;
56. logic [31:0] addr\_wraprd;
58. endinterface
60. ///////////////////////////////////////////////
61. module axi\_slave(
62. ////////////////global control signals
63. input clk,
64. input resetn,
66. ///////////////////write address channel
68. input awvalid, /// master is sending new address
69. output reg awready, /// slave is ready to accept request
70. input [3:0] awid, ////// unique ID for each transaction
71. input [3:0] awlen, ////// burst length AXI3 : 1 to 16, AXI4 : 1 to 256
72. input [2:0] awsize, ////unique transaction size : 1,2,4,8,16 ...128 bytes
73. input [31:0] awaddr, ////write adress of transaction
74. input [1:0] awburst, ////burst type : fixed , INCR , WRAP
76. /////////////////////write data channel
78. input wvalid, //// master is sending new data
79. output reg wready, //// slave is ready to accept new data
80. input [3:0] wid, /// unique id for transaction
81. input [31:0] wdata, //// data
82. input [3:0] wstrb, //// lane having valid data
83. input wlast, //// last transfer in write burst
85. ///////////////write response channel
87. input bready, ///master is ready to accept response
88. output reg bvalid, //// slave has valid response
89. output reg [3:0] bid, ////unique id for transaction
90. output reg [1:0] bresp, /// status of write transaction
92. ////////////// read address channel
94. output reg arready, //read address ready signal from slave
95. input [3:0] arid, //read address id
96. input [31:0] araddr, //read address signal
97. input [3:0] arlen, //length of the burst
98. input [2:0] arsize, //number of bytes in a transfer
99. input [1:0] arburst, //burst type - fixed, incremental, wrapping
100. input arvalid, //address read valid signal
102. ///////////////////read data channel
103. output reg [3:0] rid, //read data id
104. output reg [31:0]rdata, //read data from slave
105. output reg [1:0] rresp, //read response signal
106. output reg rlast, //read data last signal
107. output reg rvalid, //read data valid signal
108. input rready
110. );

113. //axi\_if vif();
114. typedef enum bit [1:0] {awidle = 2'b00, awstart = 2'b01, awreadys = 2'b10} awstate\_type;
115. awstate\_type awstate, awnext\_state;
117. reg [31:0] awaddrt;
119. //////reset decoder
120. always\_ff@(posedge clk , negedge resetn)
121. begin
122. if(!resetn) begin
123. awstate <= awidle; ///idle state for write address FSM
124. wstate <= widle; ///idle state for write data fsm
125. bstate <= bidle; ///// idle state for write response fsm
126. end
127. else
128. begin
129. awstate <= awnext\_state;
130. wstate <= wnext\_state;
131. bstate <= bnext\_state;
132. end
133. end

136. /////////////fsm for write address channel
137. always\_comb
138. begin
139. case(awstate)
140. awidle:
141. begin
142. awready = 1'b0;
143. awnext\_state = awstart;
144. end
146. awstart:
147. begin
148. if(awvalid)
149. begin
150. awnext\_state = awreadys;
151. awaddrt = awaddr; ////storing address
152. end
153. else
154. awnext\_state = awstart;
155. end
157. awreadys:
158. begin
159. awready = 1'b1;
160. awnext\_state = awidle;
161. end
162. endcase
163. end


167. ////////////////////fsm for write data channel

170. reg [31:0] wdatat;
171. reg [7:0] mem[128] = '{default:12};
172. reg [31:0] retaddr;
173. reg [31:0] nextaddr;
174. reg first; /// check operation executed first time


178. ///////////////////////////function to compute next address during FIXED burst type
179. function bit[31:0] data\_wr\_fixed (input [3:0] wstrb, input [31:0] awaddrt);
180. unique case (wstrb)
181. 4'b0001: begin
182. mem[awaddrt] = wdatat[7:0];
183. end
185. 4'b0010: begin
186. mem[awaddrt] = wdatat[15:8];
187. end
189. 4'b0011: begin
190. mem[awaddrt] = wdatat[7:0];
191. mem[awaddrt + 1] = wdatat[15:8];
192. end
194. 4'b0100: begin
195. mem[awaddrt] = wdatat[23:16];
196. end
198. 4'b0101: begin
199. mem[awaddrt] = wdatat[7:0];
200. mem[awaddrt + 1] = wdatat[23:16];
201. end

204. 4'b0110: begin
205. mem[awaddrt] = wdatat[15:8];
206. mem[awaddrt + 1] = wdatat[23:16];
207. end
209. 4'b0111: begin
210. mem[awaddrt] = wdatat[7:0];
211. mem[awaddrt + 1] = wdatat[15:8];
212. mem[awaddrt + 2] = wdatat[23:16];
213. end
215. 4'b1000: begin
216. mem[awaddrt] = wdatat[31:24];
217. end
219. 4'b1001: begin
220. mem[awaddrt] = wdatat[7:0];
221. mem[awaddrt + 1] = wdatat[31:24];
222. end

225. 4'b1010: begin
226. mem[awaddrt] = wdatat[15:8];
227. mem[awaddrt + 1] = wdatat[31:24];
228. end

231. 4'b1011: begin
232. mem[awaddrt] = wdatat[7:0];
233. mem[awaddrt + 1] = wdatat[15:8];
234. mem[awaddrt + 2] = wdatat[31:24];
235. end
237. 4'b1100: begin
238. mem[awaddrt] = wdatat[23:16];
239. mem[awaddrt + 1] = wdatat[31:24];
240. end
242. 4'b1101: begin
243. mem[awaddrt] = wdatat[7:0];
244. mem[awaddrt + 1] = wdatat[23:16];
245. mem[awaddrt + 2] = wdatat[31:24];
246. end
248. 4'b1110: begin
249. mem[awaddrt] = wdatat[15:8];
250. mem[awaddrt + 1] = wdatat[23:16];
251. mem[awaddrt + 2] = wdatat[31:24];
252. end
254. 4'b1111: begin
255. mem[awaddrt] = wdatat[7:0];
256. mem[awaddrt + 1] = wdatat[15:8];
257. mem[awaddrt + 2] = wdatat[23:16];
258. mem[awaddrt + 3] = wdatat[31:24];
259. end
260. endcase
261. return awaddrt;
262. endfunction

265. ///////////////////////////function to compute next address during INCR burst type
267. function bit[31:0] data\_wr\_incr (input [3:0] wstrb, input [31:0] awaddrt);
269. bit [31:0] addr;
271. unique case (wstrb)
272. 4'b0001: begin
273. mem[awaddrt] = wdatat[7:0];
274. addr = awaddrt + 1;
275. end
277. 4'b0010: begin
278. mem[awaddrt] = wdatat[15:8];
279. addr = awaddrt + 1;
280. end
282. 4'b0011: begin
283. mem[awaddrt] = wdatat[7:0];
284. mem[awaddrt + 1] = wdatat[15:8];
285. addr = awaddrt + 2;
286. end
288. 4'b0100: begin
289. mem[awaddrt] = wdatat[23:16];
290. addr = awaddrt + 1;
291. end
293. 4'b0101: begin
294. mem[awaddrt] = wdatat[7:0];
295. mem[awaddrt + 1] = wdatat[23:16];
296. addr = awaddrt + 2;
297. end

300. 4'b0110: begin
301. mem[awaddrt] = wdatat[15:8];
302. mem[awaddrt + 1] = wdatat[23:16];
303. addr = awaddrt + 2;
304. end
306. 4'b0111: begin
307. mem[awaddrt] = wdatat[7:0];
308. mem[awaddrt + 1] = wdatat[15:8];
309. mem[awaddrt + 2] = wdatat[23:16];
310. addr = awaddrt + 3;
311. end
313. 4'b1000: begin
314. mem[awaddrt] = wdatat[31:24];
315. addr = awaddrt + 1;
316. end
318. 4'b1001: begin
319. mem[awaddrt] = wdatat[7:0];
320. mem[awaddrt + 1] = wdatat[31:24];
321. addr = awaddrt + 2;
322. end

325. 4'b1010: begin
326. mem[awaddrt] = wdatat[15:8];
327. mem[awaddrt + 1] = wdatat[31:24];
328. addr = awaddrt + 2;
329. end

332. 4'b1011: begin
333. mem[awaddrt] = wdatat[7:0];
334. mem[awaddrt + 1] = wdatat[15:8];
335. mem[awaddrt + 2] = wdatat[31:24];
336. addr = awaddrt + 3;
337. end
339. 4'b1100: begin
340. mem[awaddrt] = wdatat[23:16];
341. mem[awaddrt + 1] = wdatat[31:24];
342. addr = awaddrt + 2;
343. end
345. 4'b1101: begin
346. mem[awaddrt] = wdatat[7:0];
347. mem[awaddrt + 1] = wdatat[23:16];
348. mem[awaddrt + 2] = wdatat[31:24];
349. addr = awaddrt + 3;
350. end
352. 4'b1110: begin
353. mem[awaddrt] = wdatat[15:8];
354. mem[awaddrt + 1] = wdatat[23:16];
355. mem[awaddrt + 2] = wdatat[31:24];
356. addr = awaddrt + 3;
357. end
359. 4'b1111: begin
360. mem[awaddrt] = wdatat[7:0];
361. mem[awaddrt + 1] = wdatat[15:8];
362. mem[awaddrt + 2] = wdatat[23:16];
363. mem[awaddrt + 3] = wdatat[31:24];
364. addr = awaddrt + 4;
365. end
366. endcase
367. return addr;
368. endfunction
370. ///////////////////////Function to compute Wrapping boundary

373. function bit [7:0] wrap\_boundary (input bit [3:0] awlen,input bit[2:0] awsize);
374. bit [7:0] boundary;
376. unique case(awlen)
377. 4'b0001:
378. begin
379. unique case(awsize)
380. 3'b000: begin
381. boundary = 2 \* 1;
382. end
383. 3'b001: begin
384. boundary = 2 \* 2;
385. end
386. 3'b010: begin
387. boundary = 2 \* 4;
388. end
389. endcase
390. end
391. 4'b0011:
392. begin
393. unique case(awsize)
394. 3'b000: begin
395. boundary = 4 \* 1;
396. end
397. 3'b001: begin
398. boundary = 4 \* 2;
399. end
400. 3'b010: begin
401. boundary = 4 \* 4;
402. end
403. endcase
404. end
406. 4'b0111:
407. begin
408. unique case(awsize)
409. 3'b000: begin
410. boundary = 8 \* 1;
411. end
412. 3'b001: begin
413. boundary = 8 \* 2;
414. end
415. 3'b010: begin
416. boundary = 8 \* 4;
417. end
418. endcase
419. end

422. 4'b1111:
423. begin
424. unique case(awsize)
425. 3'b000: begin
426. boundary = 16 \* 1;
427. end
428. 3'b001: begin
429. boundary = 16 \* 2;
430. end
431. 3'b010: begin
432. boundary = 16 \* 4;
433. end
434. endcase
435. end
437. endcase

440. return boundary;
441. endfunction
442. //////////////////////////////////////////////////////////////
444. function bit[31:0] data\_wr\_wrap (input [3:0] wstrb, input [31:0] awaddrt, input [7:0] wboundary);
446. bit [31:0] addr1, addr2, addr3, addr4;
447. bit [31:0] nextaddr, nextaddr2;
449. unique case (wstrb)
451. /////////////////////////////////////////////////
452. 4'b0001: begin
453. mem[awaddrt] = wdatat[7:0];
455. if((awaddrt + 1) % wboundary == 0)
456. addr1 = (awaddrt + 1) - wboundary;
457. else
458. addr1 = awaddrt + 1;
460. return addr1;
461. end
463. /////////////////////////////////////////////////
465. 4'b0010: begin
466. mem[awaddrt] = wdatat[15:8];
468. if((awaddrt + 1) % wboundary == 0)
469. addr1 = (awaddrt + 1) - wboundary;
470. else
471. addr1 = awaddrt + 1;
473. return addr1;
474. end
476. ///////////////////////////////////////////////////
478. 4'b0011: begin
479. mem[awaddrt] = wdatat[7:0];
481. if((awaddrt + 1) % wboundary == 0)
482. addr1 = (awaddrt + 1) - wboundary;
483. else
484. addr1 = awaddrt + 1;
486. mem[addr1] = wdatat[15:8];
488. if((addr1 + 1) % wboundary == 0)
489. addr2 = (addr1 + 1) - wboundary;
490. else
491. addr2 = addr1 + 1;
493. return addr2;
495. end
497. ///////////////////////////////////////////////
499. 4'b0100: begin
500. mem[awaddrt] = wdatat[23:16];
502. if((awaddrt + 1) % wboundary == 0)
503. addr1 = (awaddrt + 1) - wboundary;
504. else
505. addr1 = awaddrt + 1;
507. return addr1;
508. end
510. //////////////////////////////////////////////
512. 4'b0101: begin
513. mem[awaddrt] = wdatat[7:0];
515. if((awaddrt + 1) % wboundary == 0)
516. addr1 = (awaddrt + 1) - wboundary;
517. else
518. addr1 = awaddrt + 1;

521. mem[addr1] = wdatat[23:16];

524. if((addr1 + 1) % wboundary == 0)
525. addr2 = (addr1 + 1) - wboundary;
526. else
527. addr2 = addr1 + 1;
529. return addr2;
531. end
533. ///////////////////////////////////////////////////
535. 4'b0110: begin
536. mem[awaddrt] = wdatat[15:8];
538. if((awaddrt + 1) % wboundary == 0)
539. addr1 = (awaddrt + 1) - wboundary;
540. else
541. addr1 = awaddrt + 1;
543. mem[addr1] = wdatat[23:16];
545. if((addr1 + 1) % wboundary == 0)
546. addr2 = (addr1 + 1) - wboundary;
547. else
548. addr2 = addr1 + 1;
550. return addr2;
552. end
553. //////////////////////////////////////////////////////////////
555. 4'b0111: begin
556. mem[awaddrt] = wdatat[7:0];
557. if((awaddrt + 1) % wboundary == 0)
558. addr1 = (awaddrt + 1) - wboundary;
559. else
560. addr1 = awaddrt + 1;
562. mem[addr1] = wdatat[15:8];
564. if((addr1 + 1) % wboundary == 0)
565. addr2 = (addr1 + 1) - wboundary;
566. else
567. addr2 = addr1 + 1;
569. mem[addr2] = wdatat[23:16];
571. if((addr2 + 1) % wboundary == 0)
572. addr3 = (addr2 + 1) - wboundary;
573. else
574. addr3 = addr2 + 1;
576. return addr3;
577. end
579. 4'b1000: begin
580. mem[awaddrt] = wdatat[31:24];
582. if((awaddrt + 1) % wboundary == 0)
583. addr1 = (awaddrt + 1) - wboundary;
584. else
585. addr1 = awaddrt + 1;
587. return addr1;
588. end
590. 4'b1001: begin
591. mem[awaddrt] = wdatat[7:0];
593. if((awaddrt + 1) % wboundary == 0)
594. addr1 = (awaddrt + 1) - wboundary;
595. else
596. addr1 = awaddrt + 1;

599. mem[addr1] = wdatat[31:24];
601. if((addr1 + 1) % wboundary == 0)
602. addr2 = (addr1 + 1) - wboundary;
603. else
604. addr2 = addr1 + 1;
606. return addr2;
607. end

610. 4'b1010: begin
611. mem[awaddrt] = wdatat[15:8];
613. if((awaddrt + 1) % wboundary == 0)
614. addr1 = (awaddrt + 1) - wboundary;
615. else
616. addr1 = awaddrt + 1;
618. mem[addr1] = wdatat[31:24];
620. if((addr1 + 1) % wboundary == 0)
621. addr2 = (addr1 + 1) - wboundary;
622. else
623. addr2 = addr1 + 1;
625. return addr2;
626. end

629. 4'b1011: begin
630. mem[awaddrt] = wdatat[7:0];
632. if((awaddrt + 1) % wboundary == 0)
633. addr1 = (awaddrt + 1) - wboundary;
634. else
635. addr1 = awaddrt + 1;

638. mem[addr1] = wdatat[15:8];
640. if((addr1 + 1) % wboundary == 0)
641. addr2 = (addr1 + 1) - wboundary;
642. else
643. addr2 = addr1 + 1;
645. mem[addr2] = wdatat[31:24];
647. if((addr2 + 1) % wboundary == 0)
648. addr3 = (addr2 + 1) - wboundary;
649. else
650. addr3 = addr2 + 1;
652. return addr3;
654. end
656. 4'b1100: begin
657. mem[awaddrt] = wdatat[23:16];
659. if((awaddrt + 1) % wboundary == 0)
660. addr1 = (awaddrt + 1) - wboundary;
661. else
662. addr1 = awaddrt + 1;
664. mem[addr1] = wdatat[31:24];
666. if((addr1 + 1) % wboundary == 0)
667. addr2 = (addr1 + 1) - wboundary;
668. else
669. addr2 = addr1 + 1;
671. return addr2;
672. end
674. 4'b1101: begin
675. mem[awaddrt] = wdatat[7:0];
677. if((awaddrt + 1) % wboundary == 0)
678. addr1 = (awaddrt + 1) - wboundary;
679. else
680. addr1 = awaddrt + 1;
682. mem[addr1] = wdatat[23:16];
684. if((addr1 + 1) % wboundary == 0)
685. addr2 = (addr1 + 1) - wboundary;
686. else
687. addr2 = addr1 + 1;
689. mem[addr2] = wdatat[31:24];
691. if((addr2 + 1) % wboundary == 0)
692. addr3 = (addr2 + 1) - wboundary;
693. else
694. addr3 = addr2 + 1;
696. return addr3;
698. end
700. 4'b1110: begin
701. mem[awaddrt] = wdatat[15:8];
703. if((awaddrt + 1) % wboundary == 0)
704. addr1 = (awaddrt + 1) - wboundary;
705. else
706. addr1 = awaddrt + 1;
708. mem[addr1] = wdatat[23:16];
710. if((addr1 + 1) % wboundary == 0)
711. addr2 = (addr1 + 1) - wboundary;
712. else
713. addr2 = addr1 + 1;
715. mem[addr2] = wdatat[31:24];
717. if((addr2 + 1) % wboundary == 0)
718. addr3 = (addr2 + 1) - wboundary;
719. else
720. addr3 = addr2 + 1;
722. return addr3;
723. end
725. 4'b1111: begin
726. mem[awaddrt] = wdatat[7:0];
728. if((awaddrt + 1) % wboundary == 0)
729. addr1 = (awaddrt + 1) - wboundary;
730. else
731. addr1 = awaddrt + 1;
733. mem[addr1] = wdatat[15:8];
735. if((addr1 + 1) % wboundary == 0)
736. addr2 = (addr1 + 1) - wboundary;
737. else
738. addr2 = addr1 + 1;
740. mem[addr2] = wdatat[23:16];
742. if((addr2 + 1) % wboundary == 0)
743. addr3 = (addr2 + 1) - wboundary;
744. else
745. addr3 = addr2 + 1;

748. mem[addr3] = wdatat[31:24];
750. if((addr3 + 1) % wboundary == 0)
751. addr4 = (addr3 + 1) - wboundary;
752. else
753. addr4 = addr3 + 1;
755. return addr4;
756. end
757. endcase
759. endfunction








769. reg [7:0] boundary; ////storing boundary
770. reg [3:0] wlen\_count;
772. typedef enum bit [2:0] {widle = 0, wstart = 1, wreadys = 2, wvalids = 3, waddr\_dec = 4} wstate\_type;
773. wstate\_type wstate, wnext\_state;
775. always\_comb
776. begin
777. case(wstate)
779. widle: begin
780. wready = 1'b0;
781. wnext\_state = wstart;
782. first = 1'b0;
783. wlen\_count = 0;
784. end
786. wstart: begin
787. if(wvalid)
788. begin
789. wnext\_state = waddr\_dec;
790. wdatat = wdata;
791. end
792. else
793. begin
794. wnext\_state = wstart;
795. end
796. end
798. waddr\_dec: begin
799. wnext\_state = wreadys;
800. if(first == 0) begin
801. nextaddr = awaddr;
802. first = 1'b1;
803. end
804. else if (wlen\_count != (awlen + 1 ))
805. begin
806. nextaddr = retaddr;
807. end
808. else
809. begin
810. nextaddr = awaddr;
811. end
812. end


816. wreadys: begin
818. if(wlast == 1'b1) begin
819. wnext\_state = widle;
820. wready = 1'b0;
821. wlen\_count = 0;
822. first = 0;
823. end
824. else
825. begin
826. wnext\_state = wvalids;
827. wready = 1'b1;
828. end

831. case(awburst)
832. 2'b00: ////Fixed Mode
833. begin
834. retaddr = data\_wr\_fixed(wstrb, awaddr); ///fixed
835. end
837. 2'b01: ////Incr mode
838. begin
839. retaddr = data\_wr\_incr(wstrb,nextaddr);
840. end
842. 2'b10: //// wrapping
843. begin
844. boundary = wrap\_boundary(awlen, awsize); /////calculate wrapping boundary
845. retaddr = data\_wr\_wrap(wstrb, nextaddr, boundary); ///////generate next addr
846. end
847. endcase
848. end

851. wvalids: begin
852. wready = 1'b0;
853. wnext\_state = wstart;
855. if(wlen\_count != (awlen + 1))
856. wlen\_count = wlen\_count + 1;
857. else
858. wlen\_count = wlen\_count;
860. end
861. endcase
862. end


866. ////////////////////////fsm for write response
868. typedef enum bit [1:0] {bidle = 0, bdetect\_last = 1, bstart = 2, bwait = 3} bstate\_type;
869. bstate\_type bstate,bnext\_state;

872. always\_comb
873. begin
874. case(bstate)
875. bidle: begin
876. bid = 1'b0;
877. bresp = 1'b0;
878. bvalid = 1'b0;
879. bnext\_state = bdetect\_last;
880. end
882. bdetect\_last: begin
883. if(wlast)
884. bnext\_state = bstart;
885. else
886. bnext\_state = bdetect\_last;
887. end
889. bstart: begin
890. bid = awid;
891. bvalid = 1'b1;
892. bnext\_state = bwait;
893. if( (awaddr < 128 ) && (awsize <= 3'b011) )
894. bresp = 2'b00; ///okay
895. else if (awsize > 3'b011)
896. bresp = 2'b10; /////slverr
897. else
898. bresp = 2'b11; ///no slave address
899. end
901. bwait: begin
902. if(bready == 1'b1)
903. bnext\_state = bidle;
904. else
905. bnext\_state = bwait;
906. end
908. endcase
909. end
911. ////////////////////////fsm for read address
913. always\_ff @(posedge clk, negedge resetn)
914. begin
915. if(!resetn)
916. begin
917. arstate <= aridle;
918. rstate <= ridle;
919. end
920. else
921. begin
922. arstate <= arnext\_state;
923. rstate <= rnext\_state;
924. end
925. end

928. typedef enum bit [1:0] {aridle = 0, arstart = 1, arreadys = 2} arstate\_type;
929. arstate\_type arstate, arnext\_state;
931. reg [31:0] araddrt; ///register address
933. always\_comb
934. begin
935. case(arstate)
936. aridle: begin
937. arready = 1'b0;
938. arnext\_state = arstart;
939. end
941. arstart: begin
942. if(arvalid == 1'b1) begin
943. arnext\_state = arreadys;
944. araddrt = araddr;
945. end
946. else
947. arnext\_state = arstart;
948. end
950. arreadys: begin
951. arnext\_state = aridle;
952. arready = 1'b1;
953. end
954. endcase
955. end
957. ///////////////////////////read data in FIxed Mode
959. function void read\_data\_fixed (input [31:0] addr, input [2:0] arsize);
960. unique case(arsize)
961. 3'b000: begin
962. rdata[7:0] = mem[addr];
963. end
965. 3'b001: begin
966. rdata[7:0] = mem[addr];
967. rdata[15:8] = mem[addr + 1];
968. end
970. 3'b010: begin
971. rdata[7:0] = mem[addr];
972. rdata[15:8] = mem[addr + 1];
973. rdata[23:16] = mem[addr + 2];
974. rdata[31:24] = mem[addr + 3];
975. end
976. endcase
977. endfunction
978. //////////////////////////end of function
979. //////////////////////////// read data in INCR Mode
981. function bit [31:0] read\_data\_incr (input [31:0] addr, input [2:0] arsize);
982. bit [31:0] nextaddr;
984. unique case(arsize)
985. 3'b000: begin
986. rdata[7:0] = mem[addr];
987. nextaddr = addr + 1;
988. end
990. 3'b001: begin
991. rdata[7:0] = mem[addr];
992. rdata[15:8] = mem[addr + 1];
993. nextaddr = addr + 2;
994. end
996. 3'b010: begin
997. rdata[7:0] = mem[addr];
998. rdata[15:8] = mem[addr + 1];
999. rdata[23:16] = mem[addr + 2];
1000. rdata[31:24] = mem[addr + 3];
1001. nextaddr = addr + 4;
1002. end
1004. endcase
1006. return nextaddr;
1008. endfunction
1010. ///////////////////////////////////////////end of function
1011. function bit [31:0] read\_data\_wrap (input bit [31:0] addr, input bit [2:0] rsize, input [7:0] rboundary);
1012. bit [31:0] addr1,addr2,addr3,addr4;
1014. unique case (rsize)
1015. 3'b000: begin
1016. rdata[7:0] = mem[addr];
1018. if(((addr + 1) % rboundary ) == 0)
1019. addr1 = (addr + 1) - rboundary;
1020. else
1021. addr1 = (addr + 1);
1023. return addr1;
1024. end
1026. 3'b001: begin
1027. rdata[7:0] = mem[addr];
1029. if(((addr + 1) % rboundary ) == 0)
1030. addr1 = (addr + 1) - rboundary;
1031. else
1032. addr1 = (addr + 1);
1034. rdata[15:8] = mem[addr1];
1036. if(((addr1 + 1) % rboundary ) == 0)
1037. addr2 = (addr1 + 1) - rboundary;
1038. else
1039. addr2 = (addr1 + 1);
1041. return addr2;
1042. end
1044. 3'b010: begin
1046. rdata[7:0] = mem[addr];
1048. if(((addr + 1) % rboundary ) == 0)
1049. addr1 = (addr + 1) - rboundary;
1050. else
1051. addr1 = (addr + 1);
1053. rdata[15:8] = mem[addr1];
1055. if(((addr1 + 1) % rboundary ) == 0)
1056. addr2 = (addr1 + 1) - rboundary;
1057. else
1058. addr2 = (addr1 + 1);
1060. rdata[23:16] = mem[addr2];
1062. if(((addr2 + 1) % rboundary ) == 0)
1063. addr3 = (addr2 + 1) - rboundary;
1064. else
1065. addr3 = (addr2 + 1);
1067. rdata[31:24] = mem[addr3];
1069. if(((addr3 + 1) % rboundary ) == 0)
1070. addr4 = (addr3 + 1) - rboundary;
1071. else
1072. addr4 = (addr3 + 1);
1074. return addr4;
1075. end
1077. endcase
1079. endfunction
1081. ///////////////////////////////////////
1082. reg rdfirst;
1083. bit [31:0] rdnextaddr, rdretaddr;
1084. reg [3:0] len\_count;
1085. reg [7:0] rdboundary;
1087. typedef enum bit [2:0] {ridle = 0, rstart = 1, rwait = 2, rvalids = 3, rerror = 4} rstate\_type;
1088. rstate\_type rstate, rnext\_state;
1090. /////////////////////////////////
1091. always\_comb
1092. begin
1093. case(rstate)
1094. ridle: begin
1096. rid = 0;
1097. rdfirst = 0;
1098. rdata = 0;
1099. rresp = 0;
1100. rlast = 0;
1101. rvalid = 0;
1102. len\_count = 0;
1104. if(arvalid)
1105. rnext\_state = rstart;
1106. else
1107. rnext\_state = ridle;
1108. end
1110. rstart: begin
1111. if ((araddrt < 128) && (arsize <= 3'b010) ) begin
1112. rid = arid;
1113. rvalid = 1'b1;
1114. rnext\_state = rwait;
1115. rresp = 2'b00;
1116. unique case(arburst)
1118. ////////////////////fixed
1119. 2'b00: begin
1120. if(rdfirst == 0) begin
1121. rdnextaddr = araddr;
1122. rdfirst = 1'b1;
1123. len\_count = 0;
1124. end
1125. else if (len\_count != (arlen + 1))
1126. begin
1127. rdnextaddr = araddr;
1128. end
1130. read\_data\_fixed(araddrt, arsize);
1131. end
1132. //////////////////////end of fixed
1134. ////////////////////start of incr
1135. 2'b01: begin
1136. if(rdfirst == 0) begin
1137. rdnextaddr = araddr;
1138. rdfirst = 1'b1;
1139. len\_count = 0;
1140. end
1141. else if (len\_count != (arlen + 1))
1142. begin
1143. rdnextaddr = rdretaddr;
1144. end
1146. rdretaddr = read\_data\_incr(rdnextaddr, arsize);
1147. end
1148. ///////////////////////end of incr
1149. 2'b10: begin
1150. if(rdfirst == 0) begin
1151. rdnextaddr = araddr;
1152. rdfirst = 1'b1;
1153. len\_count = 0;
1154. end
1155. else if (len\_count != (arlen + 1))
1156. begin
1157. rdnextaddr = rdretaddr;
1158. end
1159. rdboundary = wrap\_boundary(arlen, arsize);
1160. rdretaddr = read\_data\_wrap(rdnextaddr, arsize, rdboundary);
1161. end
1162. endcase
1163. end
1164. else if ( (araddr > 128) && ( arsize <= 3'b010) ) begin
1165. rresp = 2'b11;
1166. rnext\_state = rerror;
1167. end
1168. else if (arsize > 3'b010) begin
1169. rresp = 2'b10;
1170. rnext\_state = rerror;
1171. end
1173. end
1175. rwait: begin
1176. rvalid = 1'b0;
1177. len\_count = len\_count + 1;
1178. if(rready == 1'b1)
1179. begin
1180. rnext\_state = rvalids;
1181. if(len\_count == (arlen + 1))
1182. rlast = 1'b1;
1183. else
1184. rlast = 1'b0;
1185. end
1186. else
1187. rnext\_state = rwait;
1188. end
1190. rvalids: begin
1192. if(len\_count == (arlen + 1))
1193. begin
1194. rnext\_state = ridle;
1195. end
1196. else
1197. begin
1198. rnext\_state = rstart;
1199. end
1200. end
1202. rerror : begin
1203. len\_count = len\_count + 1;
1204. if(len\_count == (arlen + 1))
1205. begin
1206. rnext\_state = rvalids;
1207. rlast = 1'b1;
1208. end
1209. else
1210. begin
1211. rlast = 1'b0;
1212. rnext\_state = rstart;
1213. end

1216. end
1217. endcase
1218. end
1219. endmodule


























1247. //////////////////////////Testbench Components
1248. /////////////////////////////////////////////////
1250. class transaction;
1252. rand bit [3:0] id;
1254. rand bit awvalid;
1255. bit awready;
1256. bit [3:0] awid;
1257. rand bit [3:0] awlen;
1258. rand bit [2:0] awsize; //4byte =010
1259. rand bit [31:0] awaddr;
1260. rand bit [1:0] awburst;
1262. bit wvalid;
1263. bit wready;
1264. bit [3:0] wid;
1265. rand bit [31:0] wdata;
1266. rand bit [3:0] wstrb;
1267. bit wlast;
1269. bit bready;
1270. bit bvalid;
1271. bit [3:0] bid;
1272. bit [1:0] bresp;

1275. rand bit arvalid; /// master is sending new address
1276. bit arready; /// slave is ready to accept request
1277. bit [3:0] arid; ////// unique ID for each transaction
1278. rand bit [3:0] arlen; ////// burst length AXI3 : 1 to 16, AXI4 : 1 to 256
1279. bit [2:0] arsize; ////unique transaction size : 1,2,4,8,16 ...128 bytes
1280. rand bit [31:0] araddr; ////write adress of transaction
1281. rand bit [1:0] arburst; ////burst type : fixed , INCR , WRAP
1283. /////////// read data channel (r)
1285. bit rvalid; //// master is sending new data
1286. bit rready; //// slave is ready to accept new data
1287. bit [3:0] rid; /// unique id for transaction
1288. bit [31:0] rdata; //// data
1289. bit [3:0] rstrb; //// lane having valid data
1290. bit rlast; //// last transfer in write burst
1291. bit [1:0] rresp; ///status of read transfer
1293. /\*
1294. constraint adr\_c {
1295. awvalid == 0;
1296. arvalid == 1;
1298. }
1299. \*/
1301. constraint valid\_c {
1302. arvalid != awvalid;
1303. }
1305. constraint addr\_c {
1306. awaddr == 5;
1307. //awaddr > 0; awaddr < 15;
1309. }
1311. constraint awsize\_c {
1313. awsize >= 0; awsize < 3;
1314. }
1316. constraint awburst\_c {
1317. awburst == 2;
1318. //awburst >= 0; awburst < 3;
1319. }

1322. constraint arlen\_c {
1323. arlen > 0; arlen <= 7;
1324. }

1327. constraint araddr\_c {
1328. araddr == 5;
1329. // araddr > 0; araddr <= 7;
1330. }
1332. constraint arburst\_c {
1333. arburst == 2;
1334. //arburst >= 0; arburst < 3;
1335. }
1337. endclass


1341. //////////////////////////////////
1342. class generator;
1344. transaction tr;
1345. mailbox #(transaction) mbxgd;
1347. event done; ///gen completed sending requested no. of transaction
1348. event drvnext; /// dr complete its wor;
1349. event sconext; ///scoreboard complete its work
1351. int count = 0;
1353. function new( mailbox #(transaction) mbxgd);
1354. this.mbxgd = mbxgd;
1355. tr =new();
1356. endfunction
1358. task run();
1360. for(int i=0; i <= count; i++) begin
1361. assert(tr.randomize) else $error("Randomization Failed");
1363. if(tr.awburst == 2'b10)
1364. begin
1365. tr.awlen = 4'b0111;
1366. end

1369. if(tr.arburst == 2'b10)
1370. begin
1371. tr.arlen = 4'b0111;
1372. end
1374. // $display("[GEN] : WRITE : %0b READ : %0b BURST MODE : %0d",tr.awvalid, tr.arvalid, tr.awburst);
1375. $display("[GEN] : WR :%0b RD:%0b WRBUR : %0d RDBUR: %0d WRADDR :%0d RDADDR : %0d WLEN :%0d RLEN :%0d",tr.awvalid, tr.arvalid, tr.awburst, tr.arburst, tr.awaddr, tr.araddr, tr.awlen, tr.arlen);
1376. mbxgd.put(tr);
1377. @(drvnext);
1378. @(sconext);
1379. end
1380. ->done;
1381. endtask

1384. endclass

1387. /////////////////////////////////////

1390. class driver;
1392. virtual axi\_if vif;
1394. transaction tr;
1396. event drvnext;
1397. event monnext;
1399. mailbox #(transaction) mbxgd;

1402. function new( mailbox #(transaction) mbxgd );
1403. this.mbxgd = mbxgd;
1404. endfunction
1406. //////////////////Resetting System
1407. task reset();
1409. vif.resetn <= 1'b0;
1411. vif.awvalid <= 1'b0;
1412. vif.awid <= 0;
1413. vif.awlen <= 0;
1414. vif.awsize <= 0;
1415. vif.awaddr <= 0;
1416. vif.awburst <= 0;
1418. vif.wvalid <= 0;
1419. vif.wid <= 0;
1420. vif.wdata <= 0;
1421. vif.wstrb <= 0;
1422. vif.wlast <= 0;
1424. vif.bready <= 0;

1427. vif.arvalid <= 1'b0;
1428. vif.arid <= 0;
1429. vif.arlen <= 0;
1430. vif.arsize <= 0;
1431. vif.araddr <= 0;
1432. vif.arburst <= 0;
1434. repeat(5) @(posedge vif.clk);
1435. vif.resetn <= 1'b1;
1436. $display("[DRV] : RESET DONE");
1437. endtask

1440. ///////////////////////////////write data in Fixed Mode
1441. task fixed\_write(input transaction tr);
1442. int len = 0;
1443. len = tr.awlen + 1; //8
1444. $display("[DRV] : FIXED MODE -> DATA WRITE DONE");
1445. @(posedge vif.clk);
1446. vif.resetn <= 1'b1;
1447. vif.awvalid <= 1'b1;
1448. vif.arvalid <= 1'b0; ////disable read
1449. vif.awid <= tr.id;
1450. vif.awlen <= tr.awlen;
1451. vif.awsize <= 3'b010; ///4 byte
1452. vif.awburst <= 2'b00; //00
1453. vif.wvalid <= 1'b1;
1454. vif.wid <= tr.id;
1455. vif.wstrb <= 4'b1111;
1456. vif.bready <= 1'b1;
1457. vif.awaddr <= tr.awaddr;
1458. vif.wdata <= $urandom\_range(1,100);
1460. @(posedge vif.wready);
1461. @(posedge vif.clk);

1464. for(int i = 1; i< len ; i++) begin /// 1 2 3 4 5 6 7
1465. vif.awaddr <= tr.awaddr;
1466. vif.wdata <= $urandom\_range(1,100);
1467. @(posedge vif.wready);
1468. @(posedge vif.clk);
1469. end
1471. vif.wlast <= 1'b1;
1472. vif.awvalid <= 1'b0;
1473. vif.arvalid <= 1'b0;
1474. vif.wvalid <= 1'b0;
1475. @(posedge vif.clk);
1476. vif.wlast <= 1'b0;
1477. @(negedge vif.bvalid);
1478. ->drvnext;
1479. endtask
1481. ///////////////////////////Write data in Incr Mode //////
1483. task incr\_write(input transaction tr);
1484. int len = 0;
1485. len = tr.awlen + 1; //8
1486. $display("[DRV] : INCR MODE -> DATA WRITE DONE");
1487. @(posedge vif.clk);
1488. vif.resetn <= 1'b1;
1489. vif.arvalid <= 1'b0; ////disable read
1490. vif.awvalid <= 1'b1;
1491. vif.awid <= tr.id;
1492. vif.awlen <= tr.awlen;
1493. vif.awsize <= 3'b010;
1494. vif.awburst <= 2'b01;
1495. vif.wvalid <= 1'b1;
1496. vif.wid <= tr.id;
1497. vif.wstrb <= 4'b1111;
1498. vif.bready <= 1'b1;
1499. vif.awaddr <= tr.awaddr;
1500. vif.wdata <= $urandom\_range(1,100);
1502. @(posedge vif.wready);
1503. @(posedge vif.clk);
1505. for(int i = 1; i< len; i++) begin ///i < 8 1 2 3 4 5 6 7
1506. vif.awaddr <= tr.awaddr + 4\*i;
1507. vif.wdata <= $urandom\_range(1,100);
1508. @(posedge vif.wready);
1509. @(posedge vif.clk);
1510. end
1512. vif.wlast <= 1'b1;
1513. vif.awvalid <= 1'b0;
1514. vif.arvalid <= 1'b0;
1515. vif.wvalid <= 1'b0;
1516. @(posedge vif.clk);
1517. vif.wlast <= 1'b0;
1518. @(negedge vif.bvalid);

1521. ->drvnext;
1522. endtask





1529. //////////////////////////////Write data in Wrap Mode
1531. task wrap\_write(input transaction tr);
1532. int len = 0;
1533. len = tr.awlen + 1; //8
1534. $display("[DRV] : WRAP MODE -> DATA WRITE DONE");
1535. @(posedge vif.clk);
1536. vif.arvalid <= 1'b0; ///disable read
1537. vif.resetn <= 1'b1;
1538. vif.awvalid <= 1'b1;
1539. vif.awid <= tr.id;
1540. vif.awlen <= tr.awlen;
1541. vif.awsize <= 3'b010;
1542. vif.awburst <= 2'b10;
1543. vif.wvalid <= 1'b1;
1544. vif.wid <= tr.id;
1545. vif.wstrb <= 4'b1111;
1546. vif.bready <= 1'b1;

1549. vif.awaddr <= tr.awaddr;
1550. vif.wdata <= $urandom\_range(1,100);
1551. @(posedge vif.wready);
1552. @(posedge vif.clk);
1554. for(int i = 0; i < 7; i++) begin ///0 1 2 3 4 5 6
1555. vif.awaddr <= vif.addr\_wrapwr;
1556. vif.wdata <= $urandom\_range(1,100);
1557. @(posedge vif.wready);
1558. @(posedge vif.clk);
1559. end
1561. vif.wlast <= 1'b1;
1562. @(posedge vif.clk);
1563. vif.wlast <= 1'b0;
1564. vif.awvalid <= 1'b0;
1565. vif.arvalid <= 1'b0;
1566. vif.wvalid <= 1'b0;
1567. @(negedge vif.bvalid);
1569. ->drvnext;
1570. endtask
1572. /////////////////////////////////read fixed mode
1574. task fixed\_read(input transaction tr);
1576. int len = 0;
1577. len = tr.arlen + 1; //8
1578. $display("[DRV] : FIXED MODE -> DATA READ");
1579. @(posedge vif.clk);
1580. vif.awvalid <= 1'b0; /////disable write transaction
1581. vif.resetn <= 1'b1;
1582. vif.arvalid <= 1'b1;
1583. vif.arid <= tr.id;
1584. vif.arlen <= tr.arlen;
1585. vif.arsize <= 3'b010;
1586. vif.arburst <= 2'b00;
1587. vif.rready <= 1'b1;
1589. for(int i = 0; i < len; i++) begin // 0 1 2 3 4 5 6 7
1590. vif.araddr <= tr.araddr;
1591. @(posedge vif.arready);
1592. @(posedge vif.clk);
1593. end
1595. @(negedge vif.rlast);
1596. vif.arvalid <= 1'b0;
1597. vif.rready <= 1'b0;
1599. ->drvnext;


1603. endtask


1607. ///////////////////////////////read incr mode
1609. task incr\_read(input transaction tr);
1611. int len = 0;
1612. len = tr.arlen + 1;
1613. $display("[DRV] : INCR MODE -> DATA READ");
1614. @(posedge vif.clk);
1615. vif.awvalid <= 1'b0; /////disable write transaction
1616. vif.resetn <= 1'b1;
1617. vif.arvalid <= 1'b1;
1618. vif.arid <= tr.id;
1619. vif.arlen <= tr.arlen;
1620. vif.arsize <= 3'b010;
1621. vif.arburst <= 2'b01;
1622. vif.rready <= 1'b1;


1626. for(int i = 0; i< len; i++) begin
1627. vif.araddr <= tr.araddr + 4\*i;
1628. @(posedge vif.arready);
1629. @(posedge vif.clk);
1630. end
1632. @(negedge vif.rlast);
1633. vif.arvalid <= 1'b0;
1634. vif.rready <= 1'b0;
1636. ->drvnext;


1640. endtask


1644. //////////////////////////////////////////wrap mode
1646. task wrap\_read(input transaction tr);
1648. int len = 0;
1649. len = 8;
1650. $display("[DRV] : WRAP MODE -> DATA READ COMPLETE");
1651. @(posedge vif.clk);
1652. vif.awvalid <= 1'b0; /////disable write transaction
1653. vif.resetn <= 1'b1;
1654. vif.arvalid <= 1'b1;
1655. vif.arid <= tr.id;
1656. vif.arlen <= 4'b0111;
1657. vif.arsize <= 3'b010;
1658. vif.arburst <= 2'b10;
1659. vif.rready <= 1'b1;
1660. vif.araddr <= tr.araddr;
1661. @(posedge vif.rvalid);
1662. @(posedge vif.clk);



1667. for(int i = 0; i< 7; i++) begin /// 0123456
1668. vif.araddr <= vif.addr\_wraprd;
1669. @(posedge vif.rvalid);
1670. @(posedge vif.clk);
1671. end
1673. @(negedge vif.rlast);
1674. vif.arvalid <= 1'b0;
1675. vif.rready <= 1'b0;
1677. ->drvnext;
1679. endtask




1685. ///////////////////////////////////////////////////////main task

1688. task run();
1690. forever begin
1692. mbxgd.get(tr);
1693. /////////////////////////write mode check and sig gen
1694. if(tr.awvalid == 1'b1) begin
1695. if(tr.awburst == 2'b00)
1696. begin
1697. fixed\_write(tr);
1698. end
1699. else if (tr.awburst == 2'b01)
1700. begin
1701. incr\_write(tr);
1702. end
1703. else if (tr.awburst == 2'b10)
1704. begin
1705. wrap\_write(tr);
1706. end
1708. end

1711. /////////////////////////////read mode check and sig gen
1713. if(tr.arvalid == 1'b1) begin
1714. if(tr.arburst == 2'b00)
1715. begin
1716. fixed\_read(tr);
1717. end
1718. else if (tr.arburst == 2'b01)
1719. begin
1720. incr\_read(tr);
1721. end
1722. else if (tr.arburst == 2'b10)
1723. begin
1724. wrap\_read(tr);
1725. end
1727. end
1728. end
1729. endtask


1733. endclass
1734. ///////////////////////////////////////////////////////
1736. /\*
1737. module tb;
1739. generator gen;
1740. driver drv;
1741. event next;
1742. event done;
1744. mailbox #(transaction) mbxgd;
1746. axi\_if vif();
1747. axi\_slave dut (vif.clk, vif.resetn, vif.awvalid, vif.awready, vif.awid, vif.awlen, vif.awsize, vif.awaddr, vif.awburst, vif.wvalid, vif.wready, vif.wid, vif.wdata, vif.wstrb, vif.wlast, vif.bready, vif.bvalid, vif.bid, vif.bresp , vif.arready, vif.arid, vif.araddr, vif.arlen, vif.arsize, vif.arburst, vif.arvalid, vif.rid, vif.rdata, vif.rresp,vif.rlast, vif.rvalid, vif.rready);
1749. initial begin
1750. vif.clk <= 0;
1751. end
1753. always #5 vif.clk <= ~vif.clk;
1755. initial begin
1757. mbxgd = new();
1758. gen = new(mbxgd);
1759. drv = new(mbxgd);
1760. gen.count = 5;
1761. drv.vif = vif;
1763. drv.drvnext = next;
1764. gen.drvnext = next;
1766. end
1768. initial begin
1769. drv.reset();
1770. fork
1771. gen.run();
1772. drv.run();
1773. join\_none
1774. wait(gen.done.triggered);
1775. $finish();
1776. end
1778. initial begin
1779. $dumpfile("dump.vcd");
1780. $dumpvars;
1781. end
1783. assign vif.addr\_wrapwr = dut.nextaddr;
1784. assign vif.addr\_wraprd = dut.rdretaddr;

1787. endmodule
1789. \*/

1792. ///////////////////////////////////////////////////

1795. class monitor;
1797. virtual axi\_if vif;

1800. transaction tr;
1802. event sconext;
1803. int len = 0;
1805. mailbox #(transaction) mbxms;




1811. function new( mailbox #(transaction) mbxms );
1812. this.mbxms = mbxms;
1813. endfunction

1816. task run();
1818. tr = new();
1820. forever
1821. begin

1824. @(posedge vif.clk);
1826. //////////////////////////write logic
1828. if(vif.awvalid == 1'b1) begin
1829. len = vif.awlen + 1;
1830. tr.awvalid = vif.awvalid;
1831. tr.arvalid = vif.arvalid;

1834. for(int i = 0; i< len; i++) begin
1835. @(posedge vif.wready);
1836. @(posedge vif.clk);
1837. tr.awaddr = vif.awaddr;
1838. tr.wdata = vif.wdata;
1839. tr.awburst = vif.awburst;
1840. mbxms.put(tr);
1841. $display("[MON] : ADDR : %0x DATA : %0x BURST TYPE : %0d",tr.awaddr, tr.wdata, tr.awburst);
1842. end
1844. @(posedge vif.clk);
1845. @(negedge vif.bvalid);
1846. @(posedge vif.clk);
1847. $display("[MON] : Transaction Complete");
1848. end
1850. /////////////////////read logic
1852. if(vif.arvalid == 1'b1)
1853. begin
1854. len = vif.arlen + 1;
1855. tr.awvalid = vif.awvalid;
1856. tr.arvalid = vif.arvalid;

1859. for(int i = 0; i< len; i++) begin
1860. @(posedge vif.rvalid);
1861. @(posedge vif.clk);
1862. tr.rdata = vif.rdata;
1863. tr.arburst = vif.arburst;
1864. tr.araddr = vif.araddr;
1865. mbxms.put(tr);
1866. $display("[MON] : ADDR : %0x DATA : %0x BURST TYPE : %0d",tr.araddr, tr.rdata, tr.arburst);
1867. end
1869. @(posedge vif.clk);
1870. @(negedge vif.rlast);
1871. @(posedge vif.clk);
1872. $display("[MON] : Transaction Complete");
1873. end
1874. ->sconext;
1875. end
1876. endtask


1880. endclass
1882. ///////////////////////////////////////

1885. class scoreboard;
1887. transaction tr;

1890. mailbox #(transaction) mbxms;


1894. bit [31:0] temp;
1896. bit [7:0] data[128] = '{default:0};
1898. int count = 0;
1899. int len = 0;


1903. function new( mailbox #(transaction) mbxms );
1904. this.mbxms = mbxms;
1905. endfunction

1908. task run();
1910. forever
1911. begin

1914. mbxms.get(tr);

1917. if(tr.awvalid == 1'b1) begin
1918. data[tr.awaddr] = tr.wdata[7:0];
1919. data[tr.awaddr + 1] = tr.wdata[15:8];
1920. data[tr.awaddr + 2] = tr.wdata[23:16];
1921. data[tr.awaddr + 3] = tr.wdata[31:24];
1923. $display("[SCO] : DATA STORED ADDR :%0x and DATA :%0x", tr.awaddr, tr.wdata[7:0]);
1924. end
1926. if(tr.arvalid == 1'b1) begin
1927. temp = {data[tr.araddr + 3],data[tr.araddr + 2],data[tr.araddr + 1],data[tr.araddr] };
1929. $display("[SCO] : DATA READ ADDR :%0x and DATA :%0x MEM :%0x", tr.araddr, tr.rdata, temp);
1930. if(tr.rdata == 32'hc0c0c0c)
1931. begin
1932. $display("[SCO] : DATA MATCHED : EMPTY LOCATION");
1933. end
1934. else if (tr.rdata == temp)
1935. begin
1936. $display("[SCO] : DATA MATCHED");
1937. end
1938. else
1939. begin
1940. $display("[SCO] : DATA MISMATCHED");
1941. end
1943. end

1946. end
1947. endtask

1950. endclass

1953. ///////////////////////////////////////////////////
1955. module tb;
1957. monitor mon;
1958. generator gen;
1959. driver drv;
1960. scoreboard sco;

1963. event nextgd;
1964. event nextgm;


1968. mailbox #(transaction) mbxgd, mbxms;
1970. axi\_if vif();
1971. axi\_slave dut (vif.clk, vif.resetn, vif.awvalid, vif.awready, vif.awid, vif.awlen, vif.awsize, vif.awaddr, vif.awburst, vif.wvalid, vif.wready, vif.wid, vif.wdata, vif.wstrb, vif.wlast, vif.bready, vif.bvalid, vif.bid, vif.bresp , vif.arready, vif.arid, vif.araddr, vif.arlen, vif.arsize, vif.arburst, vif.arvalid, vif.rid, vif.rdata, vif.rresp,vif.rlast, vif.rvalid, vif.rready);
1973. initial begin
1974. vif.clk <= 0;
1975. end
1977. always #5 vif.clk <= ~vif.clk;
1979. initial begin
1981. mbxgd = new();
1982. mbxms = new();
1983. gen = new(mbxgd);
1984. drv = new(mbxgd);
1986. mon = new(mbxms);
1987. sco = new(mbxms);
1989. gen.count = 4;
1990. drv.vif = vif;
1991. mon.vif = vif;
1993. drv.drvnext = nextgd;
1994. gen.drvnext = nextgd;
1996. gen.sconext = nextgm;
1997. mon.sconext = nextgm;
1999. end
2001. initial begin
2002. drv.reset();
2003. fork
2004. gen.run();
2005. drv.run();
2006. mon.run();
2007. sco.run();
2008. join\_any
2009. wait(gen.done.triggered);
2010. $finish;
2011. end
2013. initial begin
2014. $dumpfile("dump.vcd");
2015. $dumpvars;
2016. end
2018. assign vif.addr\_wrapwr = dut.retaddr;
2019. assign vif.addr\_wraprd = dut.rdretaddr;
2021. endmodule

AHB MEMORY:

1. `timescale 1ns / 1ps
3. `define NON\_SEQ 2'd0
4. `define SEQ 2'd1
5. `define BUSY 2'd2
6. `define IDLE 2'd3
8. `define OKAY 2'b00
9. `define ERROR 2'b01
10. `define RETRY 2'b10
11. `define SPLIT 2'b11

14. module ahb\_slave(
15. input clk,
16. input [31:0] hwdata,
17. input [31:0] haddr,
18. input [2:0] hsize,
19. input [2:0] hburst,
20. input hresetn, hsel, hwrite,
21. input [1:0] htrans,
22. output reg [1:0] hresp,
23. output reg hready,
24. output reg [31:0] hrdata
25. );


29. reg [7:0] mem[256] = '{default : 12};
31. ///////////////////////////////////////////////////////////////////////////////////////////
32. function bit[31:0] single\_tr (input bit [31:0] addr, input bit [2:0] hsize);
33. unique case(hsize)
34. 3'b000: begin
35. mem[addr] = hwdata[7:0];
36. end
38. 3'b001: begin
39. mem[addr] = hwdata[7:0];
40. mem[addr + 1] = hwdata[15:8];
41. end
43. 3'b010: begin
44. mem[addr] = hwdata[7:0];
45. mem[addr + 1] = hwdata[15:8];
46. mem[addr + 2] = hwdata[23:16];
47. mem[addr + 3] = hwdata[31:24];
48. end
49. endcase
50. return addr;
52. endfunction

55. ///////////////////////////////////////////////////////////////////////////////////
56. function bit[31:0] unincr\_wr (input bit [31:0] addr, input bit [2:0] hsize);
57. bit [31:0] raddr;
58. unique case(hsize)
60. 3'b000: begin
61. mem[addr] = hwdata[7:0];
62. raddr = addr + 1;
63. end
65. 3'b001: begin
66. mem[addr] = hwdata[7:0];
67. mem[addr + 1] = hwdata[15:8];
68. raddr = addr + 2;
69. end
71. 3'b010: begin
72. mem[addr] = hwdata[7:0];
73. mem[addr + 1] = hwdata[15:8];
74. mem[addr + 2] = hwdata[23:16];
75. mem[addr + 3] = hwdata[31:24];
76. raddr = addr + 4;
77. end
78. endcase
80. return raddr;
82. endfunction





89. //////////////////////////////////////////////////////////////////////////////
90. function bit[7:0] boundary(input bit [2:0] hburst, input [2:0] hsize);
91. bit [7:0] temp;
92. unique case(hsize)
93. 3'b000: begin
94. unique case (hburst)
95. 3'b010 : temp = 4 \* 1;
97. 3'b100 : temp = 8 \* 1;
99. 3'b110 : temp = 16 \* 1;
100. endcase
101. end
103. 3'b001 : begin
104. unique case (hburst)
105. 3'b010 : temp = 4 \* 2;
107. 3'b100 : temp = 8 \* 2;
109. 3'b110 : temp = 16 \* 2;
110. endcase
111. end

114. 3'b010 : begin
115. unique case (hburst)
116. 3'b010 : temp = 4 \* 4;
118. 3'b100 : temp = 8 \* 4;
120. 3'b110 : temp = 16 \* 4;
121. endcase
122. end
124. endcase
126. return temp;
128. endfunction

131. function bit[31:0] wrap\_wr (input bit [31:0] addr, input bit [7:0] boundary, input [2:0] hsize);
132. bit [31:0] addr1, addr2, addr3,addr4;
134. unique case(hsize)
135. 3'b000: begin
136. mem[addr] = hwdata[7:0];
138. if((addr + 1) % boundary == 0)
139. addr1 = (addr + 1) - boundary;
140. else
141. addr1 = (addr + 1);
143. return addr1;
144. end
146. 3'b001: begin
147. mem[addr] = hwdata[7:0];
149. if((addr + 1) % boundary == 0)
150. addr1 = (addr + 1) - boundary;
151. else
152. addr1 = (addr + 1);
154. mem[addr1] = hwdata[15:8];
156. if((addr1 + 1) % boundary == 0)
157. addr2 = (addr1 + 1) - boundary;
158. else
159. addr2 = (addr1 + 1);
161. return addr2;
163. end
165. 3'b010: begin
167. mem[addr] = hwdata[7:0];
169. if((addr + 1) % boundary == 0)
170. addr1 = (addr + 1) - boundary;
171. else
172. addr1 = (addr + 1);
174. mem[addr1] = hwdata[15:8];
176. if((addr1 + 1) % boundary == 0)
177. addr2 = (addr1 + 1) - boundary;
178. else
179. addr2 = (addr1 + 1);
181. mem[addr2] = hwdata[23:16];
183. if((addr2 + 1) % boundary == 0)
184. addr3 = (addr2 + 1) - boundary;
185. else
186. addr3 = (addr2 + 1);
188. mem[addr3] = hwdata[31:24];
190. if((addr3 + 1) % boundary == 0)
191. addr4 = (addr3 + 1) - boundary;
192. else
193. addr4 = (addr3 + 1);
195. return addr4;
196. end

199. endcase
201. endfunction
202. ////////////////////////////////////////////////////////////////////////////////////////////

205. function bit[31:0] incr\_wr(input bit [31:0] addr, input bit [2:0] hsize);
207. bit [31:0] raddr;
209. unique case(hsize)
211. 3'b000: begin
212. mem[addr] = hwdata[7:0];
213. raddr = addr + 1;
214. end
216. 3'b001: begin
217. mem[addr] = hwdata[7:0];
218. mem[addr + 1] = hwdata[15:8];
219. raddr = addr + 2;
220. end
222. 3'b010: begin
223. mem[addr] = hwdata[7:0];
224. mem[addr + 1] = hwdata[15:8];
225. mem[addr + 2] = hwdata[23:16];
226. mem[addr + 3] = hwdata[31:24];
227. raddr = addr + 4;
228. end
229. endcase
231. return raddr;
233. endfunction
235. ////////////////////////////////////////////////////////////////////////////////
236. ////////////////////////////single transfer read
238. function bit[31:0] single\_tr\_rd (input bit [31:0] addr, input bit [2:0] hsize);
239. unique case(hsize)
240. 3'b000: begin
241. hrdata[7:0] = mem[addr];
242. end
244. 3'b001: begin
245. hrdata[7:0] = mem[addr];
246. hrdata[15:8] = mem[addr + 1];
247. end
249. 3'b010: begin
250. hrdata[7:0] = mem[addr];
251. hrdata[15:8] = mem[addr + 1];
252. hrdata[23:16] = mem[addr + 2];
253. hrdata[31:24] = mem[addr + 3];
254. end
255. endcase
256. return addr;
258. endfunction
259. ///////////////////////////////////////////////////////
261. //////////////////////////////////////////////////////////////////////////////////
262. /////////////////////////////Read for unspec length
263. function bit[31:0] unincr\_rd (input bit [31:0] addr, input bit [2:0] hsize);
264. bit [31:0] raddr;
265. unique case(hsize)
267. 3'b000: begin
268. hrdata[7:0] = mem[addr];
269. raddr = addr + 1;
270. end
272. 3'b001: begin
273. hrdata[7:0] = mem[addr];
274. hrdata[15:8] = mem[addr + 1];
275. raddr = addr + 2;
276. end
278. 3'b010: begin
279. hrdata[7:0] = mem[addr];
280. hrdata[15:8] = mem[addr + 1];
281. hrdata[23:16] = mem[addr + 2];
282. hrdata[31:24] = mem[addr + 3];
283. raddr = addr + 4;
284. end
285. endcase
287. return raddr;
289. endfunction
291. //////////////////////////////////////////////////////
292. /////////////////////////////wrapping read////////////////////////////////////////////////////
293. function bit[31:0] wrap\_rd (input bit [31:0] addr, input bit [7:0] boundary, input [2:0] hsize);
294. bit [31:0] addr1, addr2, addr3,addr4;
296. unique case(hsize)
297. 3'b000: begin
298. hrdata[7:0] = mem[addr];
300. if((addr + 1) % boundary == 0)
301. addr1 = (addr + 1) - boundary;
302. else
303. addr1 = (addr + 1);
305. return addr1;
306. end
308. 3'b001: begin
309. hrdata[7:0] = mem[addr];
311. if((addr + 1) % boundary == 0)
312. addr1 = (addr + 1) - boundary;
313. else
314. addr1 = (addr + 1);
316. hrdata[15:8] = mem[addr1];
318. if((addr1 + 1) % boundary == 0)
319. addr2 = (addr1 + 1) - boundary;
320. else
321. addr2 = (addr1 + 1);
323. return addr2;
325. end
327. 3'b010: begin
329. hrdata[7:0] = mem[addr];
331. if((addr + 1) % boundary == 0)
332. addr1 = (addr + 1) - boundary;
333. else
334. addr1 = (addr + 1);
336. hrdata[15:8] = mem[addr1];
338. if((addr1 + 1) % boundary == 0)
339. addr2 = (addr1 + 1) - boundary;
340. else
341. addr2 = (addr1 + 1);
343. hrdata[23:16] = mem[addr2];
345. if((addr2 + 1) % boundary == 0)
346. addr3 = (addr2 + 1) - boundary;
347. else
348. addr3 = (addr2 + 1);
350. hrdata[31:24] = mem[addr3];
352. if((addr3 + 1) % boundary == 0)
353. addr4 = (addr3 + 1) - boundary;
354. else
355. addr4 = (addr3 + 1);
357. return addr4;
358. end

361. endcase
363. endfunction
365. /////////////////////////////////////////////////////////////////////////////////////////
366. function bit[31:0] incr\_rd(input bit [31:0] addr, input bit [2:0] hsize);
368. bit [31:0] raddr;
370. unique case(hsize)
372. 3'b000: begin
373. hrdata[7:0] = mem[addr];
374. raddr = addr + 1;
375. end
377. 3'b001: begin
378. hrdata[7:0] = mem[addr];
379. hrdata[15:8] = mem[addr + 1];
380. raddr = addr + 2;
381. end
383. 3'b010: begin
384. hrdata[7:0] = mem[addr];
385. hrdata[15:8] = mem[addr + 1];
386. hrdata[23:16] = mem[addr + 2];
387. hrdata[31:24] = mem[addr + 3];
388. raddr = addr + 4;
389. end
390. endcase
392. return raddr;
394. endfunction

397. //////////////////////////////////////////////////////////////////////////////////
399. typedef enum {idle = 0, check\_mode = 1, write = 2, read = 3, addr\_decode = 4} state\_type;
400. state\_type state, next\_state;
402. ///////////////////////////////////////////////////////////////////////////////////
404. always\_ff@(posedge clk)
405. begin
406. if(!hresetn)
407. state <= idle;
408. else
409. state <= next\_state;
410. end
411. ///////////////////////////////////////////////////////////////////////////////////


415. integer len\_count = 0;
416. reg first = 0;
417. reg [31:0] retaddr;
418. reg [31:0] next\_addr;
419. reg [7:0] wboundary;



424. /////////////////////////////////////////////////////////////////////////////////
425. always\_comb
426. begin
427. case(state)
429. idle :
430. begin
431. next\_state = check\_mode;
432. hready = 1'b0;
433. len\_count = 0;
434. first = 0;
435. hresp = `OKAY;
436. end
438. check\_mode :
439. begin
440. hready = 1'b0;
441. if(hresetn && hsel && hwrite)
442. begin
443. if(haddr < 256)
444. begin
445. next\_state = addr\_decode;
446. end
447. else
448. begin
449. next\_state = idle;
450. hresp = `ERROR;
451. end
452. end
453. else if (hresetn && hsel && !hwrite)
454. begin
455. if(haddr < 256)
456. begin
457. next\_state = addr\_decode;
458. end
459. else
460. begin
461. next\_state = idle;
462. hresp = `ERROR;
463. end
464. end
465. else
466. begin
467. next\_state = idle;
468. end
469. end
471. addr\_decode:
472. begin
473. if(htrans == `NON\_SEQ)
474. begin
475. next\_addr = haddr;
477. if(hwrite)
478. next\_state = write;
479. else
480. next\_state = read;

483. end
484. else if (htrans == `SEQ)
485. begin
486. next\_addr = retaddr;
488. if(hwrite)
489. next\_state = write;
490. else
491. next\_state = read;
493. end

496. end

499. write:
500. begin
501. case(hburst)
503. //////////////////////////Single Write at HADDR
504. 3'b000: begin ////single transfer
505. retaddr = single\_tr(next\_addr,hsize);
506. hready = 1'b1;
507. next\_state = idle;
508. hresp = `OKAY;
509. end
511. /////////////////////INCREMENT for UNSPECIFIED LENGTH
513. 3'b001:
514. begin ////incr mode
516. hready = 1'b1;
517. retaddr = unincr\_wr(next\_addr, hsize);
518. hresp = `OKAY;

521. if(len\_count < 32)
522. begin
523. len\_count = len\_count + 1;
524. next\_state = check\_mode;
525. end
526. else
527. begin
528. len\_count = 0;
529. next\_state = idle;
530. end
532. end
533. ////////////////////////////4 beat wrapping

536. 3'b010:
537. begin
539. hready = 1'b1;
540. wboundary = boundary(hburst, hsize);
541. retaddr = wrap\_wr(next\_addr, wboundary, hsize);
542. hresp = `OKAY;

545. if(len\_count <= 2) // 0 1 2 3
546. begin
547. len\_count = len\_count + 1;
548. next\_state = check\_mode;
549. end
550. else
551. begin
552. next\_state = idle;
553. len\_count = 0;
554. end
555. end
557. ///////////////////////////////4 beat Incrementing
559. 3'b011:
560. begin
562. hready = 1'b1;
563. retaddr = incr\_wr(next\_addr, hsize);
564. hresp = `OKAY;
566. if(len\_count <= 2)
567. begin
568. len\_count = len\_count + 1;
569. next\_state = check\_mode;
570. end
571. else
572. begin
573. next\_state = idle;
574. len\_count = 0;
575. first = 0;
576. end

579. end
581. ////////////////////////////////////////////////8 beat wrapping

584. 3'b100:
585. begin
587. hready = 1'b1;
588. wboundary = boundary(hburst, hsize);
589. retaddr = wrap\_wr(next\_addr, wboundary, hsize);
590. hresp = `OKAY;
592. if(len\_count <= 6)
593. begin
594. len\_count = len\_count + 1;
595. next\_state = check\_mode;
596. end
597. else
598. begin
599. next\_state = idle;
600. len\_count = 0;
601. end
602. end
604. ////////////////////////////////////////////////////////////////////////////////////////////////
605. /////////////////////////////////////////8 beat Incrementing
606. 3'b101:
607. begin
608. hready = 1'b1;
609. retaddr = incr\_wr(next\_addr, hsize);
610. hresp = `OKAY;
612. if(len\_count <= 6)
613. begin
614. len\_count = len\_count + 1;
615. next\_state = check\_mode;
616. end
617. else
618. begin
619. next\_state = idle;
620. len\_count = 0;
621. end
623. end
625. /////////////////////////////////////////////////////////////////////////////////////////////
626. /////////////////////////////////////////////////16 beat wrapping
628. 3'b110:
629. begin
631. hready = 1'b1;
632. wboundary = boundary(hburst, hsize);
633. retaddr = wrap\_wr(next\_addr, wboundary, hsize);
634. hresp = `OKAY;
636. if(len\_count <= 14)
637. begin
638. len\_count = len\_count + 1;
639. next\_state = check\_mode;
640. end
641. else
642. begin
643. next\_state = idle;
644. len\_count = 0;
645. end
646. end
648. /////////////////////////////////////////////////////////////////////////////////
649. //////////////////////16 beat incr
650. 3'b111:
651. begin
652. hready = 1'b1;
653. retaddr = incr\_wr(next\_addr, hsize);
654. hresp = `OKAY;
656. if(len\_count <= 14)
657. begin
658. len\_count = len\_count + 1;
659. next\_state = check\_mode;
660. end
661. else
662. begin
663. next\_state = idle;
664. len\_count = 0;
665. end
667. end
669. endcase
670. end
672. read : begin
674. case(hburst)
676. //////////////////////////Single Write at HADDR
677. 3'b000: begin ////single transfer
678. retaddr = single\_tr\_rd(haddr,hsize);
679. hready = 1'b1;
680. next\_state = idle;
681. hresp = `OKAY;
682. end
684. /////////////////////INCREMENT for UNSPECIFIED LENGTH
686. 3'b001:
687. begin ////incr mode
688. hready = 1'b1;
689. retaddr = unincr\_rd(next\_addr, hsize);
690. hresp = `OKAY;
692. if( len\_count < 32)
693. begin
694. len\_count = len\_count + 1;
695. next\_state = check\_mode;
696. end
697. else
698. begin
699. len\_count = 0;
700. next\_state = idle;
701. end
702. end
703. ////////////////////////////4 beat wrapping

706. 3'b010:
707. begin
709. hready = 1'b1;
710. wboundary = boundary(hburst, hsize);
711. retaddr = wrap\_rd(next\_addr, wboundary, hsize);
712. hresp = `OKAY;
714. if(len\_count <= 2)
715. begin
716. len\_count = len\_count + 1;
717. next\_state = check\_mode;
718. end
719. else
720. begin
721. next\_state = idle;
722. len\_count = 0;
723. end

726. end
728. ///////////////////////////////4 beat Incrementing read
730. 3'b011:
731. begin
732. hready = 1'b1;
733. retaddr = incr\_rd(next\_addr, hsize);
734. hresp = `OKAY;
736. if(len\_count <= 2)
737. begin
738. len\_count = len\_count + 1;
739. next\_state = check\_mode;
740. end
741. else
742. begin
743. next\_state = idle;
744. len\_count = 0;
745. end
747. end
749. ////////////////////////////////////////////////8 beat wrapping

752. 3'b100:
753. begin
755. hready = 1'b1;
756. wboundary = boundary(hburst, hsize);
757. retaddr = wrap\_rd(next\_addr, wboundary, hsize);
758. hresp = `OKAY;
760. if(len\_count <= 6)
761. begin
762. len\_count = len\_count + 1;
763. next\_state = check\_mode;
764. end
765. else
766. begin
767. next\_state = idle;
768. len\_count = 0;
769. end

772. end
774. ////////////////////////////////////////////////////////////////////////////////////////////////
775. /////////////////////////////////////////8 beat Incrementing
776. 3'b101:
777. begin
778. hready = 1'b1;
779. retaddr = incr\_rd(next\_addr, hsize);
780. hresp = `OKAY;
782. if(len\_count <= 6)
783. begin
784. len\_count = len\_count + 1;
785. next\_state = check\_mode;
786. end
787. else
788. begin
789. next\_state = idle;
790. len\_count = 0;
791. end
793. end
795. /////////////////////////////////////////////////////////////////////////////////////////////
796. /////////////////////////////////////////////////16 beat wrapping
798. 3'b110:
799. begin
801. hready = 1'b1;
802. wboundary = boundary(hburst, hsize);
803. retaddr = wrap\_rd(next\_addr, wboundary, hsize);
804. hresp = `OKAY;
806. if(len\_count <= 14)
807. begin
808. len\_count = len\_count + 1;
809. next\_state = check\_mode;
810. end
811. else
812. begin
813. next\_state = idle;
814. len\_count = 0;
815. end
816. end
818. /////////////////////////////////////////////////////////////////////////////////
819. //////////////////////16 beat incr
820. 3'b111:
821. begin
822. hready = 1'b1;
823. retaddr = incr\_rd(next\_addr, hsize);
824. hresp = `OKAY;
826. if(len\_count <= 14)
827. begin
828. len\_count = len\_count + 1;
829. next\_state = check\_mode;
830. end
831. else
832. begin
833. next\_state = idle;
834. len\_count = 0;
835. end
837. end
839. endcase
841. end
843. endcase
845. end
846. endmodule

TB:

1. `timescale 1ns / 1ps
2. //////////////////////////////////////////////////////////////////////////////////
3. // Company:
4. // Engineer:
5. //
6. // Create Date: 28.03.2022 21:55:11
7. // Design Name:
8. // Module Name: tb
9. // Project Name:
10. // Target Devices:
11. // Tool Versions:
12. // Description:
13. //
14. // Dependencies:
15. //
16. // Revision:
17. // Revision 0.01 - File Created
18. // Additional Comments:
19. //
20. //////////////////////////////////////////////////////////////////////////////////
22. /\*
23. module tb();
24. reg clk = 0;
25. reg [31:0] hwdata;
26. reg [31:0] haddr;
27. reg [2:0] hsize;
28. reg [2:0] hburst;
29. reg hresetn, hsel;
30. reg hwrite;
31. reg [1:0] htrans;
32. wire [1:0] hresp;
33. wire hready;
34. wire [31:0] hrdata;
36. always #10 clk =~clk;
38. ahb\_slave dut (clk,hwdata, haddr, hsize, hburst,hresetn, hsel, hwrite, htrans, hresp, hready,hrdata);
40. initial begin
42. hresetn = 1'b1;
43. hsel = 1;
44. hwrite = 1;
45. htrans = 0;
46. haddr = 12;
47. hwdata = 23;
48. hsize = 2;
49. hburst = 7;
50. @(posedge hready);
51. @(posedge clk);
52. hwrite = 0;
54. end
56. initial begin
57. #100;
58. $finish();
59. end
61. endmodule
63. \*/

66. interface ahb\_if;
68. logic clk;
69. logic [31:0] hwdata;
70. logic [31:0] haddr;
71. logic [2:0] hsize;
72. logic [2:0] hburst;
73. logic hresetn, hsel, hwrite;
74. logic [1:0] htrans;
75. logic [1:0] hresp;
76. logic hready;
77. logic [31:0] hrdata;
79. logic [31:0] next\_addr;

82. endinterface
83. ///////////////////////////////////////////////

86. class transaction;
88. rand bit [4:0] ulen;
89. rand bit [31:0] hwdata;
90. rand bit [31:0] haddr;
91. rand bit [2:0] hsize;
92. rand bit [2:0] hburst;
93. bit hresetn;
94. rand bit hwrite;
95. bit [1:0] htrans;
96. bit [1:0] hresp;
97. bit hready;
98. bit [31:0] hrdata;
100. constraint write\_c {
101. hwrite dist {1 :/ 1, 0:/ 1};
102. }
104. constraint ulen\_c {
105. ulen == 5;
106. }

109. constraint burst\_c {
110. hburst == 6;
111. }
113. constraint addr\_c {
114. haddr == 5;
115. }
117. function transaction copy();
118. copy = new();
119. copy.hwdata = this.hwdata;
120. copy.haddr = this.haddr;
121. copy.hsize = this.hsize;
122. copy.hburst = this.hburst;
123. copy.hwrite = this.hwrite;
124. copy.htrans = this.htrans;
125. copy.hresp = this.hresp;
126. copy.hready = this.hready;
127. copy.hrdata = this.hrdata;
128. copy.ulen = this.ulen;
129. endfunction
131. endclass

134. //////////////////////////////////////////////////////
136. class generator;
138. transaction tr;
140. mailbox #(transaction) mbxgd;
142. mailbox #(bit [4:0]) mbxgm;

145. event done;
146. event drvnext;
147. event sconext;
149. int count = 0;

152. function new( mailbox #(transaction) mbxgd, mailbox #(bit[4:0]) mbxgm);
153. this.mbxgd = mbxgd;
154. this.mbxgm = mbxgm;
155. tr =new();
156. endfunction


160. task run();
162. repeat(count) begin
163. assert(tr.randomize) else $error("Randomization Failed");
164. $display("[GEN] : DATA SENT TO DRV");
165. mbxgd.put(tr.copy);
166. mbxgm.put(tr.ulen);
167. @(drvnext);
168. @(sconext);
169. end
171. ->done;
172. endtask

175. endclass
176. ///////////////////////////////////////////////////////

179. class driver;
181. virtual ahb\_if vif;
183. transaction tr;
185. event drvnext;
187. mailbox #(transaction) mbxgd;

190. function new( mailbox #(transaction) mbxgd );
191. this.mbxgd = mbxgd;
192. endfunction
194. task reset();
195. vif.hresetn <= 1'b0;
196. vif.hwdata <= 0;
197. vif.haddr <= 0;
198. vif.hsize <= 0;
199. vif.hwrite <= 0;
200. vif.hsel <= 0;
201. vif.htrans <= 0;
202. repeat(10) @(posedge vif.clk);
203. vif.hresetn <= 1'b1;
204. $display("[DRV] : RESET DONE");
205. endtask
207. /////////////////////////////single transfer write
209. task single\_tr\_wr();
211. @(posedge vif.clk);
213. vif.hresetn <= 1'b1;
215. vif.hburst <= 3'b000;
217. vif.hwrite <= 1'b1;
218. vif.hsel <= 1'b1;
220. vif.hwdata <= $urandom\_range(1,50);
221. vif.haddr <= tr.haddr;
222. vif.hsize <= 3'b010; //////write 4 byte
224. vif.htrans <= 2'b00;
226. @(posedge vif.hready);
227. @(posedge vif.clk);
228. ->drvnext;
229. $display("[DRV] : SINGLE TRANSFER WRITE ADDR : %0d DATA : %0d", tr.haddr, vif.hwdata);

232. endtask
234. ///////////////////////////////Single transfer read
235. task single\_tr\_rd();
237. @(posedge vif.clk);
239. vif.hresetn <= 1'b1;
241. vif.hburst <= 3'b000;
243. vif.hwrite <= 1'b0;
244. vif.hsel <= 1'b1;
246. vif.hwdata <= 0;
247. vif.haddr <= tr.haddr;
248. vif.hsize <= 3'b010; //////write 4 byte
250. vif.htrans <= 2'b00;
252. @(posedge vif.hready);
253. @(posedge vif.clk);
254. ->drvnext;
255. $display("[DRV] : SINGLE READ TRANSFER ADDR : %0d DATA : %0d", tr.haddr, vif.hwdata);
257. endtask

260. ///////////////////////// unspec length
262. task unspec\_len\_wr();
263. @(posedge vif.clk);
265. vif.hresetn <= 1'b1;
267. vif.hburst <= 3'b001;
269. vif.hwrite <= 1'b1;
270. vif.hsel <= 1'b1;
272. vif.hwdata <= $urandom\_range(1,50);
273. vif.haddr <= tr.haddr;
274. vif.hsize <= 3'b010; //////write 4 byte
276. vif.htrans <= 2'b00; ///non seq


280. @(posedge vif.hready);
281. @(posedge vif.clk);
282. $display("[DRV] : UNSPEC LEN TRANSFER DATA : %0d ADDR : %0d", vif.hwdata, vif.haddr);
284. repeat(tr.ulen - 1) begin
285. vif.hwdata <= $urandom\_range(1,50);
286. vif.htrans <= 2'b01; ///seq
287. @(posedge vif.hready);
288. @(posedge vif.clk);
289. $display("[DRV] : UNSPEC LEN TRANSFER ADDR : %0d DATA : %0d", vif.hwdata, vif.haddr);
290. end
292. ->drvnext;
293. endtask
295. ////////////////////////////////////////////unspec len read
296. task unspec\_len\_rd();
297. @(posedge vif.clk);
299. vif.hresetn <= 1'b1;
301. vif.hburst <= 3'b001;
303. vif.hwrite <= 1'b0;
304. vif.hsel <= 1'b1;
306. vif.hwdata <= 0;
307. vif.haddr <= tr.haddr;
308. vif.hsize <= 3'b010; //////write 4 byte
310. vif.htrans <= 2'b00; ///non seq


314. @(posedge vif.hready);
315. @(posedge vif.clk);
316. $display("[DRV] : UNSPEC LEN TRANSFER ADDR : %0d DATA : %0d", vif.hwdata, vif.haddr);
318. repeat(tr.ulen - 1) begin
319. vif.hwdata <= 0;
320. vif.htrans <= 2'b01; ///seq
322. @(posedge vif.hready);
323. @(posedge vif.clk);
324. $display("[DRV] : UNSPEC LEN TRANSFER ADDR : %0d DATA : %0d", vif.hwdata, vif.haddr);
325. end
327. ->drvnext;
328. endtask


332. ////////////////////////////////////////////////////////////
333. ////////////////////////Increment 4 beats
335. task incr4\_wr ();
336. $display("[DRV] : INCR4 WRITE");
337. @(posedge vif.clk);
338. vif.hresetn <= 1'b1;
340. vif.hburst <= 3'b011;
342. vif.hwrite <= 1'b1;
343. vif.hsel <= 1'b1;
345. vif.hwdata <= $urandom\_range(1,50);
346. vif.haddr <= tr.haddr;
347. vif.hsize <= 3'b010; //////write 4 byte
349. vif.htrans <= 2'b00;
350. //$display("[DRV] : INCR4 TRANSFER ADDR : %0d DATA : %0d", vif.hwdata, vif.haddr);
351. @(posedge vif.hready);
352. @(posedge vif.clk);
354. repeat(3) begin
355. vif.hwdata <= $urandom\_range(1,50);
356. vif.htrans <= 2'b01;
357. // $display("[DRV] : INCR4 TRANSFER ADDR : %0d DATA : %0d", vif.hwdata, vif.haddr);
358. @(posedge vif.hready);
359. @(posedge vif.clk);
360. end
362. ->drvnext;
363. endtask
365. ///////////////////////////////////////////////////////////////
366. ///////////////////////////INCR4 read
367. task incr4\_rd ();
368. $display("[DRV] : INCR4 READ");
369. @(posedge vif.clk);
370. vif.hresetn <= 1'b1;
372. vif.hburst <= 3'b011;
374. vif.hwrite <= 1'b0;
375. vif.hsel <= 1'b1;
377. vif.hwdata <= 0;
378. vif.haddr <= tr.haddr;
379. vif.hsize <= 3'b010; //////write 4 byte
381. vif.htrans <= 2'b00;
382. // $display("[DRV] : INCR4 TRANSFER ADDR : %0d DATA : %0d", vif.hwdata, vif.haddr);
383. @(posedge vif.hready);
384. @(posedge vif.clk);
386. repeat(3) begin
387. vif.hwdata <= 0;
388. vif.htrans <= 2'b01;
389. // $display("[DRV] : INCR4 TRANSFER ADDR : %0d DATA : %0d", vif.hwdata, vif.haddr);
390. @(posedge vif.hready);
391. @(posedge vif.clk);
392. end
393. ->drvnext;
394. endtask

397. /////////////////////////Increment 8 beats
399. task incr8\_wr();
400. @(posedge vif.clk);
402. vif.hresetn <= 1'b1;
404. vif.hburst <= 3'b101;
406. vif.hwrite <= 1'b1;
407. vif.hsel <= 1'b1;
409. vif.hwdata <= $urandom\_range(1,50);
410. vif.haddr <= tr.haddr;
411. vif.hsize <= 3'b010; //////write 4 byte
413. vif.htrans <= 2'b00;
415. @(posedge vif.hready);
416. @(posedge vif.clk);
417. $display("[DRV] : INCR8 TRANSFER ADDR : %0d DATA : %0d", vif.hwdata, vif.haddr);

420. repeat(7) begin
421. vif.hwdata <= $urandom\_range(1,50);
422. vif.htrans <= 2'b01;
423. $display("[DRV] : INCR8 TRANSFER ADDR : %0d DATA : %0d", vif.hwdata, vif.haddr);
424. @(posedge vif.hready);
425. @(posedge vif.clk);
426. end
427. ->drvnext;
428. endtask
430. ///////////////////////////////////////////////////////////////////
431. /////////////////////////////////INCR8 read
432. task incr8\_rd ();
433. @(posedge vif.clk);
435. vif.hresetn <= 1'b1;
437. vif.hburst <= 3'b101;
439. vif.hwrite <= 1'b0;
440. vif.hsel <= 1'b1;
442. vif.hwdata <= 0;
443. vif.haddr <= tr.haddr;
444. vif.hsize <= 3'b010; //////write 4 byte
446. vif.htrans <= 2'b00;
447. $display("[DRV] : INCR8 TRANSFER ADDR : %0d DATA : %0d", vif.hwdata, vif.haddr);
449. @(posedge vif.hready);
450. @(posedge vif.clk);
452. repeat(7) begin
453. vif.hwdata <= 0;
454. vif.htrans <= 2'b01;
455. @(posedge vif.hready);
456. @(posedge vif.clk);
457. $display("[DRV] : INCR8 TRANSFER ADDR : %0d DATA : %0d", vif.hwdata, vif.haddr);
459. end
460. ->drvnext;
461. endtask

464. //////////////////////////////////////////////////
465. ////////////////////////////////Increment 16 beats
467. task incr16\_wr();
469. @(posedge vif.clk);
470. vif.hresetn <= 1'b1;
472. vif.hburst <= 3'b111;
474. vif.hwrite <= 1'b1;
475. vif.hsel <= 1'b1;
477. vif.hwdata <= $urandom\_range(1,50);
478. vif.haddr <= tr.haddr;
479. vif.hsize <= 3'b010; //////write 4 byte
481. vif.htrans <= 2'b00;
482. @(posedge vif.hready);
483. @(posedge vif.clk);
484. $display("[DRV] : INCR16 TRANSFER ADDR : %0d DATA : %0d", vif.hwdata, vif.haddr);
486. repeat(15) begin
487. vif.hwdata <= $urandom\_range(1,50);
488. vif.htrans <= 2'b01;
489. @(posedge vif.hready);
490. @(posedge vif.clk);
491. $display("[DRV] : INCR16 TRANSFER ADDR : %0d DATA : %0d", vif.hwdata, vif.haddr);
493. end
494. ->drvnext;
495. endtask
497. ////////////////////////////////////////////////////////////////////
498. ////////////////////////////////////INCR 16 RD
499. task incr16\_rd();
501. @(posedge vif.clk);
502. vif.hresetn <= 1'b1;
504. vif.hburst <= 3'b111;
506. vif.hwrite <= 1'b0;
507. vif.hsel <= 1'b1;
509. vif.hwdata <= 0;
510. vif.haddr <= tr.haddr;
511. vif.hsize <= 3'b010; //////write 4 byte
513. vif.htrans <= 2'b00;
514. @(posedge vif.hready);
515. @(posedge vif.clk);
516. $display("[DRV] : INCR16 TRANSFER ADDR : %0d DATA : %0d", vif.hwdata, vif.haddr);

519. repeat(15) begin
520. vif.hwdata <= 0;
521. vif.htrans <= 2'b01;
522. @(posedge vif.hready);
523. @(posedge vif.clk);
524. $display("[DRV] : INCR16 TRANSFER ADDR : %0d DATA : %0d", vif.hwdata, vif.haddr);
526. end
527. ->drvnext;
528. endtask

531. /////////////////////////////////////////////////
532. //////////////////////////////////Wrap 4 beats
534. task wrap4\_wr();
535. @(posedge vif.clk);
537. vif.hresetn <= 1'b1;
539. vif.hburst <= 3'b010;
541. vif.hwrite <= 1'b1;
542. vif.hsel <= 1'b1;
544. vif.hwdata <= $urandom\_range(1,50);
545. vif.haddr <= tr.haddr;
546. vif.hsize <= 3'b010; //////write 4 byte
548. vif.htrans <= 2'b00;
550. @(posedge vif.hready);
551. @(posedge vif.clk);
552. $display("[DRV] : WRAP4 TRANSFER ADDR : %0d DATA : %0d", vif.hwdata, vif.haddr);
554. repeat(3) begin
555. vif.hwdata <= $urandom\_range(1,50);
556. vif.htrans <= 2'b01;
557. @(posedge vif.hready);
558. @(posedge vif.clk);
559. $display("[DRV] : WRAP4 TRANSFER ADDR : %0d DATA : %0d", vif.hwdata, vif.haddr);
561. end
562. ->drvnext;
563. endtask
565. ///////////////////////////////////////////////////////////////////
566. ///////////////////////////////////wrap4\_rd
568. task wrap4\_rd();
569. @(posedge vif.clk);
571. vif.hresetn <= 1'b1;
573. vif.hburst <= 3'b010;
575. vif.hwrite <= 1'b0;
576. vif.hsel <= 1'b1;
578. vif.hwdata <= 0;
579. vif.haddr <= tr.haddr;
580. vif.hsize <= 3'b010; //////write 4 byte
582. vif.htrans <= 2'b00;
583. @(posedge vif.hready);
584. @(posedge vif.clk);
585. $display("[DRV] : WRAP4 TRANSFER ADDR : %0d DATA : %0d", vif.hwdata, vif.haddr);

588. repeat(3) begin
589. vif.hwdata <= 0;
590. vif.htrans <= 2'b01;
591. @(posedge vif.hready);
592. @(posedge vif.clk);
593. $display("[DRV] : WRAP4 TRANSFER ADDR : %0d DATA : %0d", vif.hwdata, vif.haddr);
595. end
596. ->drvnext;
597. endtask

600. ///////////////////////////////////Wrap 8 beats write
602. task wrap8\_wr();
603. @(posedge vif.clk);
605. vif.hresetn <= 1'b1;
607. vif.hburst <= 3'b100;
609. vif.hwrite <= 1'b1;
610. vif.hsel <= 1'b1;
612. vif.hwdata <= $urandom\_range(1,50);
613. vif.haddr <= tr.haddr;
614. vif.hsize <= 3'b010; //////write 4 byte
616. vif.htrans <= 2'b00;
617. @(posedge vif.hready);
618. @(posedge vif.clk);
619. $display("[DRV] : WRAP8 TRANSFER ADDR : %0d DATA : %0d", vif.hwdata, vif.haddr);
621. repeat(7) begin
622. vif.hwdata <= $urandom\_range(1,50);
623. vif.htrans <= 2'b01;
624. @(posedge vif.hready);
625. @(posedge vif.clk);
626. $display("[DRV] : WRAP8 TRANSFER ADDR : %0d DATA : %0d", vif.hwdata, vif.haddr);
628. end
629. ->drvnext;
630. endtask
632. /////////////////////////////////////////////////////////////////////
633. /////////////////////////////////wrap 8 read
634. task wrap8\_rd();
635. @(posedge vif.clk);
637. vif.hresetn <= 1'b1;
639. vif.hburst <= 3'b100;
641. vif.hwrite <= 1'b0;
642. vif.hsel <= 1'b1;
644. vif.hwdata <= 0;
645. vif.haddr <= tr.haddr;
646. vif.hsize <= 3'b010; //////write 4 byte
648. vif.htrans <= 2'b00;
649. @(posedge vif.hready);
650. @(posedge vif.clk);
651. $display("[DRV] : WRAP8 TRANSFER ADDR : %0d DATA : %0d", vif.hwdata, vif.haddr);
653. repeat(7) begin
654. vif.hwdata <= 0;
655. vif.htrans <= 2'b01;
656. @(posedge vif.hready);
657. @(posedge vif.clk);
658. $display("[DRV] : WRAP8 TRANSFER ADDR : %0d DATA : %0d", vif.hwdata, vif.haddr);
660. end
661. ->drvnext;
662. endtask
664. ///////////////////////////////////////wrap 16 beats write
666. task wrap16\_wr();
667. @(posedge vif.clk);
669. vif.hresetn <= 1'b1;
671. vif.hburst <= 3'b110;
673. vif.hwrite <= 1'b1;
674. vif.hsel <= 1'b1;
676. vif.hwdata <= $urandom\_range(1,50);
677. vif.haddr <= tr.haddr;
678. vif.hsize <= 3'b010; //////write 4 byte
680. vif.htrans <= 2'b00;
681. @(posedge vif.hready);
682. @(posedge vif.clk);
683. $display("[DRV] : WRAP16 TRANSFER ADDR : %0d DATA : %0d", vif.hwdata, vif.haddr);
685. repeat(15) begin
686. vif.hwdata <= $urandom\_range(1,50);
687. vif.htrans <= 2'b01;
688. @(posedge vif.hready);
689. @(posedge vif.clk);
690. $display("[DRV] : WRAP16 TRANSFER ADDR : %0d DATA : %0d", vif.hwdata, vif.haddr);
692. end
693. ->drvnext;
694. endtask
696. ///////////////////////////////////////////////////////
697. //////////////////////////wrap 16 read
699. task wrap16\_rd();
701. @(posedge vif.clk);
703. vif.hresetn <= 1'b1;
705. vif.hburst <= 3'b110;
707. vif.hwrite <= 1'b0;
708. vif.hsel <= 1'b1;
710. vif.hwdata <= 0;
711. vif.haddr <= tr.haddr;
712. vif.hsize <= 3'b010; //////write 4 byte
714. vif.htrans <= 2'b00;
715. @(posedge vif.hready);
716. @(posedge vif.clk);
717. $display("[DRV] : WRAP16 TRANSFER ADDR : %0d DATA : %0d", vif.hwdata, vif.haddr);
719. repeat(15) begin
720. vif.hwdata <= 0;
721. vif.htrans <= 2'b01;
722. @(posedge vif.hready);
723. @(posedge vif.clk);
724. $display("[DRV] : WRAP16 TRANSFER ADDR : %0d DATA : %0d", vif.hwdata, vif.haddr);
726. end
727. ->drvnext;
728. endtask
730. /////////////////////////////////////////////////////////////////////

733. task run();
734. forever begin
735. mbxgd.get(tr);
737. if(tr.hwrite == 1'b1) begin
738. case(tr.hburst)
739. 3'b000: single\_tr\_wr();
740. 3'b001: unspec\_len\_wr();
741. 3'b010: wrap4\_wr();
742. 3'b011: incr4\_wr();
743. 3'b100: wrap8\_wr();
744. 3'b101: incr8\_wr();
745. 3'b110: wrap16\_wr();
746. 3'b111: incr16\_wr();
747. endcase
748. end
749. else begin
750. case(tr.hburst)
751. 3'b000: single\_tr\_rd();
752. 3'b001: unspec\_len\_rd();
753. 3'b010: wrap4\_rd();
754. 3'b011: incr4\_rd();
755. 3'b100: wrap8\_rd();
756. 3'b101: incr8\_rd();
757. 3'b110: wrap16\_rd();
758. 3'b111: incr16\_rd();
759. endcase
760. end
762. end
763. endtask

766. endclass
768. ///////////////////////////////////////////////////////////////////////////////////////
770. class monitor;
772. virtual ahb\_if vif;
773. transaction tr;
775. int len = 0;
776. bit [4:0] temp;
778. mailbox #(transaction) mbxms;
779. mailbox #(bit[4:0]) mbxgm;
781. function new( mailbox #(transaction) mbxms, mailbox #(bit[4:0]) mbxgm);
782. this.mbxms = mbxms;
783. this.mbxgm = mbxgm;
784. endfunction
786. ///////////////////////////////////Single transfer
787. ////////////////////////////////Write Operation
788. task single\_tr\_wr();
789. @(posedge vif.hready);
790. @(posedge vif.clk);
791. tr.haddr = vif.haddr;
792. tr.hwdata = vif.hwdata;
793. tr.hwrite = 1;
794. mbxms.put(tr);
795. $display("[MON]: SINGLE TRANSFER WRITE addr : %0d data : %0d", tr.haddr, tr.hwdata);
796. @(posedge vif.clk);
797. endtask
799. ///////////////////////////////Read Operation
800. task single\_tr\_rd();
801. @(posedge vif.hready);
802. @(posedge vif.clk);
803. tr.haddr = vif.haddr;
804. tr.hwrite = 0;
805. tr.hrdata = vif.hrdata;
806. mbxms.put(tr);
807. $display("[MON]: SINGLE TRANSFER READ addr : %0d data : %0d", tr.haddr, tr.hrdata);
808. @(posedge vif.clk);
809. endtask
811. ////////////////////////////////////////////////////////////////////////////
812. ///////////////////////unspec len write
813. task unspec\_tr\_wr();
814. mbxgm.get(temp);
815. repeat(temp) begin
816. @(posedge vif.hready);
817. @(posedge vif.clk);
818. tr.haddr = vif.next\_addr;;
819. tr.hwdata = vif.hwdata;
820. tr.hwrite = 1;
821. mbxms.put(tr);
822. $display("[MON]: UNSPECWR addr : %0d data : %0d", tr.haddr, tr.hwdata);
823. @(posedge vif.clk);
824. end
825. endtask
827. ///////////////////////////////////////////////////////////////////////////////
828. //////////////////////////unspec len read
829. task unspec\_tr\_rd();
830. mbxgm.get(temp);
831. repeat(temp) begin
832. @(posedge vif.hready);
833. @(posedge vif.clk);
834. tr.haddr = vif.next\_addr;;
835. tr.hwrite = 0;
836. tr.hrdata = vif.hrdata;
837. mbxms.put(tr);
838. $display("[MON]: UNSPECRD addr : %0d data : %0d", tr.haddr, tr.hrdata);
839. @(posedge vif.clk);
840. end
841. endtask
842. /////////////////////////////////////////////////////////////////////////////////////
843. ////////////////////////////////INCR 4 WRITE
844. task incr4\_wr();
845. $display("[MON] : INCR4 DATA WRITE");
846. repeat(4) begin
847. @(posedge vif.hready);
848. @(posedge vif.clk);
849. tr.haddr = vif.next\_addr;
850. tr.hwdata = vif.hwdata;
851. tr.hwrite = 1;
852. mbxms.put(tr);
853. $display("[MON]: addr : %0d data : %0d", tr.haddr, tr.hwdata);
854. @(posedge vif.clk);
855. end
856. endtask
858. //////////////////////////////////////////////////////////////////////////////////////
859. ///////////////////////////////INCR 4 READ
861. task incr4\_rd();
862. $display("[MON] : INCR4 DATA READ");
863. repeat(4) begin
864. @(posedge vif.hready);
865. @(posedge vif.clk);
866. tr.haddr = vif.next\_addr;
867. tr.hwrite = 0;
868. tr.hrdata = vif.hrdata;
869. mbxms.put(tr);
870. $display("[MON]: addr : %0d data : %0d", tr.haddr, tr.hrdata);
871. @(posedge vif.clk);
872. end
873. endtask
875. /////////////////////////////////////////////////////
876. ////////////////////////////wrap 4 write
877. task wrap4\_wr();
878. $display("[MON] : WRAP4 DATA WRITE");
879. repeat(4) begin
880. @(posedge vif.hready);
881. @(posedge vif.clk);
882. tr.haddr = vif.next\_addr;
883. tr.hwdata = vif.hwdata;
884. tr.hwrite = 1;
885. mbxms.put(tr);
886. $display("[MON]: addr : %0d data : %0d", tr.haddr, tr.hwdata);
887. @(posedge vif.clk);
888. end
889. endtask
891. /////////////////////////////////////////////////////////////////////
892. ////////////////////wrap 4 read
893. task wrap4\_rd();
894. $display("[MON] : WRAP4 DATA READ");
895. repeat(4) begin
896. @(posedge vif.hready);
897. @(posedge vif.clk);
898. tr.haddr = vif.next\_addr;
899. tr.hwrite = 0;
900. tr.hrdata = vif.hrdata;
901. mbxms.put(tr);
902. $display("[MON]: addr : %0d data : %0d", tr.haddr, tr.hrdata);
903. @(posedge vif.clk);
904. end
905. endtask
906. ///////////////////////////////////////////////////////////////
908. ////////////////////////////////INCR 8 WRITE
909. task incr8\_wr();
910. $display("[MON] : INCR8 DATA WRITE");
911. repeat(8) begin
912. @(posedge vif.hready);
913. @(posedge vif.clk);
914. tr.haddr = vif.next\_addr;
915. tr.hwdata = vif.hwdata;
916. tr.hwrite = 1;
917. mbxms.put(tr);
918. $display("[MON]: addr : %0d data : %0d", tr.haddr, tr.hwdata);
919. @(posedge vif.clk);
920. end
921. endtask
923. //////////////////////////////////////////////////////////////////////////////////////
924. ///////////////////////////////INCR 8 READ
926. task incr8\_rd();
927. $display("[MON] : INCR8 DATA READ");
928. repeat(8) begin
929. @(posedge vif.hready);
930. @(posedge vif.clk);
931. tr.haddr = vif.next\_addr;
932. tr.hwrite = 0;
933. tr.hrdata = vif.hrdata;
934. mbxms.put(tr);
935. $display("[MON]: addr : %0d data : %0d", tr.haddr, tr.hrdata);
936. @(posedge vif.clk);
937. end
938. endtask
940. /////////////////////////////////////////////////////
941. ////////////////////////////wrap 8 write
942. task wrap8\_wr();
943. $display("[MON] : WRAP8 DATA WRITE");
944. repeat(8) begin
945. @(posedge vif.hready);
946. @(posedge vif.clk);
947. tr.haddr = vif.next\_addr;
948. tr.hwdata = vif.hwdata;
949. tr.hwrite = 1;
950. mbxms.put(tr);
951. $display("[MON]: addr : %0d data : %0d", tr.haddr, tr.hwdata);
952. @(posedge vif.clk);
953. end
954. endtask
956. /////////////////////////////////////////////////////////////////////
957. ////////////////////wrap 8 read
958. task wrap8\_rd();
959. $display("[MON] : WRAP8 DATA READ");
960. repeat(8) begin
961. @(posedge vif.hready);
962. @(posedge vif.clk);
963. tr.haddr = vif.next\_addr;
964. tr.hwrite = 0;
965. tr.hrdata = vif.hrdata;
966. mbxms.put(tr);
967. $display("[MON]: addr : %0d data : %0d", tr.haddr, tr.hrdata);
968. @(posedge vif.clk);
969. end
970. endtask
971. ///////////////////////////////////////////////////////////////
972. //////////////////////////////////////////////////////////////
974. ////////////////////////////////INCR 16 WRITE
975. task incr16\_wr();
976. $display("[MON] : INCR16 DATA WRITE");
977. repeat(16) begin
978. @(posedge vif.hready);
979. @(posedge vif.clk);
980. tr.haddr = vif.next\_addr;
981. tr.hwdata = vif.hwdata;
982. tr.hwrite = 1;
983. mbxms.put(tr);
984. $display("[MON]: addr : %0d data : %0d", tr.haddr, tr.hwdata);
985. @(posedge vif.clk);
986. end
987. endtask
989. //////////////////////////////////////////////////////////////////////////////////////
990. ///////////////////////////////INCR 4 READ
992. task incr16\_rd();
993. $display("[MON] : INCR16 DATA READ");
994. repeat(16) begin
995. @(posedge vif.hready);
996. @(posedge vif.clk);
997. tr.haddr = vif.next\_addr;
998. tr.hwrite = 0;
999. tr.hrdata = vif.hrdata;
1000. mbxms.put(tr);
1001. $display("[MON]: addr : %0d data : %0d", tr.haddr, tr.hrdata);
1002. @(posedge vif.clk);
1003. end
1004. endtask
1006. /////////////////////////////////////////////////////
1007. ////////////////////////////wrap 16 write
1008. task wrap16\_wr();
1009. $display("[MON] : WRAP16 DATA WRITE");
1010. repeat(16) begin
1011. @(posedge vif.hready);
1012. @(posedge vif.clk);
1013. tr.haddr = vif.next\_addr;
1014. tr.hwdata = vif.hwdata;
1015. tr.hwrite = 1;
1016. mbxms.put(tr);
1017. $display("[MON]: addr : %0d data : %0d", tr.haddr, tr.hwdata);
1018. @(posedge vif.clk);
1019. end
1020. endtask
1022. /////////////////////////////////////////////////////////////////////
1023. ////////////////////wrap 16 read
1024. task wrap16\_rd();
1025. $display("[MON] : WRAP4 DATA READ");
1026. repeat(16) begin
1027. @(posedge vif.hready);
1028. @(posedge vif.clk);
1029. tr.haddr = vif.next\_addr;
1030. tr.hwrite = 0;
1031. tr.hrdata = vif.hrdata;
1032. mbxms.put(tr);
1033. $display("[MON]: addr : %0d data : %0d", tr.haddr, tr.hrdata);
1034. @(posedge vif.clk);
1035. end
1036. endtask
1037. ///////////////////////////////////////////////////////////////


1041. ////////////////////////////////////////////////////////////////////////////
1042. task run();
1043. tr = new();
1044. forever begin
1045. @(posedge vif.clk);
1047. if(vif.hresetn && vif.hsel && vif.hwrite)
1048. begin
1049. case(vif.hburst)
1050. 3'b000: single\_tr\_wr();
1051. 3'b001: unspec\_tr\_wr();
1052. 3'b010: wrap4\_wr();
1053. 3'b011: incr4\_wr();
1054. 3'b100: wrap8\_wr();
1055. 3'b101: incr8\_wr();
1056. 3'b110: wrap16\_wr();
1057. 3'b111: incr16\_wr();
1058. endcase
1059. end

1062. if(vif.hresetn && vif.hsel && (vif.hwrite == 0))
1063. begin
1064. case(vif.hburst)
1065. 3'b000: single\_tr\_rd();
1066. 3'b001: unspec\_tr\_rd();
1067. 3'b010: wrap4\_rd();
1068. 3'b011: incr4\_rd();
1069. 3'b100: wrap8\_rd();
1070. 3'b101: incr8\_rd();
1071. 3'b110: wrap16\_rd();
1072. 3'b111: incr16\_rd();
1073. endcase
1074. end
1076. end

1079. endtask
1081. endclass
1083. ////////////////////////////////////////////////////////////////////////////
1085. class scoreboard;
1087. transaction tr;
1088. event sconext;
1090. mailbox #(transaction) mbxms;
1092. bit [7:0] data[256] = '{default:12};
1094. int count = 0;
1095. int len = 0;
1096. bit [31:0] rdata;

1099. function new( mailbox #(transaction) mbxms );
1100. this.mbxms = mbxms;
1101. endfunction

1104. task run();
1106. forever
1107. begin

1110. mbxms.get(tr);
1112. if(tr.hwrite == 1'b1) begin
1113. $display("[SCO] : DATA WRITE");
1114. data[tr.haddr] = tr.hwdata[7:0];
1115. data[tr.haddr + 1] = tr.hwdata[15:8];
1116. data[tr.haddr + 2] = tr.hwdata[23:16];
1117. data[tr.haddr + 3] = tr.hwdata[31:24];
1118. end
1120. if(tr.hwrite == 1'b0) begin
1121. rdata = {data[tr.haddr + 3], data[tr.haddr + 2], data[tr.haddr + 1], data[tr.haddr]};
1122. if(tr.hrdata == 32'h0c0c0c0c)
1123. $display("[SCO] : EMPTY LOCATION READ");
1124. else if (tr.hrdata == rdata)
1125. $display("[SCO] : DATA MATCHED");
1126. else
1127. $display("[SCO] : DATA MISMATCHED");
1128. end

1131. ->sconext;
1132. end
1133. endtask

1136. endclass


1140. //////////////////////////////////////////////////////////////////////////////////////
1141. module tb;
1143. monitor mon;
1144. generator gen;
1145. driver drv;
1146. scoreboard sco;


1150. event nextgd;
1151. event nextgs;
1152. event done;
1154. mailbox #(transaction) mbxgd, mbxms;
1155. mailbox #(bit[4:0]) mbxgm;
1157. ahb\_if vif();
1159. ahb\_slave dut (vif.clk, vif.hwdata, vif.haddr, vif.hsize, vif.hburst, vif.hresetn, vif.hsel, vif.hwrite, vif.htrans, vif.hresp, vif.hready, vif.hrdata);
1161. initial begin
1162. vif.clk <= 0;
1163. end
1165. always #5 vif.clk <= ~vif.clk;
1167. initial begin
1168. mbxgd = new();
1169. mbxms = new();
1170. mbxgm = new();
1172. gen = new(mbxgd, mbxgm);
1173. drv = new(mbxgd);
1174. mon = new(mbxms, mbxgm);
1175. sco = new(mbxms);
1177. gen.count = 3;
1179. drv.vif = vif;
1180. mon.vif = vif;
1182. drv.drvnext = nextgd;
1183. gen.drvnext = nextgd;
1185. gen.sconext = nextgs;
1186. sco.sconext = nextgs;
1188. end
1190. initial begin
1191. drv.reset();
1192. fork
1193. gen.run();
1194. drv.run();
1195. mon.run();
1196. sco.run();
1197. join\_none
1198. wait(gen.done.triggered);
1199. $finish();
1200. end
1202. initial begin
1203. $dumpfile("dump.vcd");
1204. $dumpvars;
1205. end
1207. assign vif.next\_addr = dut.next\_addr;
1209. endmodule

WHISH BONE:

1. `timescale 1ns / 1ps
3. interface wb\_if;
4. logic clk;
5. logic we;
6. logic strb;
7. logic rst;
8. logic [7:0] addr;
9. logic [7:0] wdata;
10. logic [7:0] rdata;
11. logic ack;
12. endinterface

15. module mem\_wb(
16. input clk,we,strb,rst,
17. input [7:0] addr,
18. input [7:0] wdata,
19. output reg [7:0] rdata,
20. output reg ack
21. );
23. reg [7:0] mem[256];
25. reg [7:0] temp;

28. typedef enum bit [1:0] {idle = 0, check\_mode = 1, write = 2, read = 3} state\_type;
29. state\_type state, next\_state;
31. //////////////////////reset decoder
32. always\_ff@(posedge clk)
33. begin
34. if(rst)
35. begin
36. state <= idle;
37. end
38. else
39. state <= next\_state;
40. end
42. ///////////next state and output decoder
43. always\_comb
44. begin
45. case(state)
47. idle: begin
48. ack = 1'b0;
49. rdata = 8'h00;
50. next\_state = check\_mode;
51. for(int i = 0; i < 256; i++) begin
52. mem[i] <= 8'h11;
53. end
55. end
57. check\_mode: begin
58. if(strb && we)
59. begin
60. next\_state = write;
61. end
62. else if (strb && !we)
63. begin
64. next\_state = read;
65. temp = mem[addr];
66. end
67. else
68. begin
69. next\_state = check\_mode;
70. end
71. end

74. write: begin
75. mem[addr] = wdata;
76. ack = 1'b1;
77. next\_state = idle;
78. end
80. read: begin
81. rdata = temp;
82. ack = 1'b1;
83. next\_state = idle;
84. end
86. default: next\_state = idle;
87. endcase
88. end
89. endmodule

TESTBENCH:

1. class transaction;
3. randc bit [1:0] opmode; /// write = 0, read = 1, random =2
4. rand bit we;
5. rand bit strb;
6. rand bit [7:0] addr;
7. rand bit [7:0] wdata;
8. bit [7:0] rdata;
9. bit ack;
11. constraint opmode\_c {
12. opmode >= 0; opmode < 3;
13. }
15. constraint addr\_c {
16. addr == 5;
17. }
19. constraint wdata\_c {
20. wdata > 0; wdata <= 8;
21. }
23. function transaction copy();
24. copy = new();
25. copy.opmode = this.opmode;
26. copy.we = this.we;
27. copy.strb = this.strb;
28. copy.addr = this.addr;
29. copy.wdata = this.wdata;
30. copy.rdata = this.rdata;
31. copy.ack = this.ack;
32. endfunction
34. function void display(input string tag);
35. $display("[%0s] : MODE :%0d WE : %0b STRB : %0b ADDR : %0d WDATA : %0d RDATA : %0d", tag,opmode, we,strb,addr,wdata,rdata);
36. endfunction

39. endclass
41. /////////////////////////////////////////////////////////////////////////

44. class generator;
46. transaction tr;
47. mailbox #(transaction) mbxgd;
48. event done; ///gen completed sending requested no. of transaction
49. event drvnext; /// dr complete its wor;
50. event sconext; ///scoreboard complete its work
52. int count = 0;
54. function new( mailbox #(transaction) mbxgd);
55. this.mbxgd = mbxgd;
56. tr =new();
57. endfunction
59. task run();
61. for(int i=0; i< count; i++) begin
62. assert(tr.randomize) else $error("Randomization Failed");
63. $display("------------------------------------------");
64. tr.display("GEN");
65. mbxgd.put(tr.copy);
66. @(drvnext);
67. @(sconext);
68. end
70. ->done;
72. endtask

75. endclass
76. ///////////////////////////////////////////////////////////
78. class driver;
80. virtual wb\_if vif;
81. transaction tr;
82. event drvnext;
84. mailbox #(transaction) mbxgd;

87. function new( mailbox #(transaction) mbxgd );
88. this.mbxgd = mbxgd;
89. endfunction
91. task reset();
92. vif.rst <= 1'b1;
93. vif.we <= 0;
94. vif.addr <= 0;
95. vif.wdata <= 0;
96. vif.strb <= 0;
97. repeat(10) @(posedge vif.clk);
98. vif.rst <= 1'b0;
99. repeat(5) @(posedge vif.clk);
100. $display("[DRV] : RESET DONE");
101. endtask
103. task write();
104. @(posedge vif.clk);
105. $display("[DRV] : DATA WRITE MODE");
106. vif.rst <= 1'b0;
107. vif.we <= 1'b1;
108. vif.strb <= 1'b1;
109. vif.addr <= tr.addr;
110. vif.wdata <= tr.wdata;
111. @(posedge vif.ack);
112. @(posedge vif.clk);
113. ->drvnext;
114. endtask
116. task read();
117. @(posedge vif.clk);
118. $display("[DRV] : DATA READ MODE");
119. vif.rst <= 1'b0;
120. vif.we <= 1'b0;
121. vif.strb <= 1'b1;
122. vif.addr <= tr.addr;
123. @(posedge vif.ack);
124. @(posedge vif.clk);
125. ->drvnext;
126. endtask
128. task random();
129. @(posedge vif.clk);
130. $display("[DRV] : RANDOM MODE");
131. vif.rst <= 1'b0;
132. vif.we <= tr.we;
133. vif.strb <= tr.strb;
134. vif.addr <= tr.addr;
135. if(tr.we == 1'b1)
136. begin
137. vif.wdata <= tr.wdata;
138. end
139. repeat(2)@(posedge vif.clk);
140. ->drvnext;
141. endtask

144. task run();
145. forever begin
146. mbxgd.get(tr);
147. if(tr.opmode == 0)
148. begin
149. write();
150. end
151. else if (tr.opmode == 1)
152. begin
153. read();
154. end
155. else if(tr.opmode == 2)
156. begin
157. random();
158. end
159. end
160. endtask

163. endclass
165. ////////////////////////////////////////////////
167. class monitor;
169. virtual wb\_if vif;
170. transaction tr;
172. mailbox #(transaction) mbxms;

175. function new( mailbox #(transaction) mbxms );
176. this.mbxms = mbxms;
177. endfunction

180. task run();
182. tr = new();
184. forever
185. begin
186. wait( vif.rst == 1'b0);
187. repeat(5) @(posedge vif.clk);
188. @(posedge vif.clk);
189. if(vif.strb == 1'b0)
190. begin
191. tr.strb = vif.strb;
192. repeat(2) @(vif.clk);
193. $display("[MON] : STRB IS ZERO");
194. mbxms.put(tr);
195. end
196. else
197. begin
198. @(posedge vif.ack);
199. tr.we = vif.we;
200. tr.strb = vif.strb;
201. tr.wdata = vif.wdata;
202. tr.addr = vif.addr;
203. tr.rdata = vif.rdata;
204. @(posedge vif.clk);
205. $display("[MON] : STRB IS VALID");
206. mbxms.put(tr);
207. end
208. end
209. endtask
211. endclass
212. ///////////////////////////////////////////////////////////////
214. class scoreboard;
215. transaction tr;
216. event sconext;
218. mailbox #(transaction) mbxms;
220. bit [7:0] data[256] = '{default:0};

223. function new( mailbox #(transaction) mbxms );
224. this.mbxms = mbxms;
225. endfunction
227. task run();
228. forever begin
229. mbxms.get(tr);
230. if(tr.strb == 1'b0)
231. begin
232. $display("[SCO] : INVALID STROBE");
233. end
234. else
235. begin
237. if(tr.we == 1'b1)
238. begin
239. data[tr.addr] = tr.wdata;
240. $display("[SCO] : DATA WRITE DATA : %0d ADDR : %0d", tr.wdata, tr.addr);
241. end
242. else
243. begin
244. if(tr.rdata == 8'h11)
245. begin
246. $display("[SCO] : DATA MATCHED : DEFAULT VALUE READ");
247. end
248. else if (tr.rdata == data[tr.addr])
249. begin
250. $display("[SCO] : DATA MATCHED DATA : %0d ADDR : %0d", tr.wdata, tr.addr);
251. end
252. else
253. begin
254. $display("[SCO] : DATA MISMATCHED DATA : %0d ADDR : %0d", tr.wdata, tr.addr);
255. end
256. end
257. end
258. $display("------------------------------------------");
259. ->sconext;
260. end
261. endtask
262. endclass



267. ///////////////////////////////////////////////////
269. module tb;
271. generator gen;
272. driver drv;
273. monitor mon;
274. scoreboard sco;
276. event drvnext, sconext;
277. event done;
279. mailbox #(transaction) mbxgd;
280. mailbox #(transaction) mbxms;
282. wb\_if vif();
283. mem\_wb dut (vif.clk, vif.we, vif.strb, vif.rst, vif.addr, vif.wdata, vif.rdata, vif.ack);
285. initial begin
286. vif.clk <= 0;
287. end
289. always #5 vif.clk <= ~vif.clk;
291. initial begin
293. mbxgd = new();
294. mbxms = new();
295. gen = new(mbxgd);
296. drv = new(mbxgd);
297. mon = new(mbxms);
298. sco = new(mbxms);
299. gen.count = 10;
300. drv.vif = vif;
301. mon.vif = vif;
303. drv.drvnext = drvnext;
304. gen.drvnext = drvnext;
306. gen.sconext = sconext;
307. sco.sconext = sconext;
309. end
311. initial begin
312. drv.reset();
313. fork
314. gen.run();
315. drv.run();
316. mon.run();
317. sco.run();
318. join\_none
319. wait(gen.done.triggered);
320. $finish();
321. end
323. initial begin
324. $dumpfile("dump.vcd");
325. $dumpvars;
326. end

329. endmodule