TUẦN 2 - MẠCH TỔ HỢP

Bài thực hành này có nội dung về mạch tổ hợp, sử dụng các toán tử, vector, và khởi tạo module trong ngôn ngữ mô tả phần cứng Verilog

Nội dung sinh viên cần ôn tập trước buổi thực hành

- Khái niệm về mạch tổ hợp
- Các toán tử trong ngôn ngữ Verilog HDL

Mạch tổ hợp – Combinational circuit

Mạch tổ hợp là mạch được thiết kế bằng cách sử dụng nhiều cổng logic được kết nối với nhau, sao cho đầu ra được tạo ra chỉ bằng cách tính toán các tổ hợp logic của các đầu vào hiện tại. Mạch không hoạt động dựa trên xung clock. Hơn nữa, không có giá trị hoặc trạng thái nào được lưu trữ trước đó được xem xét. Đầu ra hoàn toàn không phụ thuộc vào các trạng thái trước đó, mà chỉ phụ thuộc vào đầu vào ngay tức thời. Tổng hợp các đặc tính của mạch tổ hợp như sau:

- Đầu ra chỉ phụ thuộc vào đầu vào hiện tại, không có khả năng lưu trữ trạng thái
- Tốc độ xử lý nhanh, thiết kế đơn giản, dễ dàng
- Không có phản hồi giữa đầu vào và đầu ra, không phụ thuộc vào thời gian
- Khối xây dựng cơ bản là các cổng logic
- Được sử dụng cho cả phép toán số học và logic Boolean

Ở bài thực hành Tuần 1, ta đã thực hiện các mạch tổ hợp thực hiện nhiều nhiệm vụ, chức năng khác nhau. Các mạch này nhận tín hiệu đầu vào từ Switch (SW) hoặc Button (KEY) và cho ra các kết quả hiện thị LED hoặc LED 7 đoạn (HEX). Câu lệnh **“assign”** trong Verilog thường được sử dụng để điều khiển một tín hiệu có kiểu dữ liệu **“wire”** và khi tổng hợp sẽ được ánh xạ thành **mạch tổ hợp**.

Toán tử trong Verilog HDL

Các toán tử (Operator) số học (Arithmetic) và logic (Logical) thường được sử dụng để tạo ra các biểu thức. Đối với Verilog HDL, các toán tử số học và logic có phần giống với ngôn ngữ lập trình C/C++. Các bảng dưới đây tổng hợp các toán tử được dùng trong Verilog HDL.

Bảng 2.1 - Toán tử quan hệ (Relational)

Toán tử	Ý nghĩa và sử dụng

<	a < b; // a nhỏ hơn b?
	// trả giá trị 1 bit true/false

>	a > b; // a lớn hơn b?
	// trả giá trị 1 bit true/false

>=	a >= b // a lớn hơn hoặc bằng b? // trả giá trị 1 bit true/false
<=	a <= b // a nhỏ hơn hoặc bằng b? // trả giá trị 1 bit true/false

Bảng 2.2 - Toán tử số học (Arithmetic)

Toán tử	Ý nghĩa và sử dụng

*	c = a * b // nhân a với b
	// kết quả gán vào c

/	c = a / b // chia nguyên a cho b
	// kết quả gán vào c

+	sum = a + b // a cộng b
	// kết quả gán vào sum

-	diff = a - b // a trừ b
	// kết quả gán vào diff

%	amodb = a % b // chia nguyên lấy dư a cho (b)

Bảng 2.3 - Toán tử số học (Boolean)

Toán tử	Ý nghĩa và sử dụng
&&	a && b // a "đúng" VÀ b "đúng"? // trả về 1 bit true/fasle
||	a || b // a "đúng" HOẶC b "đúng"? // trả về 1 bit true/fasle

!	if(!a); // nếu a "không đúng"
	c = b; // thực hiện gán b vào c

Bảng 2.4 - Toán tử về giá trị và đẳng thức

Toán tử	Ý nghĩa và sử dụng
=	c = a // gán giá trị a vào c
==	c == a /* so sánh a có bằng b hay không?; chỉ so sánh giá trị bit 0 hoặc 1, và trả về true/false. So sánh các giá trị không xác định (X và Z) trả về kết quả false */
!=	c != a // c "không bằng" a, trả về giá trị 1 bit true/false; chỉ xét giá trị bit 0 và 1
===	c === a // c "hoàn toàn bằng" a, so sánh tất cả trường hợp 0, 1, X, Z, trả về 1 bit true/false
!==	c === a // c "hoàn toàn khác" a, so sánh tất cả trường hợp 0, 1, X, Z, trả về 1 bit true/false

Bảng 2.5 – Toán tử đơn, toán tử bit và rút gọn bit

Toán tử	Ý nghĩa và sử dụng
+	Phép cộng số học theo bit
-	Phép trừ số học theo bit

&	b = &a // phép AND tất cả các bit của a, gán vào b
	// VD: a = 3'b101 thì b = 1'b0
	// a = 3'b111 thì b = 1'b1

|	b = &a // phép OR tất cả các bit của a, gán vào b
^	b = &a // phép XOR tất cả các bit của a, gán vào b
~&, ~|, ~^	Tương tự như trên với NAND, NOR, và XNOR – phép rút gọn bit

~, &, |, ^	b = ~a; // gán ĐẢO của a (bit) vào b
	c = b & a // AND các bit của a và b
	c = b | a // OR các bit của a và b
	c = b ^ a // XOR các bit của a và b

~&, ~|, ~^	Tương tự như trên với NAND, NOT, và XNOR – phép tính trên bit

Bảng 2.6 - Các toán tử khác

<<	a << 1;
	// dịch sang trái 1 bit
	// VD: a = 3'b001 thì a = 3'010

>>	b >> 1;
	// dịch sang phải 1 bit
	// VD: b = 3'b100 thì b = 3'010

? :	c = sel ? a : b; /* nếu sel "đúng", gán c = a, ngược lại gán c = b */

BIT và VECTOR

Trong Verilog HDL, các tín hiệu **wire** và **reg** (sẽ được học sau) có thể được khai báo dưới dạng **vector**, cho phép xử lý nhiều bit cùng lúc. Ngoài ra, có thể sử dụng phương thức **bit slicing** để truy cập một phần cụ thể của vector và **concatenation** để kết hợp và biến đổi nhiều tín hiệu theo mong muốn thiết kế. Đây là các phương thức khai báo và biến đối cơ bản và quan trọng trong thiết kế mạch số bằng Verilog, giúp tối ưu hóa xử lý dữ liệu.

```
reg     [7:0]   data;     // Khai báo một thanh ghi    8 bit
wire    [15:0]  address;  // Khai báo một dây tín hiệu 16 bit
```

Trong đoạn thiết kế trên, “data” là một thanh ghi có 8 bit (data[7] là bit cao nhất, data[0] là bit thấp nhất), “address” là một dây tín hiệu có 16 bit, dùng để gán giá địa chỉ (không lưu trữ).

Về bit slicing, đây là cách thức cho phép lấy một phần của một vector (cả wire và reg) để thực hiện xử lý theo mong muốn. Đoạn thiết kế sau mô tả cách thực hiện bit slicing cho các wire với các độ lớn khác nhau.

```
wire [7:0] data = 8'b10101010;  // Vector 8 bit
wire [3:0] lower_bits = data[3:0];  // Lấy 4 bit thấp (4 bit cuối)
wire upper_bit = data[7];  // Lấy bit cao nhất
```

Lúc này, “lower_bits” chứa 4 bit cuối của “data”, tức là “1010” và “upper_bit” chứa giá trị của bit thứ cao nhất (đứng thứ 8 với giá trị index là 7), tức là “1”.

Quá trình ghép nhiều tín hiệu lại với nhau để tạo thành một tín hiệu lớn hơn thường dùng dấu “{ }” để thực hiện. Đoạn thiết kế sau mô tả cách thực hiện ghép nối và biến đổi trên các wire với các độ lớn khác nhau.

```
wire [3:0] a = 4'b1010;
wire [3:0] b = 4'b1100;
wire [7:0] combined = {a, b};  // Ghép a và b thành một vector 8 bit
```

Lúc này, “combined” sẽ có giá trị “10101100”, trong đó “a” chiếm 4 bit cao và “b” chiếm 4 bit thấp.

Khởi tạo module (Instantiate)

Khi thiết kế một hệ thống lớn, chúng ta có thể chia nhỏ thành các module con để dễ dàng quản lý và tái sử dụng. Việc instantiate một module có nghĩa là tạo một thể hiện (instance) của module con trong một module lớn hơn, bao bọc các module con. Ví dụ sau mô tả một module gồm 2 module con cổng AND và OR trong nó.

```
module and_gate (
    input wire A, B,
    output wire Y
);
    assign Y = A & B;
endmodule
```

```
module or_gate (
    input wire A, B,
    output wire Y
);
    assign Y = A | B;
endmodule
```

Như vậy "top_module" khởi tạo và sử dụng hai module ở trên trong thiết kế của mình, từ đó gán các chân tín hiệu vào tương ứng với các input và output của các module con. Có vài cách để khai báo và gán các chân module con, sinh viên tự tìm hiểu thêm về phần này.

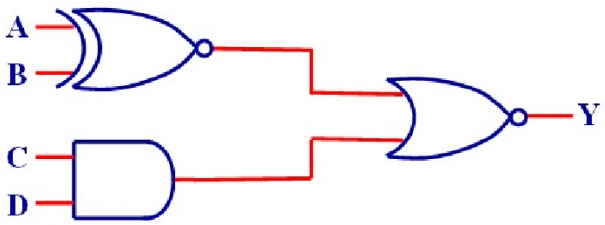
```
module top_module (
    input wire X1, X2,
    output wire Z1, Z2
);

    and_gate AND1   (.A(X1),  .B(X2),  .Y(Z1) ); // Khởi tạo "module" AND gate
    or_gate OR1     (.A(X1),  .B(X2),  .Y(Z2) ); // Khởi tạo "module" OR gate

endmodule
```

Bài tập báo cáo

1. Sử dụng Verilog HDL để **thiết kế và khảo sát** các cổng logic cơ bản gồm **AND, NAND, OR và XOR** theo module gợi ý sau. **Mô tả bảng chân trị của từng cổng logic** và **hình ảnh** triển khai trên FPGA vào báo cáo (ít nhất 3 trường hợp). Module có hai ngõ vào được gán với SW và một ngõ ra gán với LED.

```
module test (A, B, Y);
input A, B;
output Y ;
    assign Y = A <??????> B;
endmodule
```

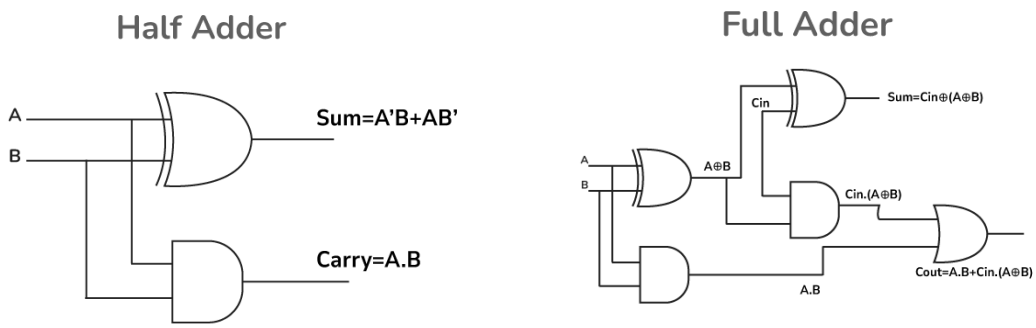
2. Sử dụng Verilog HDL để mô tả mạch sau, cấu hình xuống kit DE10-Standard và điền vào bảng chân trị trong báo cáo. Ngõ vào A, B, C, D gán với các SW và ngõ ra Y gán với một LED. Cần thể hiện ít nhất 5 trường hợp triển khai trên FPGA. Mẫu code gợi ý ở dưới.



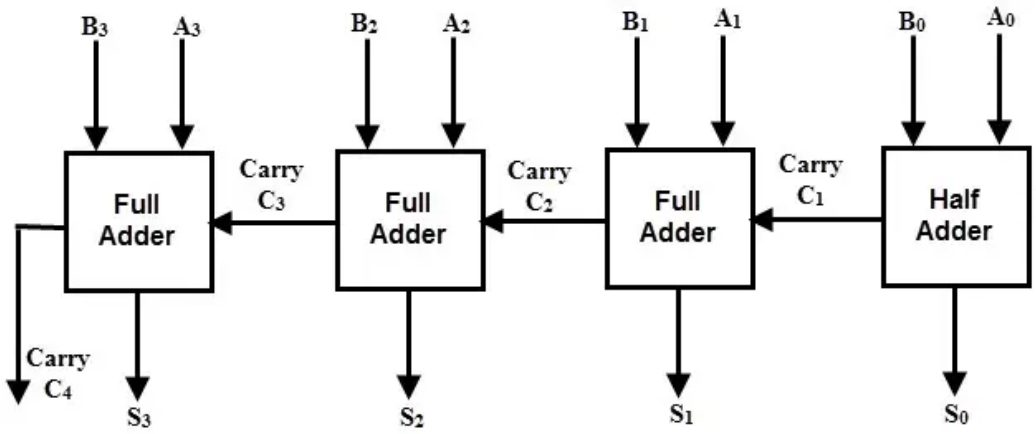
A	B	C	D	Y
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

```
module test (A, B, C, D, Y);
input A, B, C, D ;
output Y ;
    assign Y = <????????>;
endmodule
```

3. Tìm hiểu và thiết kế các mạch cộng gồm **Half Adder** và **Full Adder** cho 2 bit ngõ vào (gán với SW) và các ngõ ra gán với một LED. **Lập bảng chân trị cho từng thiết kế mạch cộng** và hình ảnh khi triển khai thiết kế trên FPGA DE10-Standard (tất cả trường hợp).



4. Từ hai mạch trên, sử dụng cách **khởi tạo module** thiết kế mạch cộng 4-bit nối tiếp như hình dưới. Các input và output được sử dụng gồm số A (4-bit) gán với SW0 → SW3, số B (4-bit) gán với SW4 → SW7, kết quả S (4-bit) gán với LEDR0 → LEDR3, kết quả Carry C4 (1-bit) gán với LEDR4. Báo cáo thiết kế và hình ảnh hoạt động trên FPGA (ít nhất 5 trường hợp).



Chủ đề tự nghiên cứu (không cần báo cáo)

Tìm hiểu **tên kỹ thuật** của thiết kế mạch cộng ở bài tập số 4. Khi mở rộng mạch (mở rộng số bit lên 16-bit, 32-bit hoặc 64-bit) thì nhược điểm của thiết kế này là gì? Ngoài thiết kế này, còn có các thiết kế mạch cộng khác không? Tìm hiểu về các thiết kế đó.

Từ các module bài tập ở trên, tìm hiểu và thiết kế mạch nhân hai số 4-bit. Phân tích các yếu tố tạo nên mạch này.