BÀI 1.2: THIẾT KẾ GIAO TIẾP VÀO RA

1.2.1 Mục tiêu bài thực hành

Mục đích của bài thực hành này là làm thế nào để kết nối các thiết bị xuất, nhập thông qua chip FPGA và thực hiện một mạch điện với các thiết bị như: switches, lights, và multiplexers. Trong bài này chúng ta sử dụng các công tắc SW_{9-0} có sẵn trên Board DE1 như là các thiết bị nhập (input devices), sử dụng các đèn LEDs và đèn LEDs 7 đoạn (7- segment displays) như là các thiết bị đầu ra (output devices).

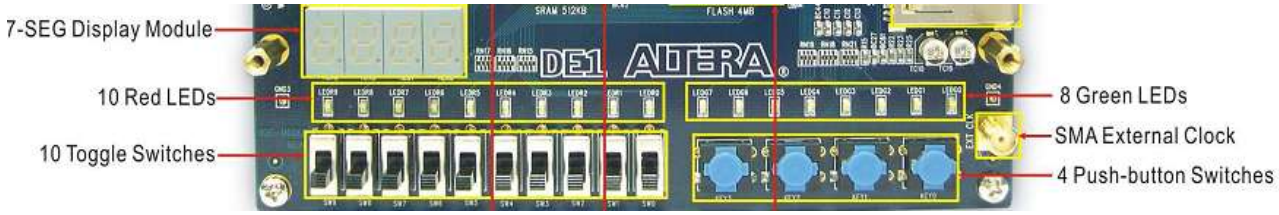
1.2.2 Yêu cầu của bài thực hành

1. Thực hiện một module hiển thị toàn bộ 10 LEDs đỏ ($LEDR_{9-0}$) được điều kiển bởi 10 công tắc (SW_{9-0}). Mỗi công tắc điều khiển một đèn LEDs.
2. Thiết kế một module thực hiện một bộ multiplexer 2-to-1 độ rộng 4-bit.
3. Thiết kế một module thực hiện một bộ multiplexer 5-to-1 (1bit) dựa trên 4 bộ multiplexer 2-to-1
4. Thiết kế bộ giải mã LED 7 đoạn với 3-bit ngõ vào hiển thị các ký tự H, E, L, O.
5. Thiết kế mạch hiển thị chuỗi ký tự "F, P, G, A" lên 4 Led 7 đoạn bằng cách kết hợp bộ giải mã LED 7 đoạn với 3-bit ngõ vào bằng 000.
6. Thiết kế một mạch điện hiển thị chuỗi ký tự "F, P, G, A, 'blank', 'blank', 'blank' " và có thể xoay các ký tự này từ trái sang phải. ('blank' = không hiển thị).
7. Thiết kế bộ giải mã số BCD sang LED 7 đoạn.

Lưu ý: trong các bài trên chỉ sử dụng câu lệnh gán ***assgin.*** Sinh viên sau khi thực hiện hết các bài tập này bằng câu lệnh assgin thì có thể mở rộng bằng các sử dụng *if - else, switch-case* .

1.2.3 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], …, SW[9]➔LEDR[9].



Hình 1.35: Sơ đồ Led đỏ và công tắt [22]

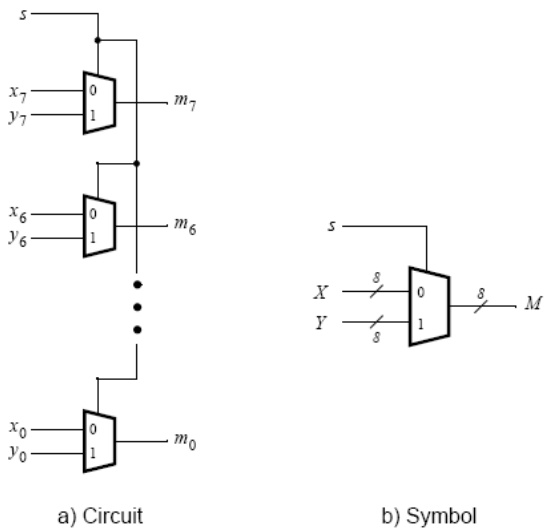
Ta tạo module Verilog sau:

```
module  Sw_Light  (
SW,
LEDR
 // dấu 2 gạch là bắt đầu câu chú thích. Luôn có dấu ‘,’ ngăn cách các khai báo biến.

// Các tên định danh cho các thành phần trên board DE1 luôn viết hoa chẳng hạn SW là tên
định //dành cho các toggle switch nó hiểu khác khi ta khai báo sw.

/* đây cũng là ký tự bắt đầu câu chú thích */

) ; // luôn có dấu ‘ ; ’ ở đây.

input [9:0]SW;    /* các keyword luôn viết thường. Với các khai báo trên có nghĩa là có 10
toggle switch . */

output [9:0]LEDR // có 10 đèn led đỏ sẽ được sử dụng.

/* kết thúc khai báo */
/* bắt đầu đoạn code*/
assign LEDR[0]  = SW[0] // lưu ý cách sử dụng assign và các khai báo các ngõ vào và ngõ ra.
assign LEDR[1]  = SW[1]
. . .
assign LEDR[9] = SW[9]

/* kết thúc đoạn code */
/* đoạn code trên để chỉ cho ta thấy lệnh các bước tuần tự nếu ta thực hiên công tắc thứ 0 điều
khiển đèn led đỏ thứ không , tiếp tục như vậy cho đến hết.*/
/* ta có thể làm cách khác như sau:
assign LEDR = SW;// verilog vẫn hiểu  thay vì chúng ta viết tường minh như đoạn code ở
trên.
*/
endmodule // đừng quên từ khóa này mỗi khi kết thúc 1 module.
```

Sau khi nạp vào FPGA ta kiểm tra từng công tắc cũng như từng đèn led.

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 mạch và ký hiệu của bộ multiplex 2–to-1 (8bit) [22]

- Sau khi sinh viên hoàn tất phần này bằng lệnh **assign** nên thực hiện bằng lệnh Switch – case, if – else, toán tử điều kiện< **? : >.**

Tạo module verilog cho bộ multiplex 2-to-1 sau:

```
module  m_2_to_1 (
x_in, // ngõ vào
y_in,// ngõ vào
select,// ngõ vào
m_out// ngõ ra );
        /*****************/
        input   x_in,  y_in, select;
        ouput   m_out;
        /*****************/
        // Bắt đầu đoạn code
        /*****************/
        assign  m_out =  ( x_in  & ~select)  |  ( y_in & select );
        // kết thúc đoạn code
        /*****************/
endmodule
```

Dựa vào đoạn code trên để tạo ra module m_2_to_1_4bit sau :

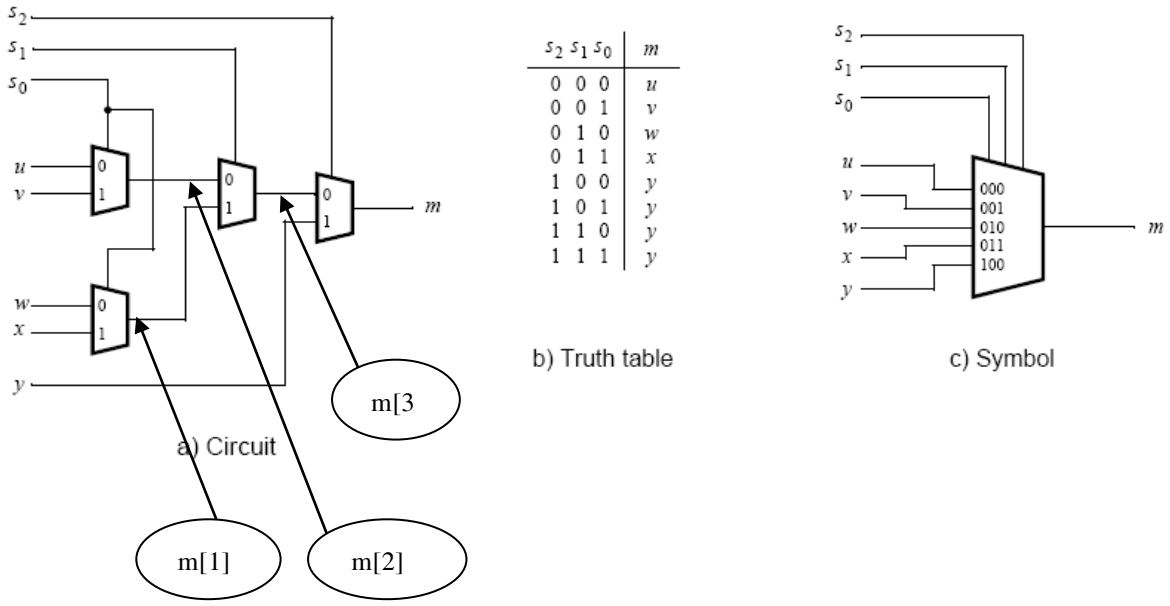
```
module  m_2_to_1_4bit_w (
x_in,
y_in,
select,
m_out
);
        /************************/
        input   [3:0] x_in, [3:0]y_in, select;
        ouput   [3:0] m_out;
        /************************/
        // bắt đầu đoạn code
        /************************/
        assign m_out[0]  = ( x_in[0] & ~select ) | ( y_in[0] & select );
        assign m_out [1] = ( x_in[1] & ~select ) | ( y_in[1] & select );
        …
endmodule
```

Lưu ý *: đây là các module con, không phải* ***top-module.***

Sau khi thực hiện xong đoạn code này bằng lệnh assign ta nên thử với lệnh ***case, if – else****;*
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 sẽ 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 mạch, bảng sự thật và ký hiệu của bộ multiplex 5–to-1 [22]

Hình trên là bộ multiplex 5-to-1 với độ rộng 1 bit ➔ ta sẽ mở rộng ra bộ multiplex 5-to-1 độ rộng 3 bit như thế nào ?

Gợi ý: Ta thực hiện tuần tự các ngõ ra : m[1], m[2], m[3], m bằng lệnh **assgin** như là với bộ multiplex 2 – to – 1 .

Bây giờ chúng ta tạo ra bộ multiplex 5_to_1_1bit_w:

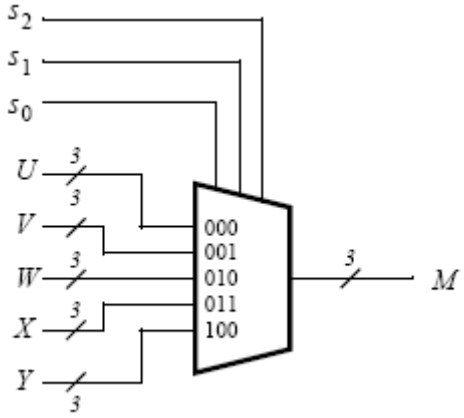
```
module  m_5_to_1_1bit_w ( u, v, w, x, y, s, m_out);
        /*****************************/
        input           u, v, w, x, y;
        input           [2:0]s;
        output          m_out;
        wire      m1, m2, m3 ;/* đây là các biến tạm không tham gia vào các ngõ ra hay ngõ vào.*/
        /*****************************/
        // Bắt đầu đoạn code //
        /*****************************/
        assign   m1 = ( u & s[0] ) | ( v & ~s[0] ) ;
        assign   m2 = ( w & s[0] ) | (x & ~s[0] ) ;
        assign   m3 =  ( m1 & s[1]) | (m2 & ~s[1] );
        assign   m_out  = ( y & s[2] ) | ( m3 & ~ s[2] ) ;
        // kết thúc đoạn code
```

/*****************************/

endmodule

Để kiểm tra trên board DE1 ta sử dụng các thành phần sau: SW[2:0] sử dụng làm select (s0, s1, s2), các SW[7:3] làm các cổng vào (từ u tới y), LEDR[0] làm cổng ra m.

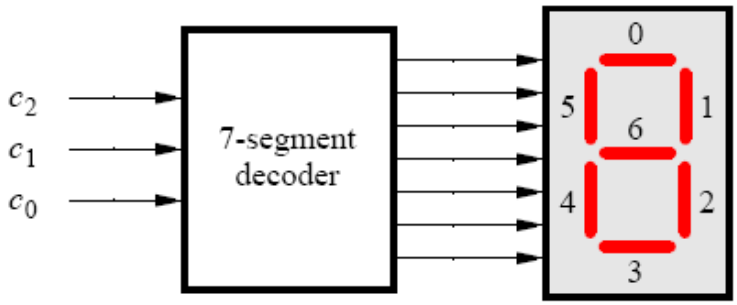
Dựa vào đoạn code trên bạn hãy xây dựng một module m_5_to_1_3bit_w ?



Hình 1.38: Bộ multiplex 5-to-1 độ rộng 3 bit [22]

4. Thiết kế bộ giải mã LED 7 đoạn với 3-bit ngõ hiển thị các ký tự H, E, L, O: trong phần này chúng ta sẽ tạo ra một module giải mã (decode) các bit đầu ra thành các ký tự ‘ H ’, ‘E’, ‘L’, ‘O’ với một đèn LED 7 đoạn.

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



Hình 1.39: Bộ giải mã 7 đoạn [22]

- Trong đó ta có 3 ngõ vào C_0, C_1, C_2 đi vào bộ giải mã 7 đoạn rồi xuất ra đèn led.
- Lưu ý 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 có 3 bit đầu vào nên ta có thể mã hóa 8 trạng thái khác nhau nhưng chỉ lấy 4 trạng thái đầu, các trạng thái còn lại không quan tâm.

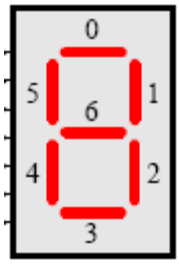
$c_2\ c_1\ c_0$	Ký tự
000	H
001	E
010	L
011	O
100	
101	
110	
111	

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

Trước khi ta thực hiện module decoder_3bit_4C ta thực hiện các bước sau:

Bước 1: Lập bảng sự thật

C2	C1	C0	Char	H0	H1	H2	H3	H4	H5	H6
0	0	0	"H"	1	0	0	1	0	0	0
0	0	1	"E"	0	1	1	0	0	0	0
0	1	0	"L"	1	1	1	0	0	0	1
0	1	1	"O"	0	0	0	0	0	0	1
1	0	0	Blank	1	1	1	1	1	1	1
1	0	1	Blank	1	1	1	1	1	1	1
1	1	0	Blank	1	1	1	1	1	1	1
1	1	1	Blank	1	1	1	1	1	1	1



Cột bên phải là các ngõ ra đèn led 7 đoạn , cột bên trái là các ngõ vào (SW), "blank" nghĩa là các ngõ ra đèn led 7 đoạn ở mức logic 1 (không sáng).

Bước 2: Dùng giản đồ Karnaugh để đơn giản mạch.

Bước 3: Thực hiện thiết kế module. (sử dụng **assign** sau đó dùng **case**).

Để kiểm tra trên board DE1, ta sử dụng các thành phần sau: SW[2:0] làm các ngõ vào (C_0, C_1, C_2). Trên board DE1 có 4 đèn LED 7 đoạn có tên định danh sau: HEX0, …, HEX3. Mỗi đèn HEX được khai báo output [6:0]HEX0 hay [0:6]HEX0 đều được. Trong phần này ta sử dụng HEX0.

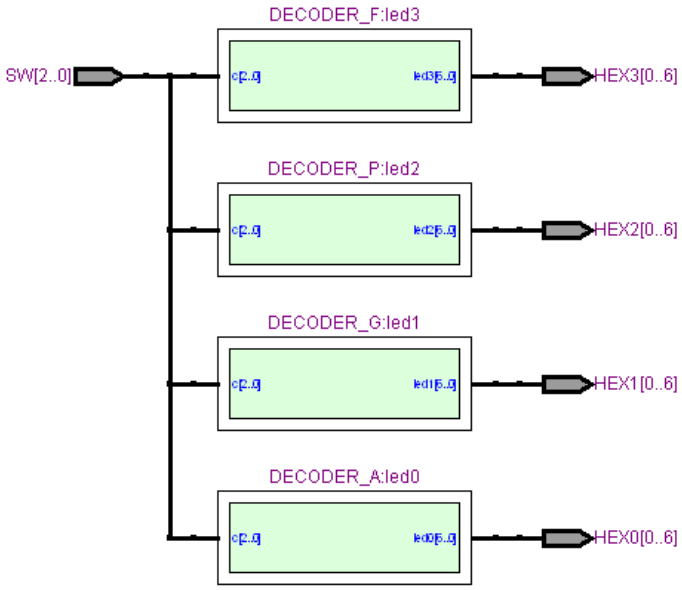
5. Thiết kế bộ giải mã để thực hiện việc hiển thị chữ FPGA trên 4 led 7 đoạn

HEX3 -> hiển thị chữ F

HEX2 -> hiển thị chữ P

HEX1 -> hiển thị chữ G

HEX0 -> hiển thị chữ A



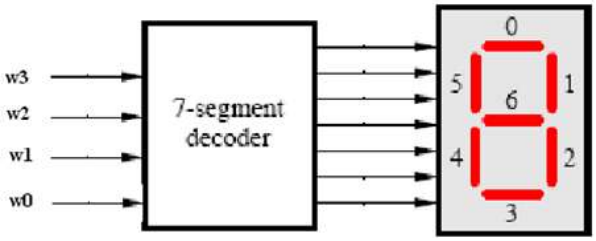
Hình 1.40: Hiển thị chữ FPGA trên 4 led 7 đoạn

6. Mở rộng câu 5 chúng ta sẽ 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 chỉnh. 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 hiển thị trên 4 led 7 đoạn

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 mã BCD sang LED 7 đoạn

Bảng 1.2: Bảng sự thật của bộ giải mã BCD sang LED 7 đoạn

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 ứng cho các tín hiệu vào w0, w1, w2, w3 và LED 7 đoạn HEX0 để thị giá trị thập phân.