ESA
Exam of Skill Assessment

「ESA-VerilogDesign」技能試験
問題II（記述・論述式問題） サンプル

問題番号	問1～問13
試験時間	80 分

注意事項

1．解答用紙の記入に当たっては、次の指示に従ってください。指示に従わない場合には、採点されません。

(1) 所属名欄
所属名を記入してください。

(2) 従業員番号、氏名欄
未記入や誤記入のない様に正しく記入して下さい。

(3) 解答欄
文字は採点者にわかるように記入して下さい。

2．乱丁・落丁がないか、試験開始直後に確認して下さい。もしある場合は速やかに申し出て下さい。

3．論理合成に関する問題については、問題文に特に指定がない限り Synopsys 社の Design Compiler™ 2000.11-SP2 の使用を前提とします。また、論理合成の結果に関する問題については、TSMC 社 0.25μmスタンダードセル(SAGE™)用のライブラリ slow.db を使用し、論理合成時に特別なタイミング制約を与えない、通常のコンパイル結果を正解とします。

指示があるまで開いてはいけません。
問題に関する質問にはお答えできません。

問１から問１３まで記述式です。

問１ 次の記述の論理合成を試みたところ、RTL シミュレーションとゲートレベル・シミュレーションとで結果が一致しなかった。一致しなかった理由を 50 文字以内で述べよ。

```
module TESTA(AI,BI,CI,YOUT);
input  AI, BI, CI;
output YOUT;
reg    YOUT;

always @(AI or BI) begin
   YOUT = (AI & BI) | (BI & CI) | (CI & AI);
end

endmodule
```

問２ 100 万ゲート規模の LSI 設計では、ホールドタイム保証（注）を実行した後にタイミングレポートを出力させても、多量のバッファ及びインバータが挿入されたパスを発見するのは困難である。どのような方法で、このようなパスがあるかを確認すればよいか。100 文字以内で述べよ。

注：「ホールドタイム保証」
タイミング解析の結果、FF のホールドタイムに違反している場合、論理合成ツールは違反をなくすためにバッファあるいはインバータを挿入する。このことを「ホールドタイム保証」と呼ぶ。

問３ 次の１～４の記述を論理合成した結果を推測し、回路面積の小さい順、動作速度の速い順にならべよ。（「動作速度が速い」とは、組み合わせ回路の遅延が小さいことをいう）

＜共通部分＞

```
module HIKAKU(ain,bin,sel);
input[15:0] ain;
input[15:0] bin;
output sel;

  //選択肢の記述

endmodule
```

＜選択肢＞

1.
```
reg    sel;
always @(ain or bin) begin
  case (ain)
    16'b0000000000000001: sel = bin[0];
    16'b0000000000000010: sel = bin[1];
    16'b0000000000000100: sel = bin[2];
    16'b0000000000001000: sel = bin[3];
    16'b0000000000010000: sel = bin[4];
    16'b0000000000100000: sel = bin[5];
    16'b0000000001000000: sel = bin[6];
    16'b0000000010000000: sel = bin[7];
    16'b0000000100000000: sel = bin[8];
    16'b0000001000000000: sel = bin[9];
    16'b0000010000000000: sel = bin[10];
    16'b0000100000000000: sel = bin[11];
    16'b0001000000000000: sel = bin[12];
    16'b0010000000000000: sel = bin[13];
    16'b0100000000000000: sel = bin[14];
    16'b1000000000000000: sel = bin[15];
    default             : sel = 1'b0;
  endcase
end
```

2.
```
reg    sel;
always @(ain or bin) begin
  case (ain)
    16'b0000000000000001: sel = bin[0];
    16'b0000000000000010: sel = bin[1];
    16'b0000000000000100: sel = bin[2];
    16'b0000000000001000: sel = bin[3];
    16'b0000000000010000: sel = bin[4];
    16'b0000000000100000: sel = bin[5];
    16'b0000000001000000: sel = bin[6];
    16'b0000000010000000: sel = bin[7];
    16'b0000000100000000: sel = bin[8];
    16'b0000001000000000: sel = bin[9];
    16'b0000010000000000: sel = bin[10];
    16'b0000100000000000: sel = bin[11];
    16'b0001000000000000: sel = bin[12];
    16'b0010000000000000: sel = bin[13];
    16'b0100000000000000: sel = bin[14];
    16'b1000000000000000: sel = bin[15];
    default             : sel = 1'bx;
  endcase
end
```

3.
```
reg    sel;
always @(ain or bin) begin
  if(ain[0])       sel = bin[0];
  else if(ain[1])  sel = bin[1];
  else if(ain[2])  sel = bin[2];
  else if(ain[3])  sel = bin[3];
  else if(ain[4])  sel = bin[4];
  else if(ain[5])  sel = bin[5];
  else if(ain[6])  sel = bin[6];
  else if(ain[7])  sel = bin[7];
  else if(ain[8])  sel = bin[8];
  else if(ain[9])  sel = bin[9];
  else if(ain[10]) sel = bin[10];
  else if(ain[11]) sel = bin[11];
  else if(ain[12]) sel = bin[12];
  else if(ain[13]) sel = bin[13];
  else if(ain[14]) sel = bin[14];
  else             sel = bin[15];
end
```

4.
```
assign sel = (ain[0]  & bin[0])  | (ain[1]  & bin[1])  |
             (ain[2]  & bin[2])  | (ain[3]  & bin[3])  |
             (ain[4]  & bin[4])  | (ain[5]  & bin[5])  |
             (ain[6]  & bin[6])  | (ain[7]  & bin[7])  |
             (ain[8]  & bin[8])  | (ain[9]  & bin[9])  |
             (ain[10] & bin[10]) | (ain[11] & bin[11]) |
             (ain[12] & bin[12]) | (ain[13] & bin[13]) |
             (ain[14] & bin[14]) | (ain[15] & bin[15])  ;
```

問４ 次に示すロード･カウントイネーブル付きカウンタの動作確認するためのテストベンチがある。このテストベンチで回路を正しく検証できるように(1)～(5)の空欄を埋めよ。

＜回路記述＞

```
module loadcounter(CLK,RST_X,LoadEn,CountEn,LoadData,CONT);
input CLK;
input RST_X;
input LoadEn;
input CountEn;
input [7:0] LoadData;
output [7:0] CONT;
reg [7:0] CONT;

always @(posedge CLK or negedge RST_X) begin
  if(!RST_X)
    CONT <= 8'd0;
  else if(LoadEn)
    CONT <= LoadData;
  else if(CountEn)
    CONT <= CONT + 1;
end

endmodule
```

＜テストベンチ記述＞

```
module loadcounter_test;
parameter STB   = 1;
parameter HALF_CYCLE = 10;
reg CLK;
reg RST_X;
reg LoadEn;
reg CountEn;
reg [7:0] LoadData;
reg [7:0] LoadValue;
wire [7:0] CONT;
integer I;

loadcounter loadcounter(.CLK(CLK),.RST_X(RST_X),
                        .LoadEn(LoadEn), .CountEn(CountEn),
                        .LoadData(LoadData), .CONT(CONT));

always begin
              CLK = 1'b1;
  #HALF_CYCLE CLK = 1'b0;
  #HALF_CYCLE ;
end

task Set_LoadData;
input [7:0] LoadValue;
begin
  @(posedge CLK )_________(1)_________;
                 _________(2)_________;
  @(posedge CLK )_________(3)_________;
end
      (4)
```

＜次ページにつづく＞

＜前ページからのつづき＞

```
initial begin
  RST_X = 1'b0; CountEn = 1'b0; LoadEn = 1'b0; LoadData = 8'd0;
  #STB ;
  @(posedge CLK )  RST_X = _____(5)_____;

  for(I=0; I<256;I=I+1) begin
     Set_LoadData(I);                  // Call  Set_LoadData task
  end

  @(posedge CLK )  CountEn = #STB 1'b1;

  for(I=0; I<256;I=I+1) begin
     Set_LoadData(I);                 // Call  Set_LoadData task
     repeat(10) @(posedge CLK);
  end

  $finish;
end

endmodule
```