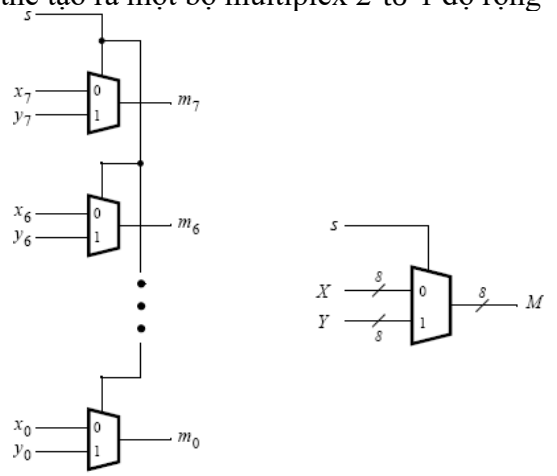
Họ và tên: Hà Minh Khuê **MSSV: 1513079**

Bài tập tuần 1

Yêu cầu của bài thực hành

1. Hiển thị các đèn LEDR thông qua các SW:
   - Các LEDR kích sáng khi lên trạng thái 1 và tắt khi ở trạng thái 0.
   - Mỗi toggle switch điều khiển tắt, một đèn LEDR: SW[0] ➔LEDR[0], …
2. Thiết kế module thực hiện bộ multiplex 2-to-1 độ rộng 4 bit:

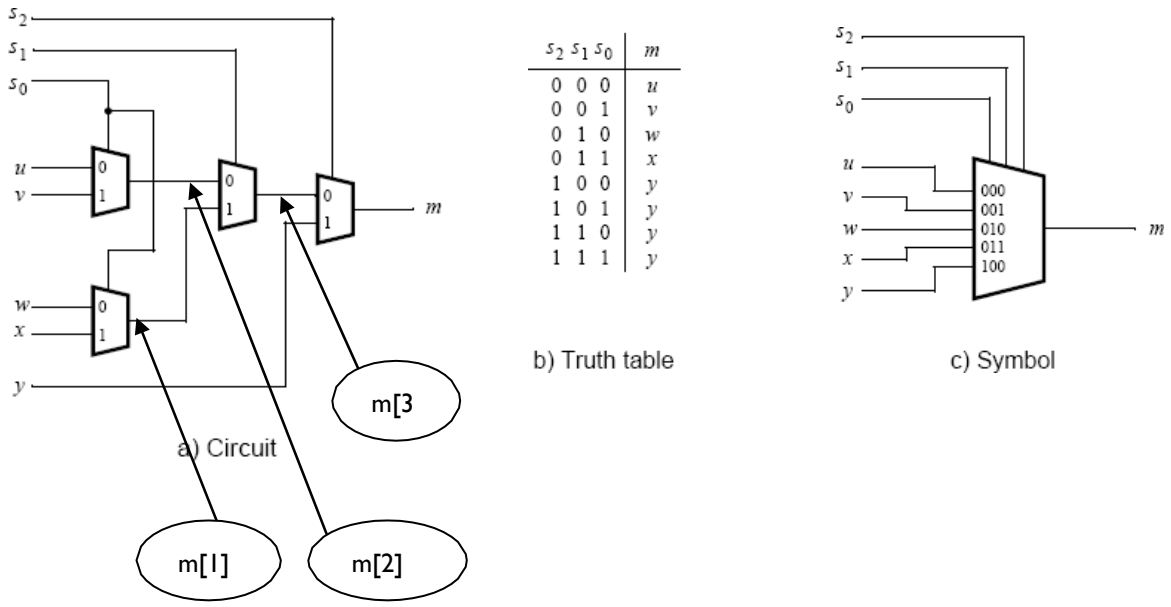
Dựa vào mô tả trên ta có thể tạo ra một bộ multiplex 2-to-1 độ rộng 4 bit như hình sau:



Hình 1.36: Sơ đồ ch và ký hiệu củ bộ u tip ex 2–to-1 (8bit) [22]

Để kiểm tra trên board DE1 ta sử dụng LEDR, SW. Ngõ vào x_in là SW[3:0], ngõ vào y_in là SW[7:4], SW[9] là select, và LEDR[3:0] là ngõ ra.

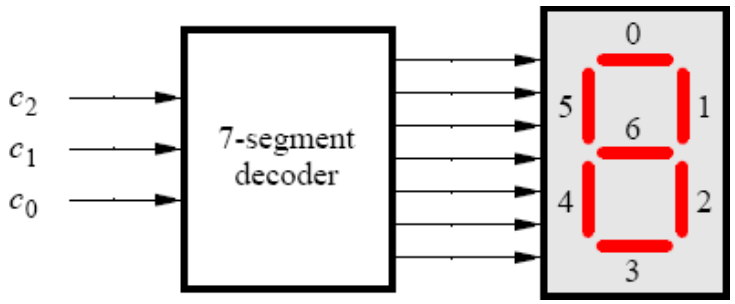
3. Mở rộng từ câu 2, trong phần này ta s thiết kế bộ multiplex 5-to-1 độ rộng 1bit.
   - Thiết kế này dựa trên cơ sở bộ multiplex 2-to-1:



Hình 1.37: Sơ đồ ch, bảng sự thật và ký hiệu củ bộ u tip ex 5–to-1 [22]

4. Thiết kế bộ giải mã LED 7 đoạn với 3-bit ngõ hiển thị các ký tự H, E, L, : trong phần này ch ng ta s tạo ra một module giải mã (decode) các bit đầu ra thành các k tự „ H ‟,„E‟, „L‟, „ ‟ với một đèn LED 7 đoạn.

- y là mô tả bằng hình bộ giải mã như sau:



Hình 1.39: Bộ giải đo n [22]

- Trong đ ta c 3 ng vào C_0, C_1, C_2 đi vào bộ giải mã 7 đoạn r i xuất ra đèn led.
- Lưu là đối với loại led 7 đoạn trên mạch này thì kích sáng ở trạng thái logic 0 và tắt ở trạng thái logic 1.
- Do c 3 bit đầu vào nên ta c thể mã h a 8 trạng thái khác nhau nhưng ch lấy 4 trạng thái đầu, các trạng thái c n lại không quan t m.

$c_2\ c_1\ c_0$	K tự
00	H
01	E
10	L
11	O
100	
101	
110	
111	

- Trong phần này ta nên áp dụng đơn giản mạch bằng giản đ Karnaugh khi thực hiện việc giải mã các k tự khi thực hiện bằng lệnh **assign**.

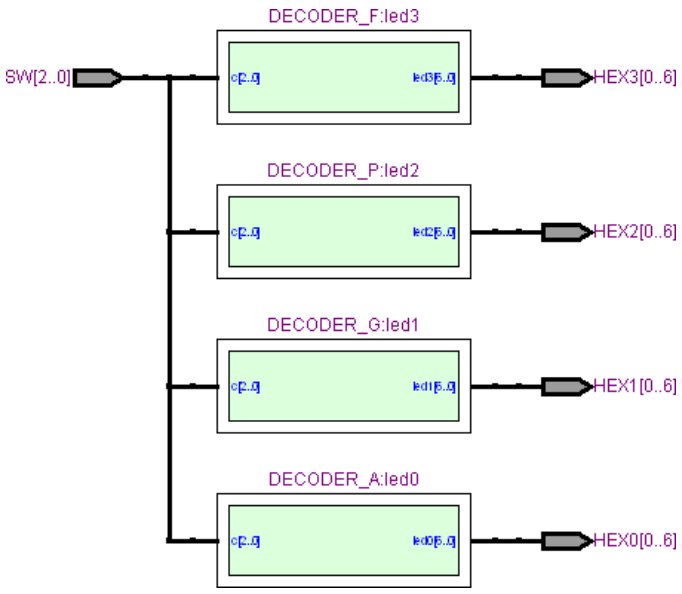
5. Thiết kế bộ giải mã để thực hiện việc hiển thị chữ FPGA trên 4 led 7 đoạn

HEX3 -> hiển thị chữ F

HEX2 -> hiển thị chữ P

HEX1 -> hiển thị chữ G

HEX0 -> hiển thị chữ A



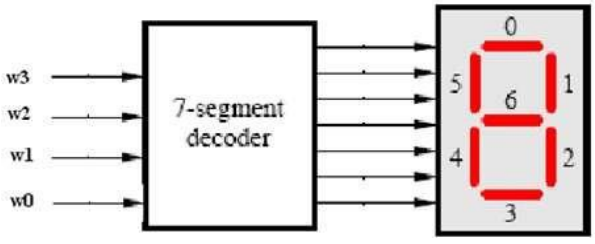
Hình 1.40: Hi n thị chữ FPGA trên 4 ed đo n

6. Mở rộng câu 5 chúng ta s hiển thị trên 4 led 7 đoạn 4 ký tự F,P,G,A và có thể xoay chữ thành một vòng dựa vào các SW điều ch nh. Dùng các switch SW_{9-7} điều khiển chữ xoay. Dùng các led 7 đoạn HEX3-HEX0 để hiển thị các ký tự.

Bảng 1.1: Giá trị hi n thị trên 4 ed đo n

SW9	**SW8**	**SW7**	**Ký tự hiển thị**			
			HEX3	HEX2	HEX1	HEX0
0	0	0	F	P	G	A
0	0	1	P	G	A	F
0	1	0	G	A	F	P
0	1	1	A	F	P	G

7. Thiết kế bộ giải mã số BCD sang LED 7 đoạn



Hình 1.41: Sơ đồ khối bộ giải BCD s ng LED đo n

Bảng 1.2: Bảng sự thật của bộ giải BCD s ng LED đo n

w3	w2	w1	w0	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	1	1	1	1
0	0	1	0	0	0	1	0	0	1	0
0	0	1	1	0	0	0	0	1	1	0
0	0	1	1	0	0	0	0	1	1	0
0	1	0	0	1	0	0	1	1	0	0
0	1	0	1	0	1	0	0	1	0	0
0	1	1	0	0	1	0	0	0	0	0
0	1	1	1	0	0	0	1	1	1	1
1	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	1	0	0
1	0	1	0	X	X	X	X	X	X	X
1	0	1	1	X	X	X	X	X	X	X
1	1	0	0	X	X	X	X	X	X	X
1	1	0	1	X	X	X	X	X	X	X
1	1	1	0	X	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X	X

Chúng ta sử dụng các SW[3:0] tương ng cho các tín hiệu vào w0, w1, w2, w3 và LED 7 đoạn HEX0 để thị giá trị thập phân.

CODE

Bài 1:

```
module cau1(SW, LEDR);
     input [9:0]SW;
     output [9:0]LEDR;
     assign LEDR = SW;
endmodule
```

Bài 2:

```
module cau2(SW,LEDR);
     input [9:0]SW;
     output [3:0]LEDR;
     mux_2_1(SW[3:0],SW[7:4], SW[9], LEDR[3:0]);
endmodule
module mux_2_1(x, y, s, m);
     input [3:0]x,y;
     input s;
     output [3:0]m;
     assign m[0] = (x[0] & ~s | y[0] & s);
     assign m[1] = (x[1] & ~s | y[1] & s);
     assign m[2] = (x[2] & ~s | y[2] & s);
     assign m[3] = (x[3] & ~s | y[3] & s);
endmodule
```

Bài 3:

```
module cau3(SW,LEDR);
     input [7:0]SW;
     output [0:0]LEDR;
     mux_2_1(SW[4:0],SW[7:5],  LEDR[0]);
endmodule
module mux_2_1(x, s, m_out);
```

```
    input [4:0]x;//
    input [2:0]s;
    output m_out;
    wire[3:1]m;
    assign m[1] = ( x[2] & ~s[0]  |  x[3] & s[0] );
    assign m[2] = ( x[0] & ~s[0]  |  x[1] & s[0] );
    assign m[3] = ( m[2] & ~s[1]  |  m[1] & s[1] );
    assign m_out = ( m[3] & ~s[2]  | x[4] & s[2] );
endmodule
```

Bài 4:

```
module cau4(SW, HEX0);
    input  [2:0]SW;
    output  [0:6]HEX0;
    dec3bit  dec(SW[2:0],HEX0);
endmodule
module  dec3bit(c,leds);
    input [2:0]c;
    output reg [0:6]leds;
    always @(c)
    case (c)     //abcdefg //0123456
        0: leds = 7'b1001000;//H
        1: leds = 7'b0110000;//E
        2: leds = 7'b1110001;//L
        3: leds = 7'b0000001;//O
        default: leds = 7'bx;
    endcase
endmodule
```

Bài 5:

```
module  cau5(SW, HEX0, HEX1, HEX2, HEX3);
    input [9:7] SW;
    output [0:6]HEX0, HEX1, HEX2, HEX3;
    DECODER_HEX0 (SW[9:7],HEX0);
    DECODER_HEX1 (SW[9:7],HEX1);
```

```
     DECODER_HEX2 (SW[9:7],HEX2);
     DECODER_HEX3 (SW[9:7],HEX3);
endmodule
module  DECODER_HEX0(c,led0);
     input [9:7]c;
     output reg [0:6]led0;
     always @(c)
     begin
          case (c)
          3'b 000 : led0 = 7'b0001000; //A
          default : led0 = 7'b1111111;
          endcase
     end
endmodule
module  DECODER_HEX1(c,led1);
     input [9:7]c;
     output reg [0:6]led1;
     always @(c)
     begin
          case(c)
          3'b 000 : led1 = 7'b0100001;//G
          default : led1 = 7'b1111111;
          endcase
     end
endmodule
module  DECODER_HEX2(c,led2);
     input [9:7]c;
     output reg [0:6]led2;
     always @(c)
     begin
          case(c)
          3'b 000 : led2 = 7'b0011000; //P
          default : led2 = 7'b1111111;
          endcase
     end
```

```
endmodule
module  DECODER_HEX3(c,led3);
     input [9:7]c;
     output reg [0:6]led3;
     always @(c)
     begin
          case(c)
          3'b 000 : led3 = 7'b0111000; //F
          default : led3 = 7'b1111111;
          endcase
     end
endmodule
```

Bài 6:

```
module  cau6(SW, HEX0, HEX1, HEX2, HEX3);
     input [9:7] SW;
     output [0:6]HEX0, HEX1, HEX2, HEX3;
     DECODER_HEX0 (SW[9:7],HEX0);
     DECODER_HEX1 (SW[9:7],HEX1);
     DECODER_HEX2 (SW[9:7],HEX2);
     DECODER_HEX3 (SW[9:7],HEX3);
endmodule
module  DECODER_HEX0 (c,led0);
     input [9:7]c;
     output reg [0:6]led0;
     always @(c)
     begin
          case (c)
          3'b 000 : led0 = 7'b0001000; //A
          3'b 011 : led0 = 7'b0100001;//G
          3'b 010 : led0 = 7'b0011000; //P
          3'b 001 : led0 = 7'b0111000; //F
          default : led0 = 7'b1111111;
          endcase
     end
```

```
endmodule
module  DECODER_HEX1 (c,led1);
     input [9:7]c;
     output reg [0:6]led1;
     always @(c)
     begin
          case (c)
          3'b 001 : led1 = 7'b0001000; //A
          3'b 000 : led1 = 7'b0100001;//G
          3'b 011 : led1 = 7'b0011000; //P
          3'b 010 : led1 = 7'b0111000; //F
          default : led1 = 7'b1111111;
          endcase
     end
endmodule
module  DECODER_HEX2 (c,led2);
     input [9:7]c;
     output reg [0:6]led2;
     always @(c)
     begin
          case (c)
          3'b 010 : led2 = 7'b0001000;//A
          3'b 001 : led2 = 7'b0100001;//G
          3'b 000 : led2 = 7'b0011000;//P
          3'b 011 : led2 = 7'b0111000;//F
          default : led2 = 7'b1111111;
          endcase
     end
endmodule
module  DECODER_HEX3 (c,led3);
     input [9:7]c;
     output reg [0:6]led3;
     always @(c)
     begin
          case (c)
```

```
          3'b 011 : led3 = 7'b0001000;//A
          3'b 010 : led3 = 7'b0100001;//G
          3'b 001 : led3 = 7'b0011000;//P
          3'b 000 : led3 = 7'b0111000;//F
          default : led3 = 7'b1111111;
          endcase
     end
endmodule
```

Bài 7:

```
module cau7(SW, HEX0);
     input [3:0] SW;
     output [0:6]HEX0;
     DECODER(SW[3:0],HEX0);
endmodule
module DECODER(c,led0);
     input [3:0]c;
     output reg [0:6]led0;
     always @(c)
     begin
          case (c)
          4'b 0000 : led0 = 7'b0000000;
          4'b 0001 : led0 = 7'b1001111;
          4'b 0010 : led0 = 7'b0010010;
          4'b 0011 : led0 = 7'b0000110;
          4'b 0100 : led0 = 7'b1001100;
          4'b 0101 : led0 = 7'b0100100;
          4'b 0110 : led0 = 7'b0100000;
          4'b 0111 : led0 = 7'b0001111;
          4'b 1000 : led0 = 7'b0000000;
          4'b 1001 : led0 = 7'b0000100;
          default : led0 = 7'b1111111;
          endcase
     end
endmodule
```

Bài tập tuần 2

Yêu cầu của bài thực hành

1. Hiển thị trên led 7 đoạn (HEX0) các số thập lục phân từ 0 – F được điều chỉnh từ các công tắc SW[3:0].

2. Hiển thị trên led 7 đoạn (HEX0, HEX1) các số thập lục phân từ 00 – FF được điều chỉnh từ các công tắc SW[3:0] -> HEX0 ; SW[7:4] -> HEX1

3. Thiết kế mạch dùng chuyển đổi số hệ nhị phân 4 bit V = v_3 v_2 v_1 v_0 thành số hệ thập phân D = d_1 d_0. Sử dụng các switch (SW 3:0) làm các ngõ vào và led 7 đoạn HEX0, HEX1 hiển thị số thập phân và đồng thời tắt các led 7 đoạn HEX2, HEX3. (Hay hiển thị số thập phân từ 00 – 15 trên 2 led 7 đoạn HEX0, HEX1)

4. Trong câu 3 chúng ta đã thực hiện một mạch chuyển số Binary thành số decimal. Trong phần này ta sẽ thiết kế mạch cộng hai số BCD và sử dụng mạch câu 3.

- Chọn các switch SW_{7-4}, SW_{3-0} và SW_8 làm ngõ vào cho hai số BCD và một bit nhớ c_{in}

- Ngõ ra là tổng của A và B gồm hai chữ số S_1S_0 thể hiện trên các led 7 đoạn HEX1, HEX0.

CODE

Câu 1:

```
module  TH_Bai1 (SW,HEX0);
 input [3:0]SW;
output [0:6]HEX0;
seg7 (SW[3:0],HEX0);
endmodule
module seg7 (bcd, leds);
input [3:0] bcd;
output reg [0:6]leds;
always @(bcd)
case (bcd)       //abcdefg
0: leds = 7'b0000001;
1: leds = 7'b1001111;
2: leds = 7'b0010010;
3: leds = 7'b0000110;
4: leds = 7'b1001100;
5: leds = 7'b0100100;
6: leds = 7'b0100000;
```

```
7: leds = 7'b0001111;
8: leds = 7'b0000000;
9: leds = 7'b0000100;
10: leds  = 7'b0001000;
11 : leds= 7'b0000000;
12 : leds= 7'b0110001;
13 : leds= 7'b0000001;
14 : leds= 7'b0110000;
15 : leds= 7'b0111000;
default: leds = 7‘b1111111;
endcase
endmodule
```

Bài 2:

```
module  TH_Bai1 (SW,HEX0, HEX1);
input [7:0]SW;
output [0:6]HEX0;
output [0:6]HEX1;
seg70 (SW[3:0],HEX0);
seg71 (SW[7:4],HEX1);
endmodule
module seg70 (bcd, led0);
input [3:0] bcd;
output reg [0:6]led0;
always @(bcd)
case (bcd)       //abcdefg
0: led0 = 7'b0000001;
1: led0 = 7'b1001111;
2: led0 = 7'b0010010;
3: led0 = 7'b0000110;
4: led0 = 7'b1001100;
5: led0 = 7'b0100100;
6: led0 = 7'b0100000;
7: led0= 7'b0001111;
8: led0= 7'b0000000;
```

```
9: led0= 7'b0000100;
10: led0  = 7'b0001001;
11 : led0= 7'b0000000;
12 : led0= 7'b0110001;
13 : led0= 7'b0000001;
14 : led0= 7'b0110000;
15 : led0= 7'b0111000;
default: led0 = 7'b1111111;
endcase

endmodule
module seg71 (bcd, led1);
input [7:4] bcd;
output reg [0:6]led1;
always @(bcd)
case (bcd)       //abcdefg
0: led1 = 7'b0000001;
1: led1 = 7'b1001111;
2: led1 = 7'b0010010;
3: led1 = 7'b0000110;
4: led1 = 7'b1001100;
5: led1 = 7'b0100100;
6: led1 = 7'b0100000;
7: led1= 7'b0001111;
8: led1= 7'b0000000;
9: led1= 7'b0000100;
10: led1  = 7'b0001001;
11 : led1= 7'b0000000;
12 : led1= 7'b0110001;
13 : led1= 7'b0000001;
14 : led1= 7'b0110000;
15 : led1= 7'b0111000;
default: led1 = 7'b1111111;
endcase
endmodule
```

Câu 3:

```
module  B1 (SW,HEX0,HEX1);
     input [3:0]SW;
     output [0:6]HEX0,HEX1;
     seg70 (SW[3:0],HEX0,HEX1);
endmodule
module seg70 (bcd, led0,led1);
     input [3:0] bcd;
     output reg [0:6]led0,led1;
     always @(bcd)
     if (bcd<10)
     begin
     led1=7'b1111111;
          case (bcd)       //abcdefg
          0: led0 = 7'b0000001;
          1: led0 = 7'b1001111;
          2: led0 = 7'b0010010;
          3: led0 = 7'b0000110;
          4: led0 = 7'b1001100;
          5: led0 = 7'b0100100;
          6: led0 = 7'b0100000;
          7: led0= 7'b0001111;
          8: led0= 7'b0000000;
          9: led0= 7'b0000100;
          default: led0 = 7'b1111111 ;
          endcase
     end
     else
     begin
     led1=7'b1001111;
          case (bcd)
          10: led0 = 7'b0000001;
          11: led0 = 7'b1001111;
          12: led0 = 7'b0010010;
          13: led0 = 7'b0000110;
```

```
          14: led0 = 7'b1001100;
          15: led0 = 7'b0100100;
          endcase
     end
endmodule
```

Câu 4:

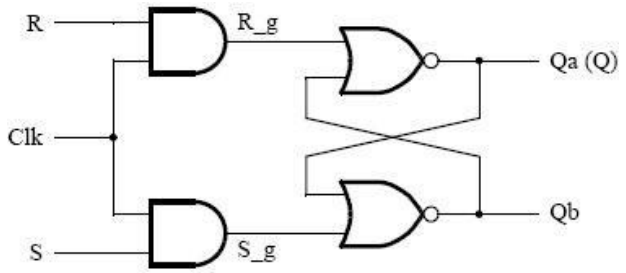
```
module  B1 (SW,HEX0,HEX1);
     input [8:0]SW;
     output [0:6]HEX0,HEX1;
     wire [3:0]s;
     wire count;
     adder4bit  (SW[3:0],SW[7:4],SW[8],s,count);
     display_0_9 (s,HEX0);
     display_0_9 (count,HEX1);
endmodule
module adder4bit (a,b,cin,BCDout,carryout);
     input [3:0] a,b;
     input cin;
     output reg [0:3]BCDout;
     output reg carryout;
     reg [4:0]z;
     always @(a,b,cin)
     begin
     z=a+b+cin;
     if (z<10)
     {carryout,BCDout}=z;
     else
     {carryout,BCDout}=z+6;
     end
endmodule
module display_0_9(c,hex);
     input [3:0]c;
     output reg [6:0]hex;
     always @(c)
```

```
            case (c)       //abcdefg
            0: hex = 7'b0000001;
            1: hex = 7'b1001111;
            2: hex = 7'b0010010;
            3: hex = 7'b0000110;
            4: hex = 7'b1001100;
            5: hex = 7'b0100100;
            6: hex = 7'b0100000;
            7: hex= 7'b0001111;
            8: hex= 7'b0000000;
            9: hex= 7'b0000100;
            default: hex = 7'b1111111 ;
      endcase
endmodule
```

Bài tập tuần 3

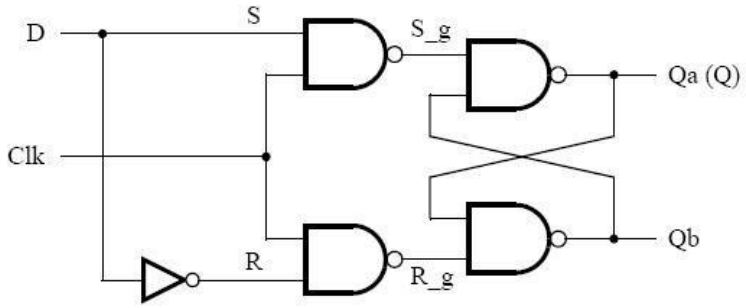
Yêu cầu của bài thực hành

Câu 1. Thực hiện module tạo ra mạch chốt RS, trong đó sử dụng SW[0] và SW[1] cho 2 tín hiệu ngõ vào R và S, SW[2] cho tín hiệu ngõ vào Clk. Kết nối tín hiệu ngõ ra Qa với LEDR0.



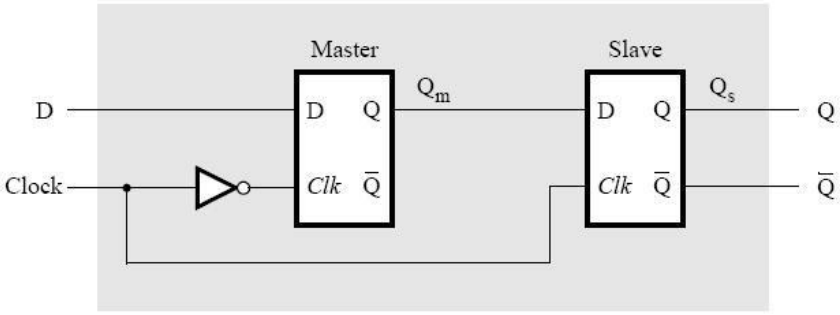
Hình 1.46: Mạch chốt RS [22]

Câu 2. Thực hiện module tạo ra mạch chốt D (D latch)



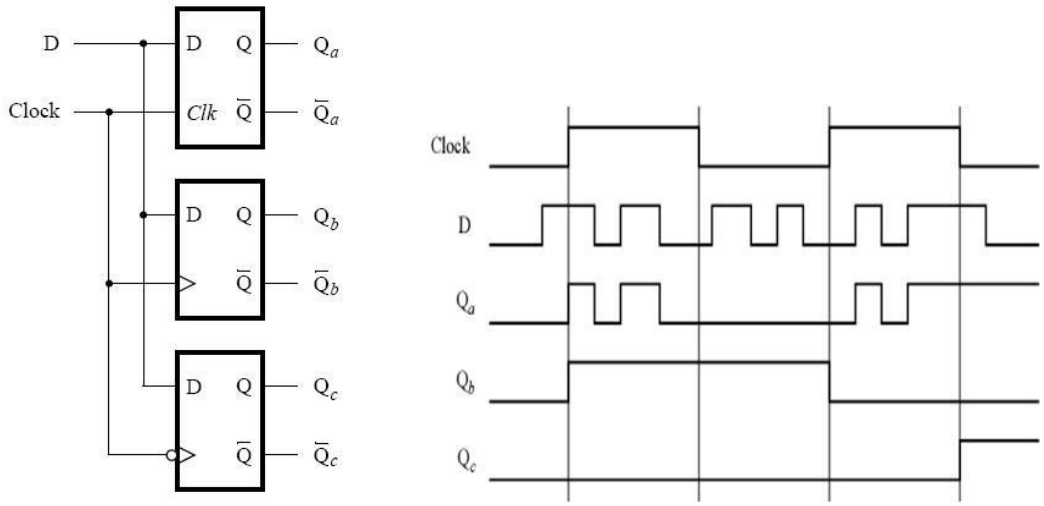
Hình 1.47: Mạch chốt D [22]

Câu 3. Thực hiện module tạo ra mạch master-slave D flipflop.



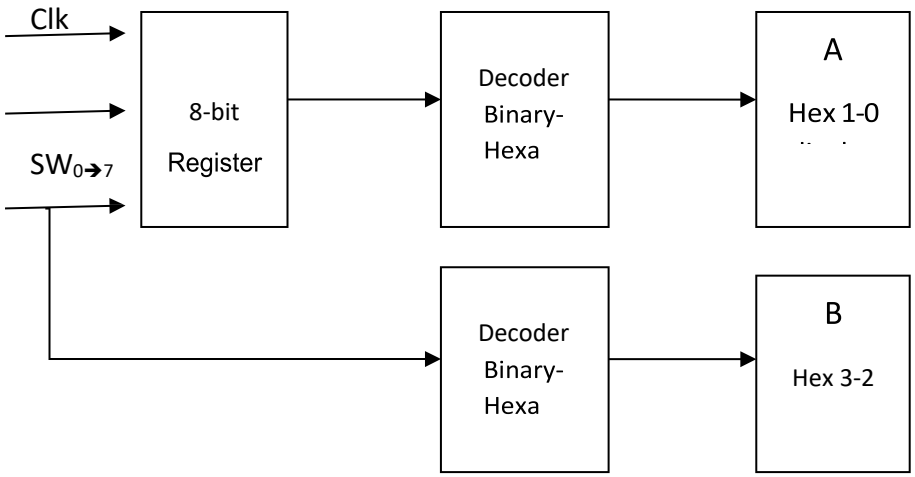
Hình 1.48: Mạch master-slave D flipflop [22]

Câu 4. Thực hiện module tạo ra các mạch flipflop D khác nhau theo hình vẽ sau:



Hình 1.49: Mạch flipflop D [22]

Câu 5. Thiết kế các module sau: một bộ giải mã số hexa 0➔ F, thiết kế thanh ghi 8 bit. Ta dùng hết 4 đèn led 7 đoạn trên board DE1, thanh ghi 8 bit ngõ ra kết nối với bộ giải mã số hexa cho 2 đèn led đầu, và 8 bit ngõ vào điều khiển 2 đèn led 7 đoạn còn lại.



Hình 1.50: Mạch hiển thị số HEX sử dụng thanh ghi 8 bit

CODE

Câu 1:

```
     module cau1(SW, LEDR);
          input [2:0]SW;
          output [1:0]LEDR;
          RS_FF (SW[0], SW[1], SW[2], LEDR[0], LEDR[1]);
endmodule

module RS_FF(R, S, clk, Qa, Qb);
```

```
        input R, S, clk;
        output wire Qa, Qb;
        wire R_g, S_g;
        and(R_g, R, clk);
        and(S_g, S, clk);
        nor(Qa, R_g, Qb);
        nor(Qb, S_g, Qa);
endmodule
```

Câu 2:

```
    module cau2(SW,LEDR);
        input [1:0]SW;
        output [1:0]LEDR;
        D_LATCH (LEDR[0], LEDR[1], SW[0],SW[1]);
        // led0-Qa..led1-Qb..sw0-clk..sw1..D
endmodule

module D_LATCH(Qa, Qb, clk, D);
        input D, clk;
        output wire Qa, Qb;
        wire R_g, S_g;
        nand (S_g, D, clk);
        nand (R_g, ~D, clk);
        nand (Qa, S_g, Qb);
        nand (Qb, R_g, Qa);
endmodule
```

Câu 3:

```
    module cau3(SW, LEDR);
        input [1:0]SW;
        output [0:0]LEDR;
        wire Qm;
        // sw1-clk..sw0-D..led0-Q
        FF_D(Qm, SW[0], ~SW[1]);
        FF_D(LEDR[0], Qm, SW[1]);
```

```
endmodule

module FF_D(Q, D, Clk);
          input Clk, D;
          output reg Q;
          always@(posedge Clk)
          begin
               if(D == 1'b1)
                    Q <= 1'b1;
               else
                    Q <= 1'b0;
          end
endmodule
```

Câu 4:

```
     module cau4(SW, LEDR);
          input [1:0]SW;
          output [2:0]LEDR;
          FF_D_Ex(SW[1], SW[0], LEDR[0], LEDR[1], LEDR[2]);
endmodule

module FF_D_Ex(d, clk, q1, q2, q3);
          input d, clk;
          output reg q1, q2, q3;
          always @(clk)
               if(clk)
                    q1 <= d;
          always @(posedge clk)
               q2 <= d;
          always @(negedge clk)
               q3 <= d;
endmodule
```

Câu 5:

```
     module cau5(SW,KEY,HEX0,HEX1,HEX2,HEX3);
```

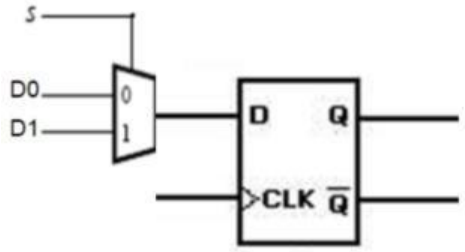
```
        input [7:0]SW;
        input [1:0]KEY;
        output [6:0]HEX0,HEX1,HEX2,HEX3;
        wire [7:0]Q;
        Decoder_HEX (SW[3:0],HEX3);
        Decoder_HEX (SW[7:4],HEX2);
        D_LATCH (SW[3:0],KEY[0],KEY[1],Q[3:0]);
        Decoder_HEX(Q[3:0],HEX1);
        D_LATCH (SW[7:4],KEY[0],KEY[1],Q[7:4]);
        Decoder_HEX(Q[7:4],HEX0);
endmodule

module  D_LATCH(D, clk, Reset, Q);
        input D, clk, Reset;
        output reg Q;
        always@(posedge clk, negedge Reset)
        if(~Reset)
            Q <= 0;
        else
            Q <= D;
endmodule

module Decoder_HEX(c, hex);
        input [3:0]c;
        output reg [6:0]hex;
        always @(c)
        case (c)       //abcdefg
            0: hex = 7'b0000001;
            1: hex = 7'b1001111;
            2: hex = 7'b0010010;
            3: hex = 7'b0000110;
            4: hex = 7'b1001100;
            5: hex = 7'b0100100;
            6: hex = 7'b0100000;
            7: hex= 7'b0001111;
```

```
                8: hex= 7'b0000000;
                9: hex= 7'b0000100;
                default: hex = 7'b1111111;
            endcase
endmodule
```

Bài tập tuần 4

Yêu cầu của bài thực hành

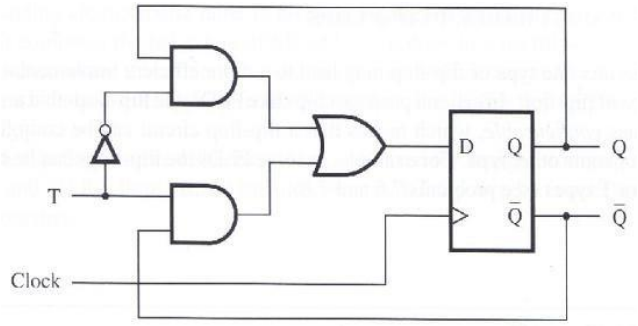
Câu 1. Viết code Verilog mô tả mạch Flip-Flop D với Mux 2:1 ở ngõ vào D 1bit như sau:



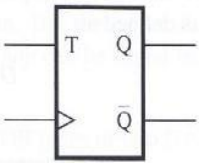
Kiểm tra mạch trên Board DE1 bằng cách sử dụng các tín hiệu:
- SW[0] = D0
- SW[1] = D1
- SW[2] = s
- LEDR[0] = Q

Câu 2. Mở rộng câu 1, viết code Verilog mô tả mạch Flip-Flop D với Mux 2:1 ở ngõ vào D 4bits. Kiểm tra mạch trên Board DE1 bằng cách sử dụng các tín hiệu:
- SW[3:0] = D0
- SW[7:4] = D1
- SW[8] = s
- LEDR[3:0] = Q

Câu 3. Viết code Verilog mô tả mạch Flip-Flop T như sau:

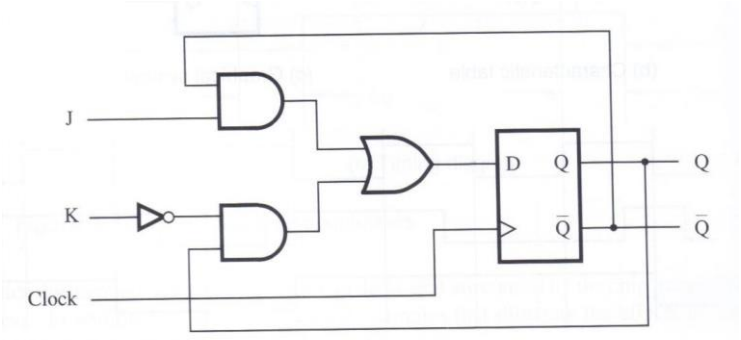


T	Q(t + 1)
0	Q(t)
1	$\bar{Q}(t)$



Kiểm tra mạch Flip-Flop T trên Board DE1 bằng cách sử dụng các tín hiệu:
- SW[0] = T
- KEY[0] = Clk
- LEDR[0] = Q

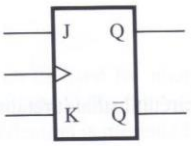
Câu 4. Viết code Verilog mô tả mạch Flip-Flop JK như sau:



(a) Circuit

J	K	Q(t + 1)
0	0	Q(t)
0	1	0
1	0	1
1	1	Q̄(t)

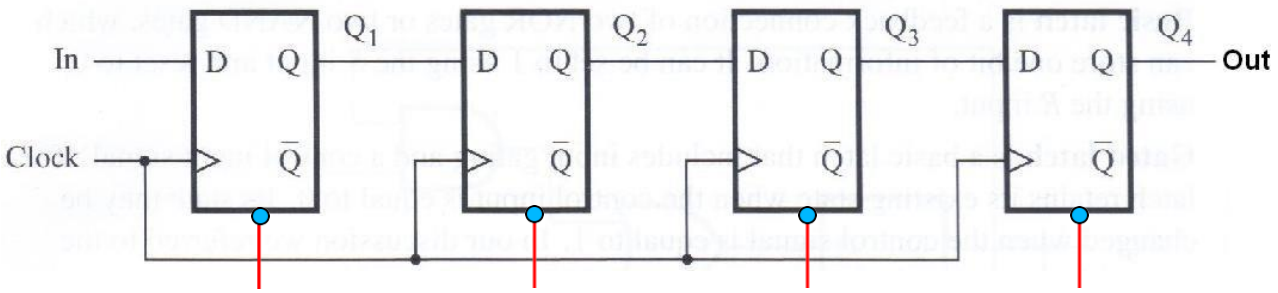
(b) Characteristic table



(c) Graphical symbol

Kiểm tra mạch Flip-Flop JK trên Board DE1 bằng cách sử dụng các tín hiệu:
- SW[0] = J
- SW[1] = K
  - KEY[0] = Clk
  - LEDR[0] = Q

Câu 5. Thiết kế mạch thanh ghi dịch như sau:



Sử dụng: SW[0] là tín hiệu vào In;
KEY[0] làm Clock; KEY[1] làm Clear;
LERD[3:0] hiển thị giá trị của Q[4:1]

CODE

Bài 1:

```
module cau1(SW, KEY, LEDR);
    input [2:0]SW; // SW0 = D0, SW1 = D1, SW2 = s
    input [0:0]KEY;
    output [0:0]LEDR; // LEDR0 = q
    wire d;
```

```
    mux_21(SW[0], SW[1], SW[2], d);
    FF_D(d, KEY[0], LEDR[0]);
endmodule

module mux_21(x, y, s, m);
    input x, y, s;
    output m;

    assign m = s?y:x;
endmodule

module FF_D(d, clk, q);
    input d, clk;
    output reg q;

    always @(posedge clk)
        q <= d;
endmodule
```

Bài 2:

```
module cau2(SW, KEY, LEDR);
    input [8:0]SW; // SW[3:0] = D0, SW[7:4] = D1, SW[8] = s
    input [0:0]KEY;
    output wire [3:0]LEDR; // LEDR[3:0] = q
    wire [3:0]d;

    mux_21_4b(SW[3:0], SW[7:4], SW[8], d);
    FF_D_4b(d, KEY[0], LEDR[3:0]);
endmodule

module mux_21_4b(x, y, s, m);
    input [3:0]x, y;
    input s;
```

```
    output [3:0]m;

    assign m = s?y:x;
endmodule

module FF_D_4b(d, clk, q);
    input [3:0]d;
    input clk;
    output reg [3:0]q;

    always @(posedge clk)
        q <= d;
endmodule
```

Bài 3:

```
module cau3(SW, KEY, LEDR);
    input [0:0]SW, KEY; // t = SW0, clk = KEY0
    output [0:0]LEDR;   // q = LEDR

    FF_T(SW[0], KEY[0], LEDR[0]);
endmodule

module FF_T(t, clk, q);
    input t, clk;
    output q;
    wire m1, m2, d;

    and(m1, ~t, q);
    and(m2, t, ~q);
    or(d, m1, m2);
    FF_D(d, clk, q);
endmodule

module FF_D(d, clk, q);
    input d, clk;
```

```
    output reg q;

    always @(posedge clk)
        q <= d;
endmodule
```

Bài 4:

```
module cau4(SW, KEY, LEDR);
    input [1:0]SW;
    input [0:0]KEY; // j = SW0, k = SW1, clk = KEY0
    output [0:0]LEDR;   // q = LEDR

    FF_T(SW[0], SW[1], KEY[0], LEDR[0]);
endmodule

module FF_T(j, k, clk, q);
    input j, k, clk;
    output q;
    wire m1, m2, d;

    and(m1, j, ~q);
    and(m2, ~k, q);
    or(d, m1, m2);
    FF_D(d, clk, q);
endmodule

module FF_D(d, clk, q);
    input d, clk;
    output reg q;

    always @(posedge clk)
        q <= d;
endmodule
```

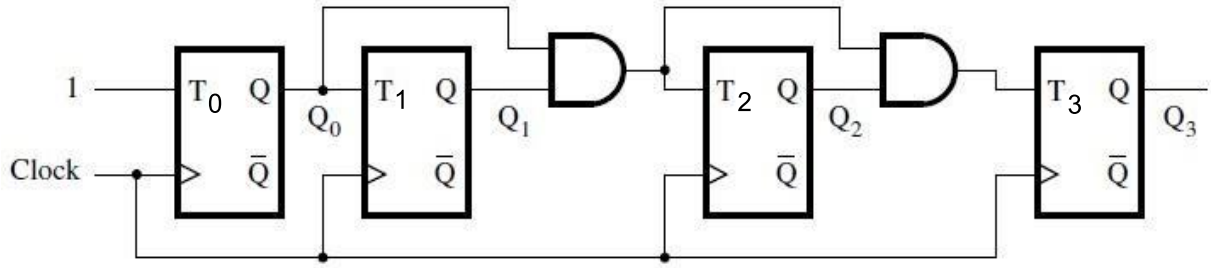
Bài 5:

```
module cau5(SW, KEY, LEDR);
    input [0:0]SW; // d = SW0
    input [1:0]KEY;// clk = KEY0, r = KEY1
    output [3:0]LEDR; // q = LEDR0
    wire q1, q2, q3;

    FF_D(SW[0], KEY[1], KEY[0], LEDR[0]);
    FF_D(LEDR[0], KEY[1], KEY[0], LEDR[1]);
    FF_D(LEDR[1], KEY[1], KEY[0], LEDR[2]);
    FF_D(LEDR[2], KEY[1], KEY[0], LEDR[3]);
endmodule

module FF_D(d, r, clk, q);
    input d, r, clk;
    output reg q;

    always @(posedge clk)
        if(r == 0)
            q <= 0;
        else
            q <= d;
endmodule
```

Bài tập tuần 5

Yêu cầu của bài thực hành

Câu 1. Viết code verilog mô tả mạch đếm lên đồng bộ 4 bit sử dụng FlipFlop T. Bộ đếm sẽ hoạt động tại mỗi cạnh lên của xung clock, ngõ vào T_0 luôn được đặt ở mức cao. Sử dụng led 7 đoạn để hiển thị kết quả bộ đếm lên từ 0 -> F.



Yêu cầu sử dụng các tín hiệu trên board như sau:
- SW[0] làm tín hiệu ngõ vào T_0
- KEY[0] làm tín hiệu xung clock
- HEX0 hiển thị kết quả của bộ đếm lên từ 0 -> F

Câu 2: Viết code verilog mô tả mạch đếm lên đồng bộ 4 bit sử dụng FlipFlop T. Bộ đếm sẽ hoạt động tại mỗi cạnh lên của xung clock khi tín hiệu cho phép (enable) được đặt lên mức cao. Sử dụng tín hiệu Clear để trả bộ đếm về 0. Sử dụng led 7 đoạn để hiển thị kết quả bộ đếm lên từ 0 -> F.
Yêu cầu sử dụng các tín hiệu trên board như sau:
- SW[0] làm tín hiệu Enable (hay T_0).
- KEY[0] làm tín hiệu xung clock, KEY[1] làm tín hiệu clear bộ đếm về 0.
- HEX0 hiển thị kết quả của bộ đếm lên từ 0 -> F

Câu 3. Mở rộng từ câu 2, viết code verilog mô tả mạch đếm lên đồng bộ 8 bit sử dụng FlipFlop T.
Yêu cầu sử dụng các tín hiệu trên board như sau:
- SW[0] làm tín hiệu Enable (hay T_0).
- KEY[0] làm tín hiệu xung clock, KEY[1] làm tín hiệu clear bộ đếm về 0.
- HEX1, HEX0 hiển thị kết quả của bộ đếm lên từ 00 -> FF.Câu 4. Viết code verilog mô tả mạch đếm xuống đồng bộ 4 bit sử dụng FlipFlop T. Bộ đếm sẽ hoạt động tại mỗi cạnh lên của xung clock khi tín hiệu cho phép (enable) được đặt lên mức cao. Sử dụng tín hiệu Clear để trả bộ đếm về 0. Sử dụng led 7 đoạn để hiển thị kết quả bộ đếm xuống từ F -> 0.
Yêu cầu sử dụng các tín hiệu trên board như sau:
- SW[0] làm tín hiệu Enable (hay T_0).
- KEY[0] làm tín hiệu xung clock, KEY[1] làm tín hiệu clear bộ đếm về 0.
- HEX0 hiển thị kết quả của bộ đếm xuống từ F -> 0

Câu 5. Viết code verilog mô tả mạch đếm lên 4 bit (tương tự câu 2) sử dụng biểu thức toán học thay cho FlipFlop T.

Câu 6. Viết code verilog mô tả mạch đếm xuống 4 bit (tương tự câu 4) sử dụng biểu thức toán học thay cho FlipFlop T.

Câu 7. Viết code verilog mô tả mạch đếm lên/xuống 4 bit sử dụng biểu thức toán học, cho phép chọn chức năng mạch đếm (dùng biến Select). Nếu Select = 1 thì mạch đếm lên, ngược lại (Select = 0) thì mạch đếm xuống.
Yêu cầu sử dụng các tín hiệu trên board như sau:
- SW[0] làm tín hiệu Enable.
- SW[1] làm tín hiệu Select.
- KEY[0] làm tín hiệu xung clock, KEY[1] làm tín hiệu clear bộ đếm về 0.
- HEX0 hiển thị kết quả của bộ đếm xuống từ 0 <-> F

Câu 8. Mở rộng từ câu 7, viết code verilog mô tả mạch đếm lên/xuống đồng bộ 8 bit sử dụng biểu thức toán học.
Yêu cầu sử dụng các tín hiệu trên board như sau:
- SW[0] làm tín hiệu Enable.
- SW[1] làm tín hiệu Select.
- KEY[0] làm tín hiệu xung clock, KEY[1] làm tín hiệu clear bộ đếm về 0.
- HEX1, HEX0 hiển thị kết quả của bộ đếm lên từ 00 <--> FF

CODE

Bài 1:

```
module cau1(SW, KEY, HEX0);
    input [0:0]SW;
    input [0:0]KEY;
    output [6:0]HEX0;
    wire t2, t3;
    wire [3:0]q;

    FF_T(SW[0], KEY[0], q[0]);
    FF_T(q[0], KEY[0], q[1]);
    and(t2, q[1], q[0]);
    FF_T(t2, KEY[0], q[2]);
    and(t3, q[2], t2);
    FF_T(t3, KEY[0], q[3]);
    Decoder_HEX(q, HEX0);
endmodule

module FF_T(t, clk, q);
    input t, clk;
    output q;
    wire m1, m2, d;

    and(m1, ~t, q);
    and(m2, t, ~q);
    or(d, m1, m2);
    FF_D(d, clk, q);
endmodule
```

```
module FF_D(d, clk, q);
    input d, clk;
    output reg q;

    always @(posedge clk)
        q <= d;
endmodule

module Decoder_HEX(c,HEX);
    input[3:0]c;
    output [6:0]HEX;
    reg[6:0]HEX;
    always@(c)
        case(c)
        4'b0000: HEX = 7'b1000000;
        4'b0001: HEX = 7'b1111001;
        4'b0010: HEX = 7'b0100100;
        4'b0011: HEX = 7'b0110000;
        4'b0100: HEX = 7'b0011001;
        4'b0101: HEX = 7'b0010010;
        4'b0110: HEX = 7'b0000010;
        4'b0111: HEX = 7'b1111000;
        4'b1000: HEX = 7'b0000000;
        4'b1001: HEX = 7'b0010000;
        4'b1010: HEX = 7'b0001000; //A
        4'b1011: HEX = 7'b0000011; //B
        4'b1100: HEX = 7'b1000110; //C
        4'b1101: HEX = 7'b0100001; //D
        4'b1110: HEX = 7'b0000110; //E
        4'b1111: HEX = 7'b0001110; //F
        default: HEX = 7'b1111111;
    endcase
endmodule
```

Bài 2:

```
module cau2(SW, KEY, HEX0);
    input [0:0]SW;   // T
    input [1:0]KEY;  // clk = key0, rst = key1
    output [6:0]HEX0;
    wire [3:0]q, t;

    FF_T(SW[0], KEY[1], KEY[0], q[0]);
    and(t[1], SW[0], q[0]);
    FF_T(t[1], KEY[1], KEY[0], q[1]);
    and(t[2], t[1], q[1]);
    FF_T(t[2], KEY[1], KEY[0], q[2]);
    and(t[3], t[2], q[2]);
    FF_T(t[3], KEY[1], KEY[0], q[3]);
    Decoder_HEX(q, HEX0);
endmodule

module FF_T(t, rst, clk, q);
    input t, rst, clk;
    output reg q;

    always@(posedge clk or negedge rst)
        if(!rst)
            q <= 1'b0;
        else
            if(t)
                q <= ~q;
endmodule

module Decoder_HEX(c,HEX);
    input[3:0]c;
    output [6:0]HEX;
    reg[6:0]HEX;
    always@(c)
        case(c)
```

```
        4'b0000: HEX = 7'b1000000;
        4'b0001: HEX = 7'b1111001;
        4'b0010: HEX = 7'b0100100;
        4'b0011: HEX = 7'b0110000;
        4'b0100: HEX = 7'b0011001;
        4'b0101: HEX = 7'b0010010;
        4'b0110: HEX = 7'b0000010;
        4'b0111: HEX = 7'b1111000;
        4'b1000: HEX = 7'b0000000;
        4'b1001: HEX = 7'b0010000;
        4'b1010: HEX = 7'b0001000; //A
        4'b1011: HEX = 7'b0000011; //B
        4'b1100: HEX = 7'b1000110; //C
        4'b1101: HEX = 7'b0100001; //D
        4'b1110: HEX = 7'b0000110; //E
        4'b1111: HEX = 7'b0001110; //F
        default: HEX = 7'b1111111;
    endcase
endmodule
```

Bài 3:

```
module cau3(SW, KEY, HEX0, HEX1);
    input [0:0]SW;   // T
    input [1:0]KEY;  // clk = key0, rst = key1
    output [6:0]HEX0, HEX1;
    wire [7:0]q, t;

    // 4 High bits
    FF_T(SW[0], KEY[1], KEY[0], q[0]);
    and(t[1], SW[0], q[0]);
    FF_T(t[1], KEY[1], KEY[0], q[1]);
    and(t[2], t[1], q[1]);
    FF_T(t[2], KEY[1], KEY[0], q[2]);
    and(t[3], t[2], q[2]);
```

```
    FF_T(t[3], KEY[1], KEY[0], q[3]);
    // 4 Low bits
    and(t[4], t[3], q[3]);
    FF_T(t[4], KEY[1], KEY[0], q[4]);
    and(t[5], t[4], q[4]);
    FF_T(t[5], KEY[1], KEY[0], q[5]);
    and(t[6], t[5], q[5]);
    FF_T(t[6], KEY[1], KEY[0], q[6]);
    and(t[7], t[6], q[6]);
    FF_T(t[7], KEY[1], KEY[0], q[7]);


    Decoder_HEX(q[3:0], HEX0);
    Decoder_HEX(q[7:4], HEX1);
endmodule

module FF_T(t, rst, clk, q);
    input t, rst, clk;
    output reg q;

    always@(posedge clk or negedge rst)
        if(!rst)
            q <= 1'b0;
        else
            if(t)
                q <= ~q;
endmodule

module Decoder_HEX(c,HEX);
    input[3:0]c;
    output [6:0]HEX;
    reg[6:0]HEX;
    always@(c)
        case(c)
        4'b0000: HEX = 7'b1000000;
```

```
        4'b0001: HEX = 7'b1111001;
        4'b0010: HEX = 7'b0100100;
        4'b0011: HEX = 7'b0110000;
        4'b0100: HEX = 7'b0011001;
        4'b0101: HEX = 7'b0010010;
        4'b0110: HEX = 7'b0000010;
        4'b0111: HEX = 7'b1111000;
        4'b1000: HEX = 7'b0000000;
        4'b1001: HEX = 7'b0010000;
        4'b1010: HEX = 7'b0001000; //A
        4'b1011: HEX = 7'b0000011; //B
        4'b1100: HEX = 7'b1000110; //C
        4'b1101: HEX = 7'b0100001; //D
        4'b1110: HEX = 7'b0000110; //E
        4'b1111: HEX = 7'b0001110; //F
        default: HEX = 7'b1111111;
    endcase
endmodule
```

Bài 4:

```
module cau4(SW, KEY, HEX0);
    input [0:0]SW;   // T
    input [1:0]KEY;  // clk = key0, rst = key1
    output [6:0]HEX0;
    wire [3:0]q, t;

    FF_T(SW[0], KEY[1], KEY[0], q[0]);
    and(t[1], SW[0], ~q[0]);
    FF_T(t[1], KEY[1], KEY[0], q[1]);
    and(t[2], t[1], ~q[1]);
    FF_T(t[2], KEY[1], KEY[0], q[2]);
    and(t[3], t[2], ~q[2]);
    FF_T(t[3], KEY[1], KEY[0], q[3]);
    Decoder_HEX(q, HEX0);
endmodule
```

```
module FF_T(t, rst, clk, q);
    input t, rst, clk;
    output reg q;

    always@(posedge clk or negedge rst)
        if(!rst)
            q <= 1'b0;
        else
            if(t)
                q <= ~q;
endmodule

module Decoder_HEX(c,HEX);
    input[3:0]c;
    output [6:0]HEX;
    reg[6:0]HEX;
    always@(c)
        case(c)
        4'b0000: HEX = 7'b1000000;
        4'b0001: HEX = 7'b1111001;
        4'b0010: HEX = 7'b0100100;
        4'b0011: HEX = 7'b0110000;
        4'b0100: HEX = 7'b0011001;
        4'b0101: HEX = 7'b0010010;
        4'b0110: HEX = 7'b0000010;
        4'b0111: HEX = 7'b1111000;
        4'b1000: HEX = 7'b0000000;
        4'b1001: HEX = 7'b0010000;
        4'b1010: HEX = 7'b0001000; //A
        4'b1011: HEX = 7'b0000011; //B
        4'b1100: HEX = 7'b1000110; //C
        4'b1101: HEX = 7'b0100001; //D
        4'b1110: HEX = 7'b0000110; //E
        4'b1111: HEX = 7'b0001110; //F
```

```
        default: HEX = 7'b1111111;
    endcase
endmodule
```

Bài 5:

```
module cau5(SW, KEY, HEX0);
    input [0:0]SW;
    input [1:0]KEY;
    output [6:0]HEX0;
    wire [3:0]bcd;

    counter(SW[0], KEY[1], KEY[0], bcd);
    Decoder_HEX(bcd, HEX0);
endmodule

module counter(en, rst, clk, bcd);
    parameter n = 4;
    input en, rst, clk;
    output reg [n-1:0]bcd;

    always@(posedge clk or negedge rst)
        if(!rst)
            bcd <= 0;
        else if(en)
            bcd <= bcd + 1;
endmodule

module Decoder_HEX(c,HEX);
    input[3:0]c;
    output [6:0]HEX;
    reg[6:0]HEX;
    always@(c)
        case(c)
        4'b0000: HEX = 7'b1000000;
```

```
        4'b0001: HEX = 7'b1111001;
        4'b0010: HEX = 7'b0100100;
        4'b0011: HEX = 7'b0110000;
        4'b0100: HEX = 7'b0011001;
        4'b0101: HEX = 7'b0010010;
        4'b0110: HEX = 7'b0000010;
        4'b0111: HEX = 7'b1111000;
        4'b1000: HEX = 7'b0000000;
        4'b1001: HEX = 7'b0010000;
        4'b1010: HEX = 7'b0001000; //A
        4'b1011: HEX = 7'b0000011; //B
        4'b1100: HEX = 7'b1000110; //C
        4'b1101: HEX = 7'b0100001; //D
        4'b1110: HEX = 7'b0000110; //E
        4'b1111: HEX = 7'b0001110; //F
        default: HEX = 7'b1111111;
    endcase
endmodule
```

Bài 6:

```
module cau6(SW, KEY, HEX0);
    input [0:0]SW;
    input [1:0]KEY;
    output [6:0]HEX0;
    wire [3:0]bcd;

    counter(SW[0], KEY[1], KEY[0], bcd);
    Decoder_HEX(bcd, HEX0);
endmodule

module counter(en, rst, clk, bcd);
    parameter n = 4;
    input en, rst, clk;
    output reg [n-1:0]bcd;
```

```
    always@(posedge clk or negedge rst)
        if(!rst)
            bcd <= 0;
        else if(en)
            bcd <= bcd - 1;
endmodule

module Decoder_HEX(c,HEX);
    input[3:0]c;
    output [6:0]HEX;
    reg[6:0]HEX;
    always@(c)
        case(c)
        4'b0000: HEX = 7'b1000000;
        4'b0001: HEX = 7'b1111001;
        4'b0010: HEX = 7'b0100100;
        4'b0011: HEX = 7'b0110000;
        4'b0100: HEX = 7'b0011001;
        4'b0101: HEX = 7'b0010010;
        4'b0110: HEX = 7'b0000010;
        4'b0111: HEX = 7'b1111000;
        4'b1000: HEX = 7'b0000000;
        4'b1001: HEX = 7'b0010000;
        4'b1010: HEX = 7'b0001000; //A
        4'b1011: HEX = 7'b0000011; //B
        4'b1100: HEX = 7'b1000110; //C
        4'b1101: HEX = 7'b0100001; //D
        4'b1110: HEX = 7'b0000110; //E
        4'b1111: HEX = 7'b0001110; //F
        default: HEX = 7'b1111111;
    endcase
endmodule
```

Bài 7:

```
module cau7(SW, KEY, HEX0);
    input [1:0]SW;
    input [1:0]KEY;
    output [6:0]HEX0;
    wire [3:0]bcd;

    counter(SW[0], SW[1], KEY[1], KEY[0], bcd);
    Decoder_HEX(bcd, HEX0);
endmodule

module counter(en, s, rst, clk, bcd);
    parameter n = 4;
    input s, en, rst, clk;
    output reg [n-1:0]bcd;

    always@(posedge clk or negedge rst)
        if(!rst)
            bcd <= 0;
        else if(en)
            bcd <= bcd + (s?1:-1);
endmodule

module Decoder_HEX(c,HEX);
    input[3:0]c;
    output [6:0]HEX;
    reg[6:0]HEX;
    always@(c)
        case(c)
        4'b0000: HEX = 7'b1000000;
        4'b0001: HEX = 7'b1111001;
        4'b0010: HEX = 7'b0100100;
        4'b0011: HEX = 7'b0110000;
        4'b0100: HEX = 7'b0011001;
        4'b0101: HEX = 7'b0010010;
```

```
        4'b0110: HEX = 7'b0000010;
        4'b0111: HEX = 7'b1111000;
        4'b1000: HEX = 7'b0000000;
        4'b1001: HEX = 7'b0010000;
        4'b1010: HEX = 7'b0001000; //A
        4'b1011: HEX = 7'b0000011; //B
        4'b1100: HEX = 7'b1000110; //C
        4'b1101: HEX = 7'b0100001; //D
        4'b1110: HEX = 7'b0000110; //E
        4'b1111: HEX = 7'b0001110; //F
        default: HEX = 7'b1111111;
    endcase
endmodule
```

Bài 8:

```
module cau8(SW, KEY, HEX0, HEX1);
    input [1:0]SW;
    input [1:0]KEY;
    output [6:0]HEX0;
    output [6:0]HEX1;
    wire [7:0]bcd;

    counter(SW[0], SW[1], KEY[1], KEY[0], bcd);
    Decoder_HEX(bcd[3:0], HEX0);
    Decoder_HEX(bcd[7:4], HEX1);
endmodule

module counter(en, s, rst, clk, bcd);
    parameter n = 8;
    input s, en, rst, clk;
    output reg [n-1:0]bcd;

    always@(posedge clk or negedge rst)
        if(!rst)
```

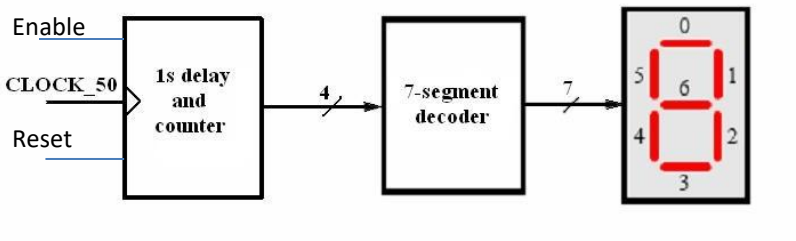
```
            bcd <= 0;
        else if(en)
            bcd <= bcd + (s?1:-1);
endmodule

module Decoder_HEX(c,HEX);
    input[3:0]c;
    output [6:0]HEX;
    reg[6:0]HEX;
    always@(c)
        case(c)
        4'b0000: HEX = 7'b1000000;
        4'b0001: HEX = 7'b1111001;
        4'b0010: HEX = 7'b0100100;
        4'b0011: HEX = 7'b0110000;
        4'b0100: HEX = 7'b0011001;
        4'b0101: HEX = 7'b0010010;
        4'b0110: HEX = 7'b0000010;
        4'b0111: HEX = 7'b1111000;
        4'b1000: HEX = 7'b0000000;
        4'b1001: HEX = 7'b0010000;
        4'b1010: HEX = 7'b0001000; //A
        4'b1011: HEX = 7'b0000011; //B
        4'b1100: HEX = 7'b1000110; //C
        4'b1101: HEX = 7'b0100001; //D
        4'b1110: HEX = 7'b0000110; //E
        4'b1111: HEX = 7'b0001110; //F
        default: HEX = 7'b1111111;
    endcase
endmodule
```

Bài tập tuần 6

Yêu cầu của bài thực hành

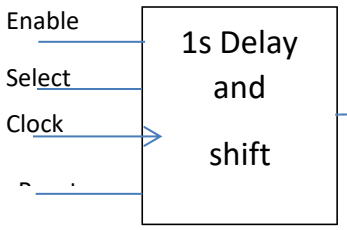
Câu 1. Thược hiện lại các câu 2, 4, 5, 6 của Bài 5 sử dụng bộ dao động 50MHZ (CLOCK_50) tạo trì hoãn 1 giây thay cho KEY[0].

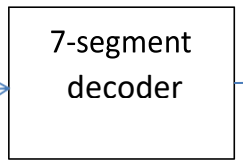
Câu 2. Viết code Verilog mô tả mạch đếm lên số thập phân từ 0 – 9 hiển thị trên led 7 đoạn HEX0 khi có tín hiệu Enable được đặt lên 1. Mỗi 1 giây tăng lên 1 đơn vị (sử dụng bộ dao động 50MHZ tạo trì hoãn 1 giây). Sử dụng KEY[0] để làm reset và SW[0] làm Enable.



Câu 3. Viết code verilog mô tả mạch dịch chuyển ký tự FPGA trên 4 led 7 đoạn từ HEX0 – HEX3. Dịch chuyển ký tự từ phải sang trái với thời gian trì hoãn là 1 giây (sử dụng bộ dao động 50MHZ).

Câu 4. Viết code verilog mô tả mạch dịch chuyển ký tự FPGA trên 4 led 7 đoạn từ HEX0 – HEX3. Khi nhấn Reset thì tắt các led. Ngược lại, khi tín hiệu Enable được đặt lên 1, nếu Select = 1 thì dịch ký tự từ phải sang trái, ngược lại Select = 0 dịch ký tự từ trái sang phải. Với thời gian trì hoãn là 1 giây (sử dụng bộ dao động 50MHZ).





Ký tự hiển thị			
HEX3	**HEX2**	**HEX1**	**HEX0**
F	P	G	A

Sử dụng các tin hiệu trên board DE1:

- Enable = SW[0]
- Select = SW[1]
- Clock = CLOCK_50
- Reset = KEY[0]

Câu 5. Viết code verilog mô tả mạch đếm lên số thập phân 2 chữ số từ 00 --> 99 (sử dụng biểu thức toán học) hiển thị lên 2 led 7 đoạn: HEX0, HEX1. Khi Enable (SW[0]) được đặt lên 1 thì cứ mỗi giây tăng giá trị đếm lên 1 giá trị (sử dụng bộ dao động 50MHZ để tạo trì hoãn 1 giây). Khi nhấn Reset (KEY[0]) thì đưa bộ đếm về 00.

Câu 6: Viết code verilog mô tả mạch đếm xuống số thập phân 2 chữ số từ 99 --> 00 (sử dụng biểu thức toán học) hiển thị lên 2 led 7 đoạn: HEX0, HEX1. Khi Enable (SW[0]) được về mức 0 thì cứ mỗi giây giảm giá trị đếm đi 1 giá trị (sử dụng bộ dao động 50MHZ để tạo trì hoãn 1 giây). Khi nhấn Reset (KEY[0]) thì đưa bộ đếm về 00.

Câu 7. Kết hợp câu 5 và 6, viết code verilog mô tả mạch đếm lên và đếm xuống số thập phân từ 00 <--> 99.

- Nếu SW[0]=1 chọn mạch đếm lên, ngược lại SW[0]=0 mạch đếm xuống.
- Mỗi 1 giây tăng/giảm giá trị đếm lên/xuống 1 giá trị (sử dụng bộ dao động 50MHZ để tạo trì hoãn 1 giây).
- Hiển thị trên 2 led 7 đoạn HEX0 và HEX1.

Câu 8.Viết code verilog mô tả mạch đếm lên số thập phân 3 chữ số từ 000 --> 999 (sử dụng biểu thức toán học) hiển thị lên 3 led 7 đoạn: HEX0, HEX1, HEX2. Mỗi giây tăng giá trị đếm lên 1 giá trị (sử dụng bộ dao động 50MHZ để tạo trì hoãn 1 giây). Khi nhấn Reset (KEY[0]) thì đưa bộ đếm về 000.

CODE

Bài 1:

```
module cau2(SW, KEY, HEX0);
    input [0:0]SW;   // T
    input [1:0]KEY;  // clk = key0, rst = key1
    output [6:0]HEX0;
    wire [3:0]q, t;

    FF_T(SW[0], KEY[1], KEY[0], q[0]);
    and(t[1], SW[0], q[0]);
    FF_T(t[1], KEY[1], KEY[0], q[1]);
    and(t[2], t[1], q[1]);
    FF_T(t[2], KEY[1], KEY[0], q[2]);
    and(t[3], t[2], q[2]);
    FF_T(t[3], KEY[1], KEY[0], q[3]);
    Decoder_HEX(q, HEX0);
```

```
endmodule

module FF_T(t, rst, clk, q);
    input t, rst, clk;
    output reg q;

    always@(posedge clk or negedge rst)
        if(!rst)
            q <= 1'b0;
        else
            if(t)
                q <= ~q;
endmodule

module Decoder_HEX(c,HEX);
    input[3:0]c;
    output [6:0]HEX;
    reg[6:0]HEX;
    always@(c)
        case(c)
        4'b0000: HEX = 7'b1000000;
        4'b0001: HEX = 7'b1111001;
        4'b0010: HEX = 7'b0100100;
        4'b0011: HEX = 7'b0110000;
        4'b0100: HEX = 7'b0011001;
        4'b0101: HEX = 7'b0010010;
        4'b0110: HEX = 7'b0000010;
        4'b0111: HEX = 7'b1111000;
        4'b1000: HEX = 7'b0000000;
        4'b1001: HEX = 7'b0010000;
        4'b1010: HEX = 7'b0001000; //A
        4'b1011: HEX = 7'b0000011; //B
        4'b1100: HEX = 7'b1000110; //C
```

```
        4'b1101: HEX = 7'b0100001; //D
        4'b1110: HEX = 7'b0000110; //E
        4'b1111: HEX = 7'b0001110; //F
        default: HEX = 7'b1111111;
    endcase
endmodule
```

Bài 2:

```
module cau2(clock_50,KEY,HEX0,SW);
    input clock_50;
    input [0:0]KEY,SW;
    output [6:0]HEX0;
    wire [3:0]a;
    counter(SW[0],clock_50,KEY[0],a);
    seg70(a,HEX0);
endmodule
module counter(enable,clock,reset,count);
    input reset,clock,enable;
    output reg[3:0]count;
    reg [25:0]delay;
    always @(posedge clock)
    begin
    if (delay==49999999)
        delay<=0;
        else
        delay<=delay+1;
    end
    always@(posedge clock , negedge reset)
    if(reset==0)
    count<=0;
    else
    if (enable)
        if (delay==0)
```

```
            if(count==9)
                count<=0;
    else
        count<=count+1;
endmodule

module seg70 (bcd, HEX);
    input [3:0] bcd;
    output reg [6:0]HEX;
    always @(bcd)
    case (bcd)       //abcdefg
            4'b0000: HEX = 7'b1000000;
            4'b0001: HEX = 7'b1111001;
            4'b0010: HEX = 7'b0100100;
            4'b0011: HEX = 7'b0110000;
            4'b0100: HEX = 7'b0011001;
            4'b0101: HEX = 7'b0010010;
            4'b0110: HEX = 7'b0000010;
            4'b0111: HEX = 7'b1111000;
            4'b1000: HEX = 7'b0000000;
            4'b1001: HEX = 7'b0010000;
            4'b1010: HEX = 7'b0001000; //A
            4'b1011: HEX = 7'b0000011; //B
            4'b1100: HEX = 7'b1000110; //C
            4'b1101: HEX = 7'b0100001; //D
            4'b1110: HEX = 7'b0000110; //E
            4'b1111: HEX = 7'b0001110; //F
            default: HEX = 7'b1111111;
    endcase
endmodule
```

Bài 3:

```
module cau3(SW,KEY,CLOCK_50,HEX0,HEX1,HEX2,HEX3);
```

```
    input [1:0]SW;
    input [0:0]KEY;
    input CLOCK_50;
    output [6:0]HEX0,HEX1,HEX2,HEX3;
    wire [2:0]dem;
    //assign dem[0] = 0;

    count_and_delay (KEY[0], SW[0], CLOCK_50, dem);
    decoder_FPGA (dem, HEX0);
    decoder_FPGA (dem - 1, HEX1);
    decoder_FPGA (dem - 2, HEX2);
    decoder_FPGA (dem - 3, HEX3);
endmodule

module count_and_delay (reset, enable, clock, dem);
    input reset,enable,clock;
    output reg[2:0]dem;
    reg [25:0]delay;
    always @(posedge  clock)
    begin
    if (delay==49999999)
        delay <= 0;
    else
        delay <= delay +1;
    end
    always@(posedge clock)
    if(!reset)
        dem = 0;
    else
        if(enable)
            begin
                if(delay == 0)
                    dem = dem + 1;
```

```
            end
endmodule

module decoder_FPGA(c,hex);
    input [2:0]c;
    output reg[6:0]hex;
    always@(c)
    case (c)
    3'b 000: hex = 7'b 1111111; //led off
    3'b 001: hex = 7'b 0001110; //F
    3'b 010: hex = 7'b 0001100; //P
    3'b 011: hex = 7'b 0000010; //G
    3'b 100: hex = 7'b 0001000; //A
    default : hex= 7'b1111111;//led off;
    endcase
endmodule
```

Bài 4:

```
module cau4(SW,KEY,CLOCK_50,HEX0,HEX1,HEX2,HEX3);
    input [1:0]SW;
    input [0:0]KEY;
    input CLOCK_50;
    output [6:0]HEX0,HEX1,HEX2,HEX3;
    wire [2:0]dem;

    count_and_delay (KEY[0],SW[0],SW[1],CLOCK_50,dem);
    decoder_FPGA (dem,HEX0);
    decoder_FPGA (dem-1,HEX1);
    decoder_FPGA (dem-2,HEX2);
    decoder_FPGA (dem-3,HEX3);
endmodule

module count_and_delay (reset,enable,select,clock,dem);
```

```
    input reset,enable,clock,select;
    output reg[2:0]dem;
    reg [25:0]delay;
    always @(posedge  clock)
    begin
    if (delay==49999999)
        delay <= 0;
    else
        delay <= delay +1;
    end
    always@(posedge clock)
    if(!reset)
        dem = 0;
    else
        if(enable)

            begin
                if(delay == 0)
                    dem = dem + (select?1:-1);
            end
endmodule

module decoder_FPGA(c,hex);
    input [2:0]c;
    output reg[6:0]hex;
    always@(c)
    case (c)
    3'b 000: hex = 7'b 1111111; //led off
    3'b 001: hex = 7'b 0001110; //F
    3'b 010: hex = 7'b 0001100; //P
    3'b 011: hex = 7'b 0000010; //G
    3'b 100: hex = 7'b 0001000; //A
    default : hex= 7'b1111111;//led off;
```

```
    endcase
endmodule
```

Bài 5:

```
module cau5(SW, HEX0, HEX1, CLOCK_50, KEY);
    input [0:0]SW;
    input [0:0]KEY;
    input CLOCK_50;
    output [6:0]HEX0, HEX1;
    wire [3:0]a, b;
    counter(CLOCK_50, KEY[0], a, b, SW[0]);
    display_0_9(a, HEX0);
    display_0_9(b, HEX1);
endmodule

module counter(clock,reset,BCD0, BCD1, enable);
    input reset,clock, enable;
    output reg[3:0]BCD0, BCD1;
    reg [25:0]count;
    always @(posedge clock)
        begin
            if(count==49999999)
                count<=0;
            else
            count=count+1;
        end
    always@(posedge clock)
        if(!reset)
            begin
                BCD0<=0;
                BCD1<=0;
            end
```

```
        else
            if(enable)
            begin
                if(count==0)
                    begin
                        if(BCD0==9)
                            begin
                                if(BCD1==9)
                                    begin
                                        BCD1<=0;
                                        BCD0<=0;
                                    end
                                else
                                    begin
                                        BCD1<=BCD1+1;
                                        BCD0<=0;
                                    end
                            end
                        else
                            BCD0<=BCD0+1;
                    end
            end

endmodule

module display_0_9(c,hex);
    input [3:0]c;
    output reg [6:0]hex;
    always@(c)
        case(c)
            4'b0000 :hex =7'b1000000;
            4'b0001 :hex =7'b1111001;
            4'b0010 :hex =7'b0100100;
```

```
            4'b0011 :hex =7'b0110000;
            4'b0100 :hex =7'b0011001;
            4'b0101 :hex =7'b0010010;
            4'b0110 :hex =7'b0000010;
            4'b0111 :hex =7'b1111000;
            4'b1000 :hex =7'b0000000;
            4'b1001 :hex =7'b0010000;
            //4'b 1010: hex = 7'b 0100000;
            //4'b 1011: hex = 7'b 0000011;
            //4'b 1100: hex = 7'b 1000110;
            //4'b 1101: hex = 7'b 0100001;
            //4'b 1110: hex = 7'b 0000110;
            //4'b 1111: hex = 7'b 0001110;
            default :hex =7'b1111111;
        endcase
endmodule
```

Bài 6:

```
module cau6(CLOCK_50,KEY,SW,HEX0, HEX1);
    input CLOCK_50;
    input [0:0]KEY;
    input [0:0]SW;
    output [6:0]HEX0, HEX1;
    wire [3:0]a,b;
        counter (CLOCK_50, KEY[0],SW[0], a,b);
        display_0_9 (a, HEX0);
        display_0_9 (b, HEX1);
endmodule

module counter(CLOCK_50,reset,enable,BCD0, BCD1);
    input reset,CLOCK_50,enable;
    output reg[3:0]BCD0, BCD1;
```

```
    reg [25:0]delay;
    //tao delay 1s
    always @(posedge CLOCK_50)
        begin
            if(delay==49999999)
                delay<=0;
            else
            delay=delay+1;
        end
    always@(posedge CLOCK_50 or negedge reset)
        if(!reset)
            begin
                BCD0<=0;
                BCD1<=0;
            end
        else
        if (enable==0)
        begin
            if(delay==0)
                    begin
                        if(BCD0==0)
                            begin
                                BCD0<=9;
                                    if(BCD1==0)
                                        BCD1<=9;
                                    else
                                        BCD1<=BCD1-1;
                            end
                        else
                            BCD0<=BCD0-1;
                    end
                    end
endmodule
```

```
module display_0_9(c,hex);
    input [3:0]c;
    output reg [6:0]hex;
    always@(c)
        case(c)
            4'b0000 :hex =7'b1000000;
            4'b0001 :hex =7'b1111001;
            4'b0010 :hex =7'b0100100;
            4'b0011 :hex =7'b0110000;
            4'b0100 :hex =7'b0011001;
            4'b0101 :hex =7'b0010010;
            4'b0110 :hex =7'b0000010;
            4'b0111 :hex =7'b1111000;
            4'b1000 :hex =7'b0000000;
            4'b1001 :hex =7'b0010000;
            //4'b 1010: hex = 7'b 0100000;
            //4'b 1011: hex = 7'b 0000011;
            //4'b 1100: hex = 7'b 1000110;
            //4'b 1101: hex = 7'b 0100001;
            //4'b 1110: hex = 7'b 0000110;
            //4'b 1111: hex = 7'b 0001110;
            default :hex =7'b1111111;
        endcase
endmodule
```

Bài 7:

```
module cau7(CLOCK_50,KEY,SW,HEX0, HEX1);
    input CLOCK_50;
    input [0:0]KEY;
    input [1:0]SW;
```

```
    output [6:0]HEX0, HEX1;
    wire [3:0]a,b;
        counter (CLOCK_50, KEY[0],SW[0],SW[1], a,b);
        display_0_9 (a, HEX0);
        display_0_9 (b, HEX1);
endmodule

module counter(CLOCK_50,reset,enable,select,BCD0, BCD1);
    input reset,CLOCK_50,enable,select;
    output reg[3:0]BCD0, BCD1;
    reg [25:0]delay;
    //tao delay 1s
    always @(posedge CLOCK_50)
        begin
            if(delay==49999999)
                delay<=0;
            else
            delay=delay+1;
        end
    always@(posedge CLOCK_50 or negedge reset)
        if(!reset)
            begin
                BCD0<=9;
                BCD1<=9;
            end
        else
        if (enable)
        begin
            if(select==0)
            begin
            if(delay==0)
                    begin
                        if(BCD0==0)
```

```
                    begin
                        BCD0<=9;
                            if(BCD1==0)
                                BCD1<=9;
                            else
                                BCD1<=BCD1-1;
                    end
                else
                    BCD0<=BCD0-1;
            end
            end

            else
                begin
        if(delay==0)
            begin
                if(BCD0==9)
                    begin
                        BCD0<=0;
                            if(BCD1==9)
                                BCD1<=0;
                            else
                                BCD1<=BCD1+1;

                        end
                else
                    BCD0<=BCD0+1;

            end
            end
            end
endmodule
```

```
module display_0_9(c,hex);
    input [3:0]c;
    output reg [6:0]hex;
    always@(c)
        case(c)
            4'b0000 :hex =7'b1000000;
            4'b0001 :hex =7'b1111001;
            4'b0010 :hex =7'b0100100;
            4'b0011 :hex =7'b0110000;
            4'b0100 :hex =7'b0011001;
            4'b0101 :hex =7'b0010010;
            4'b0110 :hex =7'b0000010;
            4'b0111 :hex =7'b1111000;
            4'b1000 :hex =7'b0000000;
            4'b1001 :hex =7'b0010000;
            default :hex =7'b1111111;
        endcase
endmodule
```

Bài 8:

```
module cau8(SW, HEX0, HEX1, HEX2, CLOCK_50, KEY);
    input [0:0]SW;
    input [0:0]KEY;
    input CLOCK_50;
    output [6:0]HEX0, HEX1, HEX2;
    wire [3:0]a, b,c;
    counter(CLOCK_50, KEY[0], a, b, c, SW[0]);
    display_0_9(a, HEX0);
    display_0_9(b, HEX1);
    display_0_9(c, HEX2);
endmodule
```

```
module counter(clock,reset,BCD0, BCD1, BCD2, enable);
    input reset,clock, enable;
    output reg[3:0]BCD0, BCD1,BCD2;
    reg [25:0]count;
    always @(posedge clock)
            begin
                if(count==1000000)
                    count<=0;
                else
                    count=count+1;
            end
    always@(posedge clock)
        if(!reset)
            begin
                BCD0<=0;
                BCD1<=0;
                BCD2<=0;
            end
        else
            if(enable)//mach dem len
            begin
                if(count==0)
                    begin
                        if(BCD0==9)
                            begin
                                if(BCD1==9)
                                    begin
                                        if (BCD2==9)
                                            begin
                                                BCD2<=0;
                                                BCD1<=0;
                                                BCD0<=0;
                                            end
```

```
                                    else
                                        begin
                                            BCD2<=BCD2+1;
                                            BCD1<=0;
                                            BCD0<=0;
                                        end
                                end
                            else
                                begin
                                    BCD1<=BCD1+1;
                                    BCD0<=0;
                                end
                        end
                    else
                        BCD0<=BCD0+1;
                end
            end
endmodule

module display_0_9(c,hex);
    input [3:0]c;
    output reg [6:0]hex;
    always@(c)
        case(c)
            4'b0000 :hex =7'b1000000;
            4'b0001 :hex =7'b1111001;
            4'b0010 :hex =7'b0100100;
            4'b0011 :hex =7'b0110000;
            4'b0100 :hex =7'b0011001;
            4'b0101 :hex =7'b0010010;
            4'b0110 :hex =7'b0000010;
            4'b0111 :hex =7'b1111000;
            4'b1000 :hex =7'b0000000;
```

```
            4'b1001 :hex =7'b0010000;
            default :hex =7'b1111111;
        endcase
endmodule
```

Bài tập tuần 7

Yêu cầu của bài thực hành

Câu 1. Viết code verilog mô tả mạch đếm lên số thập phân chẵn/lẻ gồm 2 chữ số.

- Nếu SW[0]=0 mạch đếm lên chẵn, ngược lại SW[0]=1 mạch đếm lên lẻ.
- Mỗi 1 giây tăng lên 1 giá trị (sử dụng bộ dao động 50MHZ để tạo trì hoãn 1 giây).
- Hiển thị trên 2 led 7 đoạn HEX0, HEX1.

Câu 2. Viết code verilog mô tả mạch đếm xuống số thập phân chẵn/lẻ gồm 2 chữ số.

- Nếu SW[0]=0 mạch đếm xuống chẵn, ngược lại SW[0]=1 mạch đếm xuống lẻ.
- Mỗi 1 giây giảm đi 1 giá trị (sử dụng bộ dao động 50MHZ để tạo trì hoãn 1 giây).
- Hiển thị trên 2 led 7 đoạn HEX0, HEX1.

Câu 3. Thiết kế mạch đồng hồ hiển thị giờ, phút,

giây như sau: Giây : hiển thị trên HEX0 và

HEX1

Phút : hiển thị trên HEX2

và HEX3. Giờ : hiển thị

trên LEDR[7:0].

Câu 4. Thiết kế mạch đèn báo động.

SW[7:0] sử dụng để định thời gian (giây). Khi đồng hồ đếm xuống (sử dụng HEX0, HEX1 để hiển thời gian) đến thời gian đã định trước thì mạch kích mở toàn bộ 10 LEDR sáng lên.

Câu 5. Thiết kế mạch đèn giao thông. Đèn xanh sáng trong vòng 30 giây sau đó chuyển sang đèn đỏ, cũng sáng trong vòng 30 giây (sử dụng HEX0 và HEX1 để hiển thị thời gian và LEDR0, LEDG7 là tín hiệu đèn đỏ và đèn xanh)

CODE

Bài 1:

```
module cau1(SW, HEX0, HEX1, CLOCK_50, KEY);
    input [1:0]SW;
    input [0:0]KEY;
    input CLOCK_50;
    output [6:0]HEX0, HEX1;
    wire [3:0]a, b;
    counter(CLOCK_50, KEY[0], a, b, SW[1], SW[0]);
    display_0_9(a, HEX0);
    display_0_9(b, HEX1);
```

```
endmodule

module counter(clock, reset, BCD0, BCD1, enable, even_odd);
    input reset,clock, enable, even_odd;
    output reg[3:0]BCD0, BCD1;
    reg [25:0]count;
    always @(posedge clock)
        begin
            if(count==49999999)
                count<=0;
            else
            count=count+1;
        end
    always@(posedge clock)
        if(!reset)
            begin
                BCD0<=0;
                BCD1<=0;
            end
        else
            if(enable)
            begin
                if(even_odd == 0) // EVEN
                    begin
                        if (count == 0)
                        begin
                            if(BCD0 % 2 == 0)
                                if(BCD0 == 8)
                                    begin
                                        BCD0 <= 0;
                                        if(BCD1 == 9)
                                            BCD1 <= 0;
                                        else
```

```
                                        BCD1 <= BCD1 + 1;
                                    end
                                else
                                    BCD0 <= BCD0 + 2;
                            else
                                BCD0 <= BCD0 + 1;
                        end
                    end
                else // ODD
                    begin
                        if (count == 0)
                        begin
                            if(BCD0 % 2 == 1)
                                if(BCD0 == 9)
                                    begin
                                        BCD0 <= 1;
                                        if(BCD1 == 9)
                                            BCD1 <= 0;
                                        else
                                            BCD1 <= BCD1 + 1;
                                    end
                                else
                                    BCD0 <= BCD0 + 2;
                            else
                                BCD0 <= BCD0 + 1;
                        end
                    end

            end

endmodule

module display_0_9(c,hex);
```

```
    input [3:0]c;
    output reg [6:0]hex;
    always@(c)
        case(c)
            4'b0000 :hex =7'b1000000;
            4'b0001 :hex =7'b1111001;
            4'b0010 :hex =7'b0100100;
            4'b0011 :hex =7'b0110000;
            4'b0100 :hex =7'b0011001;
            4'b0101 :hex =7'b0010010;
            4'b0110 :hex =7'b0000010;
            4'b0111 :hex =7'b1111000;
            4'b1000 :hex =7'b0000000;
            4'b1001 :hex =7'b0010000;
            default :hex =7'b1111111;
        endcase
endmodule
```

Bài 2:

```
module cau2(SW, HEX0, HEX1, CLOCK_50, KEY);
    input [1:0]SW;
    input [0:0]KEY;
    input CLOCK_50;
    output [6:0]HEX0, HEX1;
    wire [3:0]a, b;
    counter(CLOCK_50, KEY[0], a, b, SW[1], SW[0]);
    display_0_9(a, HEX0);
    display_0_9(b, HEX1);
endmodule

module counter(clock, reset, BCD0, BCD1, enable, even_odd);
    input reset,clock, enable, even_odd;
```

```
output reg[3:0]BCD0, BCD1;
reg [25:0]count;
always @(posedge clock)
    begin
        if(count==49999999)
            count<=0;
        else
        count=count+1;
    end
always@(posedge clock)
    if(!reset)
        begin
            BCD0 <= 0;
            BCD1 <= 0;
        end
    else
        if(enable)
        begin
            if(even_odd == 0) // EVEN
                begin
                    if (count == 0)
                    begin
                        if(BCD0 % 2 == 0)
                            if(BCD0 == 0)
                                begin
                                    BCD0 <= 8;
                                    if(BCD1 == 0)
                                        BCD1 <= 9;
                                    else
                                        BCD1 <= BCD1 - 1;
                                end
                            else
                                BCD0 <= BCD0 - 2;
```

```
                            else
                                BCD0 <= BCD0 - 1;
                        end
                    end
                else // ODD
                    begin
                        if (count == 0)
                        begin
                            if(BCD0 % 2 == 1)
                                if(BCD0 == 1)
                                    begin
                                        BCD0 <= 9;
                                        if(BCD1 == 0)
                                            BCD1 <= 9;
                                        else
                                            BCD1 <= BCD1 - 1;
                                    end
                                else
                                    BCD0 <= BCD0 - 2;
                            else
                                BCD0 <= BCD0 - 1;
                        end
                    end

            end

endmodule

module display_0_9(c,hex);
    input [3:0]c;
    output reg [6:0]hex;
    always@(c)
        case(c)
```

```
            4'b0000 :hex =7'b1000000;
            4'b0001 :hex =7'b1111001;
            4'b0010 :hex =7'b0100100;
            4'b0011 :hex =7'b0110000;
            4'b0100 :hex =7'b0011001;
            4'b0101 :hex =7'b0010010;
            4'b0110 :hex =7'b0000010;
            4'b0111 :hex =7'b1111000;
            4'b1000 :hex =7'b0000000;
            4'b1001 :hex =7'b0010000;
            default :hex =7'b1111111;
        endcase
endmodule
```

Bài 3:

```
module cau3(LEDR,KEY,CLOCK_50,HEX0,HEX1,HEX2,HEX3);
    input [0:0]CLOCK_50,KEY;
    output [7:0]LEDR;
    output [6:0]HEX0,HEX1,HEX2,HEX3;
    wire [3:0]bcd0, bcd1, bcd2, bcd3, led0, led1;

    Counter(CLOCK_50,KEY[0],bcd0,bcd1,bcd2,bcd3,led0);
    decoder a (bcd0,HEX0);
    decoder b (bcd1,HEX1);
    decoder c (bcd2,HEX2);
    decoder d (bcd3,HEX3);
    assign LEDR[7:0] = led0;
endmodule

module Counter(clock,reset,bcd0,bcd1,bcd2,bcd3,led0);
    input clock,reset;
    output reg [3:0]bcd0,bcd1,bcd2,bcd3;
```

```
output reg [7:0]led0;
reg [25:0]count;

always @(posedge clock)
        begin
            if (count==99999)
                count <=0;
            else
                count <= count + 1;
        end
always@(posedge clock)
        begin
                if (!reset)
                begin
                    bcd0<=0;
                    bcd1<=0;
                    bcd2<=0;
                    bcd3<=0;
                    led0<=0;
                end
                else
                    if (count == 0)
                    begin
                        if (bcd0 == 9)
                        begin
                            bcd0 <= 0;
                            if (bcd1 == 5)
                                begin
                                bcd1 <= 0;
                                    if (bcd2 == 9)
                                    begin
                                    bcd2 <= 0;
                                        if (bcd3 == 5)
```

```
                                                    begin
                                                    bcd3 <= 0;
                                                    if (led0 == 23)
                                                        begin
                                                            led0 <= 0;
                                                            bcd0 <= 0;
                                                            bcd1 <= 0;
                                                            bcd2 <= 0;
                                                            bcd3 <= 0;
                                                        end
                                                    else
                                                        begin
                                                            if (led0 ==
23)
                                                                led0 <=
0;
                                                            else led0 <=
led0 + 1;
                                                        end
                                                    end
                                                else bcd3 <= bcd3 + 1;
                                                end
                                                else bcd2 <= bcd2 + 1;
                                                    end
                                                else bcd1 <= bcd1 + 1;
                                            end
                                        else bcd0 <= bcd0 + 1;
                                    end
            end
endmodule

module decoder(c,hex);
    input [3:0]c;
    output reg [6:0]hex;
```

```
    always@(c)
    case (c)
        4'b 0000: hex = 7'b 1000000;
        4'b 0001: hex = 7'b 1111001;
        4'b 0010: hex = 7'b 0100100;
        4'b 0011: hex = 7'b 0110000;
        4'b 0100: hex = 7'b 0011001;
        4'b 0101: hex = 7'b 0010010;
        4'b 0110: hex = 7'b 0000010;
        4'b 0111: hex = 7'b 1111000;
        4'b 1000: hex = 7'b 0000000;
        4'b 1001: hex = 7'b 0010000;
    endcase
endmodule
```

Bài 4:

```
module cau4(CLOCK_50, KEY, SW, HEX0, HEX1, LEDR);
    input CLOCK_50;
    input [0:0]KEY;
    input [8:0]SW;
    output [6:0]HEX0, HEX1;
    output [9:0]LEDR;
    wire [3:0]a,b;

    counter (CLOCK_50, KEY[0],SW[8], a, b, SW[3:0], SW[7:4], LEDR);
    display_0_9 (a, HEX0);
    display_0_9 (b, HEX1);
endmodule

module counter(CLOCK_50, reset, enable, BCD0, BCD1, var0, var1,
led);
    input reset,CLOCK_50,enable;
```

```
input [3:0]var0, var1;
output reg[3:0]BCD0, BCD1;
output reg[9:0]led;
reg [25:0]delay;
//tao delay 1s
always @(posedge CLOCK_50)
    begin
        if(delay==49999999)
            delay<=0;
        else
        delay=delay+1;
    end

always@(posedge CLOCK_50 or negedge reset)
    if(!reset)
        begin
            BCD0 <= 0;
            BCD1 <= 0;
            led = 10'b0000000000;
        end
    else
    if (enable == 1)
    begin
        if(delay == 0)
            begin
                if(BCD0 == 0)
                    begin
                        BCD0 <= 9;
                        if(BCD1 == 0)
                            begin
                            BCD1 <= 9;
                            end
                        else
```

```
                                BCD1 <= BCD1 - 1;
                            end
                        else
                            BCD0 <= BCD0 - 1;
                            if ((BCD0 == var0) && (BCD1 == var1))
                            begin
                                led = 10'b1111111111;
                            end
                    end
            end
endmodule

module display_0_9(c,hex);
    input [3:0]c;
    output reg [6:0]hex;
    always@(c)
        case(c)
            4'b0000 :hex =7'b1000000;
            4'b0001 :hex =7'b1111001;
            4'b0010 :hex =7'b0100100;
            4'b0011 :hex =7'b0110000;
            4'b0100 :hex =7'b0011001;
            4'b0101 :hex =7'b0010010;
            4'b0110 :hex =7'b0000010;
            4'b0111 :hex =7'b1111000;
            4'b1000 :hex =7'b0000000;
            4'b1001 :hex =7'b0010000;
            default :hex =7'b1111111;
        endcase
endmodule
```

Bài 5:

```
module cau5(CLOCK_50,KEY,HEX0,HEX1,LEDR,LEDG);
    input CLOCK_50;
    input [0:0]KEY;
    output [0:6]HEX0,HEX1;
    output [0:0]LEDR,LEDG;
    wire [3:0]giay0,giay1;
    count (CLOCK_50,KEY[0],giay0,giay1,LEDG[0],LEDR[0]);
    decoder_BCD dv (giay0,HEX0);
    decoder_BCD ch (giay1,HEX1);
endmodule

module count (CLK,CLR,giay0,giay1,green,red);
    input CLK,CLR;
    output reg [3:0]giay0,giay1;
    output reg green,red;
    reg [26:0]count;
    always @ (posedge CLK)
        if (count==10000000) count <= 0;
        else count <= count+1;

    always @ (posedge CLK)
    begin
        if (!CLR)
            begin giay0 <=0;
                  giay1 <=0;
                  green <=1;
                  red <=0;
            end
        else if (count==0)
                    if (giay0==0)
                        begin
```

```
                                giay0 <=9;
                                if (giay1==0)
                                    begin
                                        giay1 <=2;
                                        green <=~green;
                                        red <=~red;
                                    end
                                else
                                    giay1 <=giay1-1;
                            end
                        else giay0 <=giay0-1;
    end
endmodule

module decoder_BCD (i,led);
    input [3:0]i;
    output reg [6:0]led;
    always @(i)
    case(i)
    0:led=7'b0000001;
    1:led=7'b1001111;
    2:led=7'b0010010;
    3:led=7'b0000110;
    4:led=7'b1001100;
    5:led=7'b0100100;
    6:led=7'b0100000;
    7:led=7'b0001111;
    8:led=7'b0000000;
    9:led=7'b0000100;
    10:led=7'b0001000;
    11:led=7'b1100000;
    12:led=7'b0110001;
    13:led=7'b1000010;
```

```
    14:led=7'b0110000;
    15:led=7'b0111000;
    endcase
endmodule
```

Bài tập tuần 8

Yêu cầu của bài thực hành

Câu 1: Viết code verilog mô tả ALU đơn giản, với ALU 4-bit Inputs: A, B thực hiện các phép toán logic như sau:

Opcode	ALU Operation
000	ALU_Out = NOT A
001	ALU_Out = A AND B
010	ALU_Out = A OR B
100	ALU_Out = A XOR B
default	ALU_Out = 4'b0000

a) Xem kết quả tổng hợp mạch (RTL Viewer)
b) Kiểm tra hoạt động của ALU trên board DE1 với ALU 4-bit Inputs:
A = 0110; B = 0101;
Sử dụng tín hiệu trên board:
SW[2:0] = Opcode
LEDR[3:0] = ALU_Out

Câu 2: Mở rộng câu 1: bổ sung thêm kết quả ALU_Out được hiển thị lên led 7 đoạn HEX0.
Sử dụng tín hiệu trên board:
SW[2:0] = Opcode
LEDR[3:0] = ALU_Out (dùng để hiển thị kết quả dưới dạng số nhị phân)
HEX0 = ALU_Out (dùng để hiển thị kết quả số thập lục phân)

Câu 3: Mở rộng câu 2 sang 8 bit với:
A= 01010110; B = 10110101;
Sử dụng tín hiệu trên board:
SW[2:0] = Opcode
LEDR[7:0] = ALU_Out
HEX0 hiển thị kết quả số thập lục phân tương ứng với 4 bit thấp của ALU_Out
HEX1 hiển thị kết quả số thập lục phân tương ứng với 4 bit cao của ALU_Out

Câu 4: Viết code Verilog mô tả khối ALU trong bộ vi xử lý 8 bit, với khả năng thực hiện 14 thao tác khác nhau như trình bày trong bảng giá trị của ALU như sau:

Bảng giá trị của ALU

Opcode								
S4	**S3**	**S2**	**S1**	**S0**	**Cin**	**Phép toán**	**Chức năng**	**Khối thực hiện**
0	0	0	0	0	0	Y <= A	Truyền A	Đơn vị số học
0	0	0	0	0	1	Y <= A + 1	Tăng A	Đơn vị số học
0	0	0	0	1	0	Y <= A + B	A cộng B	Đơn vị số học
0	0	0	0	1	1	Y <= A + B+1	A cộng B có bit cờ	Đơn vị số học
0	0	0	1	0	0	Y <= A + Bbar	A cộng với bù-1 của B	Đơn vị số học
0	0	0	1	0	1	Y <= A – Bbar +1	Trừ	Đơn vị số học
0	0	0	1	1	0	Y <= A -1	Giảm	Đơn vị số học
0	0	0	1	1	1	Y<= A	Truyền A	Đơn vị số học
0	0	1	0	0	0	Y <= A and B	AND	Đơn vị logic
0	0	1	0	1	0	Y <= A or B	OR	Đơn vị logic
0	0	1	1	0	0	Y <= A xor B	XOR	Đơn vị logic
0	0	1	1	1	0	Y <= Abar	Bù A	Đơn vị logic
0	0	0	0	0	0	Y <= A	Truyền A	Mạch dịch bit
0	1	0	0	0	0	Y <= shl A	Dịch trái A	Mạch dịch bit
1	0	0	0	0	0	Y <= shr A	Dịch phải A	Mạch dịch bit
1	1	0	0	0	0	Y <= 0	Truyền 0	Mạch dịch bit

- Với giá trị của A, B được gán trực tiếp trong code chương trình: A= 01010110; B = 10110101;
- Sử dụng tín hiệu trên board:

SW[5:0] = Opcode

LEDR[7:0] = ALU_Out

HEX0 hiển thị kết quả số thập lục phân tương ứng với 4 bit thấp của ALU_Out HEX1 hiển thị kết quả số thập lục phân tương ứng với 4 bit cao của ALU_Out

CODE

Bài 1:

```
module cau1(SW,LEDR);
    input [2:0]SW;
    output [3:0]LEDR;
    wire [3:0]A = 4'b0110;
    wire [3:0]B = 4'b0101;
    alu(A, B, SW[2:0], LEDR);
```

```
endmodule

module alu(A, B, opcode, ALU_OUT);
    input [2:0]opcode;
    input wire [3:0]A,B;
    output reg [3:0]ALU_OUT;
    always@(*)
    case (opcode)
        3'b000: ALU_OUT = ~A;
        3'b001: ALU_OUT = A & B;
        3'b010: ALU_OUT = A | B;
        3'b100: ALU_OUT = A ^ B;
        default: ALU_OUT = 4'b0000;
    endcase
endmodule
```

Bài 2:

```
module cau2(SW,LEDR,HEX0);
    input [2:0]SW;
    output [3:0]LEDR;
    output [6:0]HEX0;
    wire [3:0]A = 4'b0110;
    wire [3:0]B = 4'b0101;

    alu(A, B, SW[2:0], LEDR);
    decoder(LEDR, HEX0);
endmodule

module alu(A, B, opcode, ALU_OUT);
    input [2:0]opcode;
    input wire [3:0]A,B;
    output reg [3:0]ALU_OUT;
```

```
    always@(*)
    case (opcode)
        3'b000: ALU_OUT = ~A;
        3'b001: ALU_OUT = A & B;
        3'b010: ALU_OUT = A | B;
        3'b100: ALU_OUT = A ^ B;
        default: ALU_OUT = 4'b0000;
    endcase
endmodule

module decoder(c,hex);
    input [3:0]c;
    output reg [6:0]hex;

    always@(c)
    case (c)
        4'b 0000: hex = 7'b 1000000;
        4'b 0001: hex = 7'b 1111001;
        4'b 0010: hex = 7'b 0100100;
        4'b 0011: hex = 7'b 0110000;
        4'b 0100: hex = 7'b 0011001;
        4'b 0101: hex = 7'b 0010010;
        4'b 0110: hex = 7'b 0000010;
        4'b 0111: hex = 7'b 1111000;
        4'b 1000: hex = 7'b 0000000;
        4'b 1001: hex = 7'b 0010000;
    endcase
endmodule
```

Bài 3:

```
module cau3(SW,LEDR,HEX0,HEX1);
    input [2:0]SW;
```

```
    output [7:0]LEDR;
    output [6:0]HEX0,HEX1;
    wire [7:0]A = 8'b01010110;
    wire [7:0]B = 8'b10110101;
    alu(A, B, SW[2:0], LEDR[7:0]);
    decoder(LEDR[3:0], HEX0);
    decoder(LEDR[7:4], HEX1);
endmodule

module alu(A, B, opcode, ALU_OUT);
    input [2:0]opcode;
    input wire [7:0]A,B;
    output reg [7:0]ALU_OUT;

    always@(*)
    case (opcode)
        3'b000: ALU_OUT = ~A;
        3'b001: ALU_OUT = A & B;
        3'b010: ALU_OUT = A | B;
        3'b100: ALU_OUT = A ^ B;
        default: ALU_OUT = 4'b00000000;
    endcase
endmodule

module decoder(c,hex);
    input [3:0]c;
    output reg [6:0]hex;
    always@(c)
    case (c)
        4'b 0000: hex = 7'b 1000000;
        4'b 0001: hex = 7'b 1111001;
        4'b 0010: hex = 7'b 0100100;
        4'b 0011: hex = 7'b 0110000;
```

```
        4'b 0100: hex = 7'b 0011001;
        4'b 0101: hex = 7'b 0100100;
        4'b 0110: hex = 7'b 0100000;
        4'b 0111: hex = 7'b 1111000;
        4'b 1000: hex = 7'b 0000000;
        4'b 1001: hex = 7'b 0011000;
        4'b 1010: hex = 7'b 0001000;
        4'b 1011: hex = 7'b 0000011;
        4'b 1100: hex = 7'b 1000110;
        4'b 1101: hex = 7'b 0100001;
        4'b 1110: hex = 7'b 0000110;
        4'b 1111: hex = 7'b 0001110;
    endcase
endmodule
```

Bài 4:

```
module cau4(SW,HEX0,HEX1,LEDR);
    input [5:0]SW;
    output [7:0]LEDR;
    output [6:0]HEX0,HEX1;
    wire [7:0]x=8'b01010110;
    wire [7:0]y=8'b10110101;
    alu(SW[4:0],SW[5],x,y,LEDR);
    decoder(LEDR[3:0],HEX0);
    decoder(LEDR[7:4],HEX1);
endmodule

module alu(sel,car_in,A,B,Y);
    input [4:0]sel;
    input car_in;
    input wire [7:0]A,B;
    output reg [7:0]Y;
    reg [7:0]alu_logic, alu_arith;
```

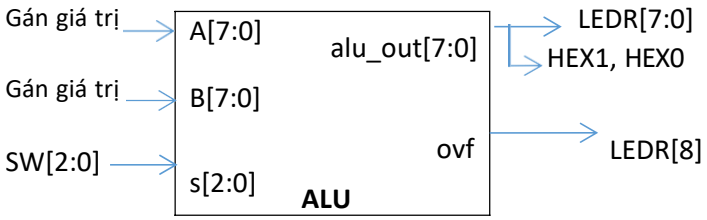
```
    always @ (sel or A or B or car_in)
    begin
    case (sel[1:0])
        2'b00:alu_logic=A&B;
        2'b01:alu_logic=A|B;
        2'b10:alu_logic=A^B;
        2'b11:alu_logic=!A;
    endcase
    case ({sel[1:0],car_in})
        3'b000:alu_arith=A;
        3'b001:alu_arith=A+1;
        3'b010:alu_arith=A+B;
        3'b011:alu_arith=A+B+1;
        3'b100:alu_arith= A+!B;
        3'b101:alu_arith=A-!B+1;
        3'b110:alu_arith=A-1;
        3'b111:alu_arith=A;
    endcase
    if(sel[2])
        Y = alu_logic;
    case(sel[4:3])
        2'b00: Y=alu_arith;
        2'b01: Y<= (A<<1);
        2'b10: Y<= (A>>1);
        2'b11: Y<=0;
    endcase
    end
endmodule

module decoder (bcd, HEX);
input [3:0] bcd;
output reg [6:0]HEX;
always @(bcd)
```

```
case (bcd)
        4'b0000: HEX = 7'b1000000;
        4'b0001: HEX = 7'b1111001;
        4'b0010: HEX = 7'b0100100;
        4'b0011: HEX = 7'b0110000;
        4'b0100: HEX = 7'b0011001;
        4'b0101: HEX = 7'b0010010;
        4'b0110: HEX = 7'b0000010;
        4'b0111: HEX = 7'b1111000;
        4'b1000: HEX = 7'b0000000;
        4'b1001: HEX = 7'b0010000;
        4'b1010: HEX = 7'b0001000;
        4'b1011: HEX = 7'b0000011;
        4'b1100: HEX = 7'b1000110;
        4'b1101: HEX = 7'b0100001;
        4'b1110: HEX = 7'b0000110;
        4'b1111: HEX = 7'b0001110;
        default: HEX = 7'b1111111;
    endcase
endmodule
```

Bài tập tuần 9

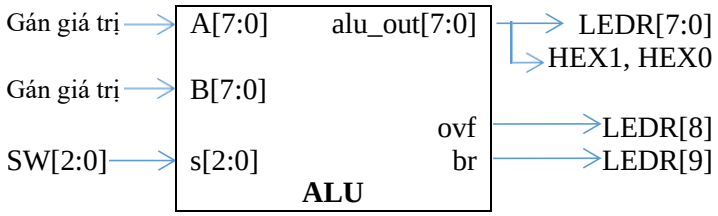
Yêu cầu của bài thực hành

Câu 1: Viết code verilog mô tả ALU với ngõ vào A, B 8bits, thực hiện các phép toán số học và logic như sau:



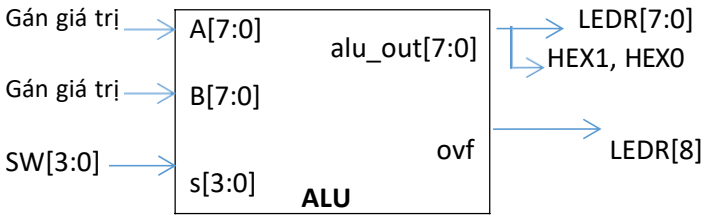
Opcode s[2:0]	ALU Operation alu_out[7:0]	Overflow ovf
000	A + B	ovf
001	A - B	0
010	A - 1	0
011	A + 1	ovf
100	A & B	0
101	A | B	0
default	~A	0

Câu 2: Viết code verilog mô tả ALU với ngõ vào A, B 8bits, thực hiện các phép toán số học và logic như sau:



Opcode s[2:0]	ALU Operation alu_out[7:0]	Overflow ovf	Branch br
000	A + B	ovf	0
001	~A	0	0
010	A ^ B	0	0
011	~(A & B)	0	0
100	A >> 1	0	0
101	B << 1	0	0
110	A == B	0	1 : đúng 0 : sai
111	A > B	0	1 : đúng 0 : sai

Câu 3: Viết code verilog mô tả ALU với ngõ vào A, B 8bits, thực hiện các phép toán số học và logic như sau:


Gán giá trị
Gán giá trị
SW[3:0]
A[7:0]
B[7:0]
s[3:0]
alu_out[7:0]
ovf
ALU
LEDR[7:0]
HEX1, HEX0
LEDR[8]

Opcode s[3:0]	ALU Operation alu_out[7:0]	Overflow ovf	Description
0000	A + B	ovf	Addition
0001	A - B	0	Subtraction
0010	A + 1	0	Increase
0011	~ B	0	Invert
0100	A << 1	0	Logical shift left
0101	A >> 1	0	Logical shift right
0110	{A[6:0],A[7]}	0	A Rotate left by 1
0111	{A[0],A[7:1]}	0	A Rotate right by 1
1000	A & B	0	Logical and
1001	A | B	0	Logical or
1010	A ^ B	0	Logical xor
1011	~(A | B)	0	Logical nor
1100	~(A & B)	0	Logical nand
1101	~(A ^ B)	0	Logical xnor
1110	(A>B)? alu_out=11111111: nếu đúng alu_out=00000000: nếu sai	0	Greater comparison
1111	(A==B)? alu_out=11111111: nếu đúng alu_out=00000000: nếu sai	0	Equal comparison

CODE

Bài 1:

```
module cau1(SW,LEDR,HEX0,HEX1);
    input [2:0]SW;
    output [8:0]LEDR;
    output[6:0]HEX0,HEX1;
    wire [7:0]alu_out;
    wire [7:0]A=8'b11111111;
    wire [7:0]B=8'b11111111;
    alu_logic(A,B,SW[2:0],alu_out,LEDR[8]);
    assign LEDR[7:0]=alu_out;

    decoder (alu_out[3:0],HEX0);
    decoder (alu_out[7:4],HEX1);
endmodule

module alu_logic(A,B,opcode,alu_out,ovf);
    input [7:0]A,B;
    input [2:0]opcode;
```

```
    output reg[7:0]alu_out;
    output reg ovf;
    always @(*)
    begin
    case (opcode)
    3'b000:
        begin
        alu_out=A+B;
        ovf=(A[7]&B[7]&~alu_out[7])|~(A[7]&~B[7]&alu_out[7])|(A[7]&B[7]&al
u_out[7]);
        end
    3'b001:
        begin
        alu_out=A-B;
        ovf=0;
        end
    3'b010:
        begin
        alu_out =A-1;
        ovf=0;
        end
    3'b011:
        begin
        alu_out=A+1;
        ovf=(A[7]&1&~alu_out[7])|~(A[7]&~1&alu_out[7])|(A[7]&1&alu_out[7])
;
        end
    3'b100:
        begin
        alu_out=A&B;
        ovf = 0;
        end
    3'b101:
        begin
        alu_out=A|B;
```

```
        ovf=0;
        end
    default:
        begin
        alu_out =~A;
        ovf=0;
        end
    endcase
    end
endmodule

module decoder(c,hex);
    input [7:0]c;
    output reg [6:0]hex;
    always@(c)
    case (c)
    4'b 0000: hex = 7'b 1000000;
    4'b 0001: hex = 7'b 1111001;
    4'b 0010: hex = 7'b 0100100;
    4'b 0011: hex = 7'b 0110000;
    4'b 0100: hex = 7'b 0011001;
    4'b 0101: hex = 7'b 0010010;
    4'b 0110: hex = 7'b 0000010;
    4'b 0111: hex = 7'b 1111000;
    4'b 1000: hex = 7'b 0000000;
    4'b 1001: hex = 7'b 0010000;
    4'b1010: hex = 7'b0001000; //A
    4'b1011: hex= 7'b0000011; //B
    4'b1100: hex = 7'b1000110; //C
    4'b1101: hex= 7'b0100001; //D
    4'b1110: hex = 7'b0000110; //E
    4'b1111: hex = 7'b0001110; //F
    endcase
endmodule
```

Bài 2:

```
module bai1(SW,LEDR,HEX0,HEX1);
    input [2:0]SW;
    output [9:0]LEDR;
    output[6:0]HEX0,HEX1;
    wire [7:0]alu_out;
    wire [7:0]A=8'b11111111;
    wire [7:0]B=8'b11111111;
    alu_logic(A,B,SW[2:0],alu_out,LEDR[8],LEDR[9]);
    assign LEDR[7:0]=alu_out;

    decoder (alu_out[3:0],HEX0);
    decoder (alu_out[7:4],HEX1);
endmodule

module alu_logic(A,B,opcode,alu_out,ovf,br);
    input [7:0]A,B;
    input [2:0]opcode;
    output reg[7:0]alu_out;
    output reg ovf,br;

    always @(*)
    begin
    case (opcode)
    3'b000:
        begin
        alu_out=A+B;

ovf=(A[7]&B[7]&~alu_out[7])|~(A[7]&~B[7]&alu_out[7])|(A[7]&B[7]&alu_out[7]
);
            br=0;
        end
    3'b001:
        begin
```

```
    alu_out=~A;
    ovf=0;
        br=0;
    end
3'b010:
    begin
    alu_out =A^B;
    ovf=0;
        br=0;
    end
3'b011:
    begin
    alu_out=~(A+B);
    ovf=0;
        br=0;
    end
3'b100:
    begin
    alu_out=A>>1;
    ovf = 0;
        br=0;
    end
3'b101:
    begin
    alu_out=B<<1;
    ovf=0;
        br=0;
    end
3'b110:
  begin
    br=(A==B)?1:0;
        ovf=0;
        end
3'b111:
```

```
      begin
        br=(A>B)?1:0;
            ovf=0;
            end
    endcase
    end
endmodule

module decoder(c,hex);
    input [7:0]c;
    output reg [6:0]hex;
    always@(c)
    case (c)
        4'b 0000: hex = 7'b 1000000;
        4'b 0001: hex = 7'b 1111001;
        4'b 0010: hex = 7'b 0100100;
        4'b 0011: hex = 7'b 0110000;
        4'b 0100: hex = 7'b 0011001;
        4'b 0101: hex = 7'b 0010010;
        4'b 0110: hex = 7'b 0000010;
        4'b 0111: hex = 7'b 1111000;
        4'b 1000: hex = 7'b 0000000;
        4'b 1001: hex = 7'b 0010000;
        4'b1010: hex = 7'b0001000; //A
        4'b1011: hex= 7'b0000011; //B
        4'b1100: hex = 7'b1000110; //C
        4'b1101: hex= 7'b0100001; //D
        4'b1110: hex = 7'b0000110; //E
        4'b1111: hex = 7'b0001110; //F
endcase
endmodule
```

Bài 3:

```
module cau3(SW,LEDR,HEX0,HEX1);
    input [3:0]SW;
```

```
    output [8:0]LEDR;
    output[6:0]HEX0,HEX1;
    wire [7:0]alu_out;
    wire [7:0]A=8'b11111111;
    wire [7:0]B=8'b11111110;
    alu_logic(A,B,SW[3:0],alu_out,LEDR[8]);
    assign LEDR[7:0]=alu_out;

    decoder (alu_out[3:0],HEX0);
    decoder (alu_out[7:4],HEX1);
endmodule

module alu_logic(A,B,opcode,alu_out,ovf);
    input [7:0]A,B;
    input [3:0]opcode;
    output reg[7:0]alu_out;
    output reg ovf;
    always @(*)
    begin
    case (opcode)
    4'b000:
        begin
        alu_out=A+B;
        ovf=(A[7]&B[7]&~alu_out[7])|~(A[7]&~B[7]&alu_out[7])|(A[7]&B[7]&al
u_out[7]);
        end
    4'b0001:
        begin
        alu_out=A-B;
        ovf=0;
        end
    4'b0010:
        begin
        alu_out =A+1;
```

```
    ovf=0;
    end
4'b0011:
    begin
    alu_out=~B;
    ovf=0;
    end
4'b0100:
    begin
    alu_out=A<<1;
    ovf = 0;
    end
4'b0101:
    begin
    alu_out=A>>1;
    ovf=0;
    end
4'b0110:
    begin
    alu_out={A[6:0],A[7]};
    ovf=0;
    end
4'b0111:
    begin
    alu_out={A[0],A[7:1]};
    ovf=0;
    end
4'b1000:
    begin
    alu_out=A&B;
    ovf=0;
    end
4'b1001:
    begin
```

```
    alu_out=A|B;
    ovf=0;
    end
4'b1010:
    begin
    alu_out=A^B;
    ovf=0;
    end
4'b1011:
    begin
    alu_out=~(A&B);
    ovf=0;
    end
4'b1100:
    begin
    alu_out=~(A^B);
    ovf=0;
    end
4'b1101:
    begin
    alu_out=A&B;
    ovf=0;
    end
4'b1110:
    begin
    alu_out=(A>B)?8'b11111111:8'b00000000;
    ovf=0;
    end
4'b1111:
    begin
    alu_out=(A==B)?8'b11111111:8'b00000000;
    ovf=0;
    end
endcase
```

```
    end
endmodule

module decoder(c,hex);
    input [7:0]c;
    output reg [6:0]hex;
    always@(c)
    case (c)
        4'b 0000: hex = 7'b 1000000;
        4'b 0001: hex = 7'b 1111001;
        4'b 0010: hex = 7'b 0100100;
        4'b 0011: hex = 7'b 0110000;
        4'b 0100: hex = 7'b 0011001;
        4'b 0101: hex = 7'b 0010010;
        4'b 0110: hex = 7'b 0000010;
        4'b 0111: hex = 7'b 1111000;
        4'b 1000: hex = 7'b 0000000;
        4'b 1001: hex = 7'b 0010000;
        4'b1010: hex = 7'b0001000; //A
        4'b1011: hex= 7'b0000011; //B
        4'b1100: hex = 7'b1000110; //C
        4'b1101: hex= 7'b0100001; //D
        4'b1110: hex = 7'b0000110; //E
        4'b1111: hex = 7'b0001110; //F
    endcase
endmodule
```

Bài tập tuần 10

Yêu cầu của bài thực hành

Câu 1: Sử dụng Multiplexer (MUX) để tổng hợp chức năng logic và viết code verilog mô tả mạch chức năng logic.

a)

w_1	w_2	w_3	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

b)

w_1	w_2	w_3	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

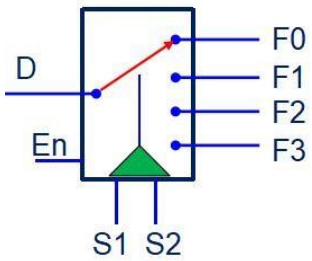
c)

w_1	w_2	w_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Kiểm tra hoạt động của các mạch chức năng logic trên board DE1, sử dụng các tín hiện trên board như sau:

- SW[0] = w1
- SW[1] = w2
- SW[2] = w3
- LEDR[0] = f

Câu 2: Cho mạch tách kênh (demultiplexer) như sau:

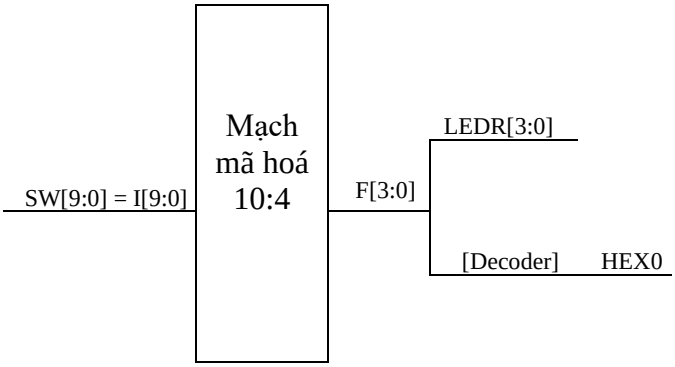


En	S1	S0	Ngõ ra
0	x	x	F0=F1=F2=F3=0
1	0	0	F0 = D
1	0	1	F1 = D
1	1	0	F2 = D
1	1	1	F3 = D

Mạch tách kênh từ 1 đường sang 4 đường nên có 2 ngõ chọn (S1, S2). Khi ngõ cho phép En ở mức 0 thì cấm không cho phép dữ liệu vào được truyền ra ở bất kì ngõ ra nào nên tất cả các ngõ ra đều bằng 0. Khi En ở mức 1, nếu ngõ vào chọn S1S0=00 thì dữ liệu D được đưa ra ngõ F0, các ngõ khác không đổi. Tương tự với các tổ hợp S1S0 khác thì lần lượt D sẽ ra ở ngõ F1, F2 và F3.

Viết code verilog mô tả mạch tách kênh 1 sang 4 trên và kiểm tra hoạt động của mạch trên board DE1. Sử dụng các tín hiệu trên board như sau: SW[0]=D; SW[1]=S1; SW[2]=S2; SW[3]=En; LEDR[0]=F[0]; LEDR[1]=F[1]; LEDR[2]=F[2]; LEDR[3]=F[3].

Câu 3: Viết code verilog mô tả mạch mã hoá 10 đường sang 4 đường như sau:



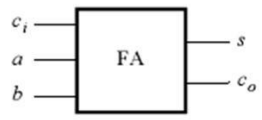
I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	F3	F2	F1	F0
1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0	0	0	0	1	0	1
0	0	0	0	0	0	1	0	0	0	0	1	1	0
0	0	0	0	0	0	0	1	0	0	0	1	1	1
0	0	0	0	0	0	0	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	0	1	1	0	0	1

Câu 4: Viết code verilog mô tả mạch thực hiện chức năng chuyển đổi số hệ nhị phân 4 bit (B[3:0]) thành số hệ thập phân 2 chữ số (d1 d0). Kiểm tra hoạt động của mạch trên board DE1, sử dụng các tín hiệu: - SW[3:0] = B[3:0] : làm 4 bit ngõ vào

- HEX1, HEX0 = d1, d0 : hiển thị số thập phân 2 chữ số

B[3:0]	d1	d0
0000	0	0
0001	0	1
0010	0	2
...	...	...
1001	0	9
1010	1	0
1011	1	1
1100	1	2
1101	1	3
1110	1	4
1111	1	5

Câu 5: Viết code verilog mô tả mạch Full Adder 4 bit được ghép từ 4 Full Adder 1 bit. Cho mạch Full Adder 1 bit như sau:



Full adder symbol

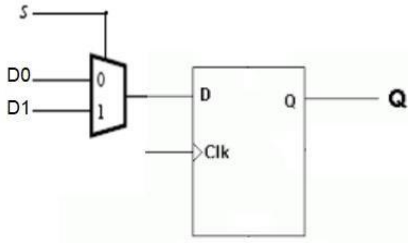
b	a	c_i	c_o	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Full adder truth table

```
module fulladder1bit(cin,a,b,s,cout);
  input  cin,a,b;

  output s,cout;
  assign s=a^b^cin;

  assign cout=(a&b)|(cin&a)|(cin&b);
```

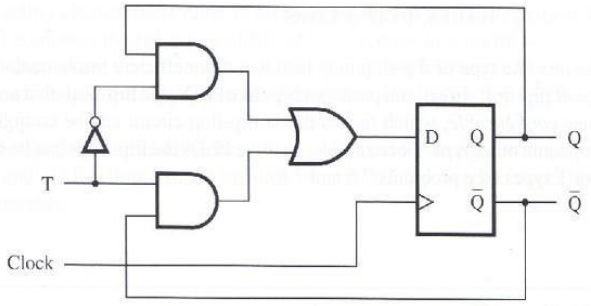
Câu 6: Viết code verilog mô tả mạch Full Adder 4 bit sử dụng **parameter n=4**.

Câu 7. Viết code verilog mô tả mạch Flip-Flop D với Mux 2:1 ở ngõ vào D (4 bits) như hình sau:

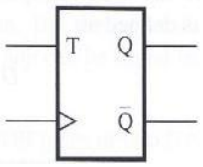


Kiểm tra mạch trên Board DE1 bằng cách sử dụng các tín hiệu sau:
- SW[3:0] = D0
- SW[7:4] = D1
- SW[9] = s
- LEDR[3:0] = Q

Câu 8. Viết code verilog mô tả mạch Flip-Flop T như sau:

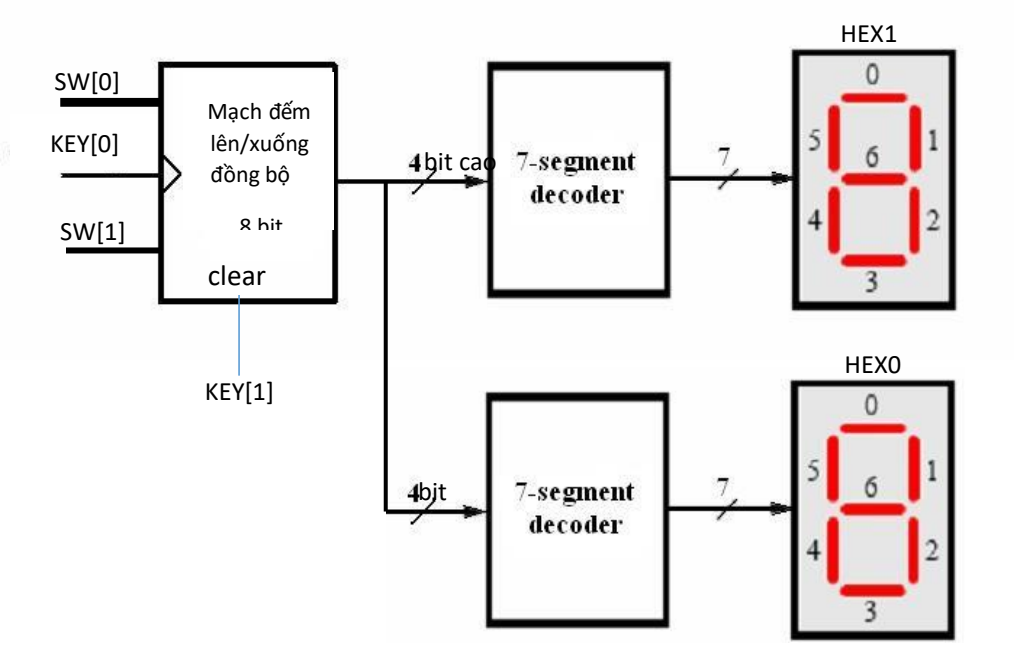


T	Q(t+1)
0	Q(t)
1	$\bar{Q}(t)$



Kiểm tra mạch trên Board DE1 bằng cách sử dụng các tín hiệu:
- SW[9] = T
- KEY[0] = Clk
- LEDR[0] = Q

Câu 9: Viết code verilog mô tả mạch đếm lên/xuống đồng bộ 8 bit sử dụng biểu thức toán học.



Yêu cầu sử dụng các tín hiệu trên board như sau:

- SW[0] làm tín hiệu Enable.
- SW[1] làm tín hiệu Select: nếu SW[1]=0 thực hiện mạch đếm lên, SW[1]=1 thực hiện mạch đếm xuống.
- KEY[0] làm tín hiệu xung clock.
- KEY[1] làm tín hiệu clear bộ đếm về 0.
- HEX1, HEX0 hiển thị kết quả của bộ đếm lên từ 00 <--> FF

Câu 10: Tương tự câu 9, nhưng thay tín hiệu xung clock bằng việc sử dụng bộ dao động CLOCK_50MHZ để tạo trì hoãn 1/2 giây.

CODE

Bài 1:

```
module cau1(SW, LEDR);
    input [2:0]SW;
    output [0:0]LEDR;
    wire f0;

    //Cau a
    //mux2_1(SW[2], ~SW[2], SW[1], f0);
    //mux2_1(f0, ~f0, SW[0], LEDR[0]);
```

```
    //Cau b
    mux2_1(SW[1]&SW[2], SW[1]|SW[2], SW[0], LEDR[0]);
endmodule

module mux2_1(x, y, s, m_out);
    input x, y, s;
    output m_out;

    assign m_out = s?y:x;
endmodule
```

Bài 2:

```
module cau2(SW, LEDR);
    input [3:0]SW;
    output [3:0]LEDR;

    demux1_4(SW[0], SW[2:1], SW[3], LEDR);
endmodule

module demux1_4(d, s, en, f);
    input d, en;
    input [1:0]s;
    output reg [3:0]f;

    always@(en, s)
    if(~en)
        begin
```

```
            f[0] = 0;
            f[1] = 0;
            f[2] = 0;
            f[3] = 0;
        end
    else
        begin
            //always@(s)
            case(s)
                2'b00: f[0] = d;
                2'b01: f[1] = d;
                2'b10: f[2] = d;
                2'b11: f[3] = d;
            endcase
        end
endmodule
```

Bài 3:

```
module cau3(SW, LEDR, HEX0);
    input [0:9]SW;
    output [3:0]LEDR;
    output [6:0]HEX0;

    decode_10_4(SW, LEDR);
    display_0_9(LEDR, HEX0);
endmodule
```

```
module decode_10_4(i, f);
    input [9:0]i;
    output reg [3:0]f;

    always@(i)
    case(i)
        10'b1000000000: f = 4'b0000;
        10'b0100000000: f = 4'b0001;
        10'b0010000000: f = 4'b0010;
        10'b0001000000: f = 4'b0011;
        10'b0000100000: f = 4'b0100;
        10'b0000010000: f = 4'b0101;
        10'b0000001000: f = 4'b0110;
        10'b0000000100: f = 4'b0111;
        10'b0000000010: f = 4'b1000;
        10'b0000000001: f = 4'b1001;
    endcase
endmodule

module display_0_9(c,hex);
    input [3:0]c;
    output reg [6:0]hex;
    always@(c)
        case(c)
            4'b0000 :hex =7'b1000000;
            4'b0001 :hex =7'b1111001;
            4'b0010 :hex =7'b0100100;
            4'b0011 :hex =7'b0110000;
```

```
            4'b0100 :hex =7'b0011001;
            4'b0101 :hex =7'b0010010;
            4'b0110 :hex =7'b0000010;
            4'b0111 :hex =7'b1111000;
            4'b1000 :hex =7'b0000000;
            4'b1001 :hex =7'b0010000;
            default :hex =7'b1111111;
        endcase
endmodule
```

Bài 4:

```
module cau4(SW, HEX0, HEX1);
     input [3:0]SW;
     output [0:6]HEX0, HEX1;
     seg70(SW[3:0], HEX0, HEX1);
endmodule


module seg70(bcd, led0, led1);
     input [3:0] bcd;
     output reg [0:6]led0, led1;

     always @(bcd)
     if (bcd<10)
          begin
          led1 = 7'b1111111;
          case (bcd)       //abcdefg
               0: led0 = 7'b0000001;
```

```
                1: led0 = 7'b1001111;
                2: led0 = 7'b0010010;
                3: led0 = 7'b0000110;
                4: led0 = 7'b1001100;
                5: led0 = 7'b0100100;
                6: led0 = 7'b0100000;
                7: led0 = 7'b0001111;
                8: led0 = 7'b0000000;
                9: led0 = 7'b0000100;
                default: led0 = 7'b1111111;
          endcase
          end
     else
          begin
          led1 = 7'b1001111;
          case (bcd)
                10: led0 = 7'b0000001;
                11: led0 = 7'b1001111;
                12: led0 = 7'b0010010;
                13: led0 = 7'b0000110;
                14: led0 = 7'b1001100;
                15: led0 = 7'b0100100;
          endcase
          end
endmodule
```

Bài 5:

```
module cau5(SW,LEDR);
    input [8:0] SW;
    output [4:0] LEDR;
    add4bit(SW[4], SW[3:0], SW[8:5], LEDR[3:0], LEDR[4]);
endmodule

module add4bit(cin,a,b,s,c);
    input cin;
    input [3:0]a,b;
    output [3:0]s;
    output [4:1] c;
    fulladder(cin, a[0], b[0], s[0], c[1]);
    fulladder(c[1:1], a[1:1], b[1:1], s[1:1], c[2:2]);
    fulladder(c[2:2], a[2:2], b[2:2], s[2:2], c[3:3]);
    fulladder(c[3:3], a[3:3], b[3:3], s[3:3], c[4:4]);
endmodule

module fulladder(cin,a,b,s,cout);
    input cin,a,b;
    output s,cout;
    assign s=a^b^cin;
    assign cout=(a&b)|(cin&a)|(cin&b);
endmodule
```

Bài 6:

```
module cau6(SW,LEDR);
    input [8:0]SW;
```

```
    output [4:0]LEDR;
    fulladder4bit (SW[8], SW[3:0], SW[7:4], LEDR[3:0], LEDR[4]);
endmodule

module fulladder4bit(cin, a[3:0], b[3:0], s[3:0], cout);
    parameter n=4;
    input [n-1:0]a,b;
    input cin;
    output reg [n-1:0]s;
    output reg cout;
    reg [n:0]c;
    integer k;

    always @(a,b,cin)
    begin
        c[0]=cin;
        for (k=0;k<n;k=k+1)
            begin
                s[k]=a[k]^b[k]^c[k];
                c[k+1]=(a[k]&b[k])|(a[k]&c[k])|(c[k]&b[k]);
            end
        cout = c[n];
    end
endmodule
```

Bài 7:

```
module cau2(SW, KEY, LEDR);
    input [9:0]SW; // SW[3:0] = D0, SW[7:4] = D1, SW[8] = s
```

```
    input [0:0]KEY;
    output wire [3:0]LEDR; // LEDR[3:0] = q
    wire [3:0]d;

    mux_21_4b(SW[3:0], SW[7:4], SW[9], d);
    FF_D_4b(d, KEY[0], LEDR[3:0]);
endmodule

module mux_21_4b(x, y, s, m);
    input [3:0]x, y;
    input s;
    output [3:0]m;

    assign m = s?y:x;
endmodule

module FF_D_4b(d, clk, q);
    input [3:0]d;
    input clk;
    output reg [3:0]q;

    always @(posedge clk)
        q <= d;
endmodule
```

Bài 8:

```
module cau8(SW, KEY, LEDR);
    input [9:9]SW;
```

```
    input [0:0]KEY; // t = SW9, clk = KEY0
    output [0:0]LEDR;   // q = LEDR

    FF_T(SW[9], KEY[0], LEDR[0]);
endmodule

module FF_T(t, clk, q);
    input t, clk;
    output q;
    wire m1, m2, d;

    and(m1, ~t, q);
    and(m2, t, ~q);
    or(d, m1, m2);
    FF_D(d, clk, q);
endmodule

module FF_D(d, clk, q);
    input d, clk;
    output reg q;

    always @(posedge clk)
        q <= d;
endmodule
```

Bài 9:

```
module cau9(SW, KEY, HEX0, HEX1);
    input [1:0]SW;
```

```
    input [1:0]KEY;
    output [6:0]HEX0;
    output [6:0]HEX1;
    wire [7:0]bcd;

    counter(SW[0], SW[1], KEY[1], KEY[0], bcd);
    Decoder_HEX(bcd[3:0], HEX0);
    Decoder_HEX(bcd[7:4], HEX1);
endmodule

module counter(en, s, rst, clk, bcd);
    parameter n = 8;
    input s, en, rst, clk;
    output reg [n-1:0]bcd;

    always@(posedge clk or negedge rst)
        if(!rst)
            bcd <= 0;
        else if(en)
            bcd <= bcd + (s?-1:1);
endmodule

module Decoder_HEX(c,HEX);
    input[3:0]c;
    output [6:0]HEX;
    reg[6:0]HEX;
    always@(c)
        case(c)
```

```
        4'b0000: HEX = 7'b1000000;
        4'b0001: HEX = 7'b1111001;
        4'b0010: HEX = 7'b0100100;
        4'b0011: HEX = 7'b0110000;
        4'b0100: HEX = 7'b0011001;
        4'b0101: HEX = 7'b0010010;
        4'b0110: HEX = 7'b0000010;
        4'b0111: HEX = 7'b1111000;
        4'b1000: HEX = 7'b0000000;
        4'b1001: HEX = 7'b0010000;
        4'b1010: HEX = 7'b0001000; //A
        4'b1011: HEX = 7'b0000011; //B
        4'b1100: HEX = 7'b1000110; //C
        4'b1101: HEX = 7'b0100001; //D
        4'b1110: HEX = 7'b0000110; //E
        4'b1111: HEX = 7'b0001110; //F
        default: HEX = 7'b1111111;
    endcase
endmodule
```

Bài 10:

```
module bai1(CLOCK_50,KEY,SW,HEX0, HEX1);
    input CLOCK_50;
    input [0:0]KEY;
    input [1:0]SW;
    output [6:0]HEX0, HEX1;
    wire [3:0]a,b;
```

```
        counter (CLOCK_50, KEY[0],SW[0],SW[1], a,b);
        display_0_9 (a, HEX0);
        display_0_9 (b, HEX1);
endmodule

module counter(CLOCK_50,reset,enable,select,BCD0, BCD1);
    input reset,CLOCK_50,enable,select;
    output reg[3:0]BCD0, BCD1;
    reg [25:0]delay;
    //tao delay 1s
    always @(posedge CLOCK_50)
        begin
            if(delay==25000000)
                delay<=0;
            else
            delay=delay+1;
        end
    always@(posedge CLOCK_50 or negedge reset)
        if(!reset)
            begin
                BCD0<=9;
                BCD1<=9;
            end
        else
        if (enable)
        begin
            if(select==0)
            begin
```

```
if(delay==0)
        begin
            if(BCD0==0)
                begin
                    BCD0<=9;
                        if(BCD1==0)
                            BCD1<=9;
                        else
                            BCD1<=BCD1-1;
                end
            else
                BCD0<=BCD0-1;
        end
        end


        else
            begin
if(delay==0)
        begin
            if(BCD0==9)
                begin
                    BCD0<=0;
                        if(BCD1==9)
                            BCD1<=0;
                        else
                            BCD1<=BCD1+1;


                    end
```

```
                        else
                            BCD0<=BCD0+1;

                    end
                    end
                    end
endmodule

module display_0_9(c,hex);
    input [3:0]c;
    output reg [6:0]hex;
    always@(c)
        case(c)
            4'b0000 :hex =7'b1000000;
            4'b0001 :hex =7'b1111001;
            4'b0010 :hex =7'b0100100;
            4'b0011 :hex =7'b0110000;
            4'b0100 :hex =7'b0011001;
            4'b0101 :hex =7'b0010010;
            4'b0110 :hex =7'b0000010;
            4'b0111 :hex =7'b1111000;
            4'b1000 :hex =7'b0000000;
            4'b1001 :hex =7'b0010000;
            default :hex =7'b1111111;
        endcase
endmodule
```