Q4

A

```
module register(output reg q,input d,input clk);
always@(posedge clk)
q<=d;
endmodule

module register_8bit(output [7:0]q,input [7:0]d,input clk);
register r0(q[0],d[0],clk);
register r1(q[1],d[1],clk);
register r2(q[2],d[2],clk);
register r3(q[3],d[3],clk);
register r4(q[4],d[4],clk);
register r5(q[5],d[5],clk);
register r6(q[6],d[6],clk);
register r7(q[7],d[7],clk);
endmodule

module register_8bit_tb;
reg [7:0]d;
wire [7:0]q;
reg clk;
register_8bit inst(q,d,clk);
initial begin
clk=1'b1;
repeat(15)
#10 clk=~clk;
end
initial begin
{d}=0;
#25 d=$random();
#25 d=$random();
#30 d=$random();
#15 d=$random();
#5 d=$random();
#10 d=$random();
#15 d=$random();
#25 d=$random();
end
endmodule
```

B

```
module register(output reg q,input d,input clk);
always@(posedge clk)
q<=d;
endmodule

module register_16_bit(output [15:0]q,input [15:0]d,input clk);
register r0(q[0],d[0],clk);
register r1(q[1],d[1],clk);
```

```
register r2(q[2],d[2],clk);
register r3(q[3],d[3],clk);
register r4(q[4],d[4],clk);
register r5(q[5],d[5],clk);
register r6(q[6],d[6],clk);
register r7(q[7],d[7],clk);
register r8(q[8],d[8],clk);
register r9(q[9],d[9],clk);
register r10(q[10],d[10],clk);
register r11(q[11],d[11],clk);
register r12(q[12],d[12],clk);
register r13(q[13],d[13],clk);
register r14(q[14],d[14],clk);
register r15(q[15],d[15],clk);
endmodule

module register_16bit_tb;
reg [15:0]d;
wire [15:0]q;
reg clk;
register_16_bit inst(q,d,clk);
initial begin
clk=1'b1;
repeat(15)
#10 clk=~clk;
end
initial begin
{d}=0;
#25 d=$random();
#25 d=$random();
#30 d=$random();
#15 d=$random();
#5 d=$random();
#10 d=$random();
#15 d=$random();
#25 d=$random();
end
endmodule
```

C

```
module block(output [7:0]q,input [7:0]d,input clk);
register r0(q[0],d[0],clk);
register r1(q[1],d[1],clk);
register r2(q[2],d[2],clk);
register r3(q[3],d[3],clk);
register r4(q[4],d[4],clk);
register r5(q[5],d[5],clk);
register r6(q[6],d[6],clk);
register r7(q[7],d[7],clk);
endmodule

module register(output reg q,input d,input clk);
always@(posedge clk)
```

```
q<=d;
endmodule

module registers_4_8(output [7:0]q1,q2,q3,q4,input [7:0]d1,d2,d3,d4,input clk);
block u0(q1,d1,clk);
block u1(q2,d2,clk);
block u2(q3,d3,clk);
block u3(q4,d4,clk);
endmodule

module registers_4_8_tb;
reg [7:0]d1,d2,d3,d4;
wire [7:0]q1,q2,q3,q4;
reg clk;
registers_4_8 inst(q1,q2,q3,q4,d1,d2,d3,d4,clk);
initial begin
clk=1'b1;
repeat(15)
#10 clk=~clk;
end
initial begin
{d1}=0;
{d2}=0;
{d3}=0;
{d4}=0;
#25 d1=$random();
  d2=$random();
  d3=$random();
  d4=$random();
#25 d1=$random();
  d2=$random();
  d3=$random();
  d4=$random();
#30 d1=$random();
  d2=$random();
  d3=$random();
  d4=$random();
#15 d1=$random();
  d2=$random();
  d3=$random();
  d4=$random();
#5 d1=$random();
  d2=$random();
  d3=$random();
  d4=$random();
#10 d1=$random();
  d2=$random();
  d3=$random();
  d4=$random();
#15 d1=$random();
  d2=$random();
  d3=$random();
  d4=$random();
#25 d1=$random();
  d2=$random();
```

```
  d3=$random();
  d4=$random();
end
endmodule
```

D

```
module Decoder24(output [3:0]d,input s0,s1);
 assign d[0]=(~s0)&(~s1);
 assign d[1]=(s0)&(~s1);
 assign d[2]=(~s0)&(s1);
 assign d[3]=(s0)&(s1);
endmodule

module d_latch(output reg  Q,input D,en,clear);
 always @(en,clear)begin
 if (clear & en) begin
  Q=0;
 end
 else if(en)begin
  Q=D;
  end
 end
endmodule

module sram4*2(Dout,Din,address,writenable);
 output [1:0] Dout;
 input [1:0] Din,address;
 input writenable;
 wire [3:0] en,andr;
 wire [7:0]q;
 Decoder24 inst(en,address[0],address[1]);
 assign andr[0]=en[0]&writenable;
 assign andr[1]=en[1]&writenable;
 assign andr[2]=en[2]&writenable;
 assign andr[3]=en[3]&writenable;
 d_latch inst1(q[0],Din[1],en[0],andr[0]);
 d_latch inst2(q[1],Din[1],en[1],andr[1]);
 d_latch inst3(q[2],Din[1],en[2],andr[2]);
 d_latch inst4(q[3],Din[1],en[3],andr[3]);
 d_latch inst5(q[4],Din[0],en[0],andr[0]);
 d_latch inst6(q[5],Din[0],en[1],andr[1]);
 d_latch inst7(q[6],Din[0],en[2],andr[2]);
 d_latch inst8(q[7],Din[0],en[3],andr[3]);

 assign Dout[1]=q[0]|q[1]|q[2]|q[3];
 assign Dout[0]=q[4]|q[5]|q[6]|q[7];

endmodule

module sram4*2_tb;
 wire [1:0] Dout;
 reg [1:0] Din,address;
```

```
 reg wr;
 initial begin:loop
 integer count;
 for (count=0;count<32;count=count+1) begin
  {Din,address,wr}=count;
  #10;
  end
 end
 Sram4*2 what(Dout,Din,address,wr);
endmodule
```

E

```
module Decoder24(output [3:0]d,input s0,s1);
 assign d[0]=(~s0)&(~s1);
 assign d[1]=(s0)&(~s1);
 assign d[2]=(~s0)&(s1);
 assign d[3]=(s0)&(s1);
endmodule

module d_latch(output reg  Q,input D,en,clear);
 always @(en,clear)begin
 if (clear & en) begin
  Q=0;
 end
 else if(en)begin
  Q=D;
  end
 end
endmodule

module Sram4*4(Dout,Din,address,writenable);
 output [3:0] Dout;
 input [3:0] Din,address;
 input writenable;
 wire [3:0] en,andr;
 wire [15:0]q;
 Decoder24 inst(en,address[0],address[1]);

 assign andr[0]=en[0]&writenable;
 assign andr[1]=en[1]&writenable;
 assign andr[2]=en[2]&writenable;
 assign andr[3]=en[3]&writenable;

 d_latch inst1(q[0],Din[0],en[0],andr[0]);
 d_latch inst2(q[1],Din[0],en[1],andr[1]);
 d_latch inst3(q[2],Din[0],en[2],andr[2]);
 d_latch inst4(q[3],Din[0],en[3],andr[3]);

 d_latch inst5(q[4],Din[1],en[0],andr[0]);
 d_latch inst6(q[5],Din[1],en[1],andr[1]);
 d_latch inst7(q[6],Din[1],en[2],andr[2]);
 d_latch inst8(q[7],Din[1],en[3],andr[3]);
```

```
 d_latch inst11(q[8],Din[2],en[0],andr[0]);
 d_latch inst21(q[9],Din[2],en[1],andr[1]);
 d_latch inst31(q[10],Din[2],en[2],andr[2]);
 d_latch inst41(q[11],Din[2],en[3],andr[3]);

 d_latch inst51(q[12],Din[3],en[0],andr[0]);
 d_latch inst61(q[13],Din[3],en[1],andr[1]);
 d_latch inst71(q[14],Din[3],en[2],andr[2]);
 d_latch inst81(q[15],Din[3],en[3],andr[3]);

 assign Dout[0]=q[0]|q[1]|q[2]|q[3];
 assign Dout[1]=q[4]|q[5]|q[6]|q[7];
 assign Dout[2]=q[8]|q[9]|q[10]|q[11];
 assign Dout[3]=q[12]|q[13]|q[14]|q[15];
endmodule

module sram4*4_tb;
 wire [3:0] Dout;
 reg [3:0] Din;
 reg [1:0] address;
 reg wr;
 initial begin:loop
 integer count;
 for (count=0;count<128;count=count+1) begin
  {Din,address,wr}=count;
  #10;
  end
 end
 Sram4*4 what(Dout,Din,address,wr);
endmodule
```