3.2.2. Mô hình hành vi

Mạch tổ hợp được mô tả bằng mô hình hành vi trong Verilog thông qua khối lệnh always không có xung nhịp

a. Khối lệnh always

- Cú pháp:

  always @(danh_sách_kích_hoạt)
  begin
  ...
  danh_sách_lệnh_thủ_tục
  end
  Trong đó
  danh_sách_kích_hoạt là danh sách các biến/tín hiệu phân cách bởi từ khóa **or** hoặc dấu phẩy (,)
- Hoạt động:
  - Khối always sẽ được thực hiện bất cứ khi nào có sự thay đổi giá trị của các biến trong *danh_sách_kích_hoạt.*
  - Khi khối always được thực hiện, các lệnh trong khối always sẽ được thực hiện tuần tự
  - Các khối always sẽ hoạt động song song/độc lập cùng các khối initial, phép gán liên tục. *Thứ tự thực hiện các khối always, initial, phép gán liên tục là không xác định.*
- Tổng hợp
  - Khối always khi không có xung nhịp được tổng hợp thành mạch tổ hợp hoặc mạch tuần tự sử dụng mạch chốt.
  - Chú ý: Cần hạn chế sử dụng khối always không xung nhịp để mô tả mạch tuần tự.

b. Phép gán blocking (phép gán tuần tự)

- Cú pháp

  biến_LHS = biểu_thức; (biến_LHS = #giá_trị_trễ biểu_thức)
  Trong đó: biến_LHS là biến khai báo kiểu **reg**; biểu_thức là biểu thức Verilog tương tự trong phép gán liên tục
- Hoạt động
  - B1: Tính toán biểu_thức
  - B2: Gán giá trị biểu_thức cho biến_LHS tại thời điểm mô phỏng hiện tại + giá_trị_trễ
  - B3: Kết thúc phép gán tuần tự (thực hiện lệnh tiếp theo)
  - **Chú ý:** Lệnh tiếp theo của phép gán tuần tự chỉ được thực hiện sau khi phép gán tuần tự kết thúc
- Tổng hợp:
  - Thông thường, phép gán blocking sẽ được tổng hợp thành mạch logic tổ hợp
  - Nếu biến_LHS không được gán giá trị trong một số đường thực hiện của khối lệnh always thì phép gán blocking sẽ được tổng hợp thành mạch tuần tự sử dụng mạch chốt
  - Nếu biến_LHS được sử dụng hồi tiếp trong các khối always không có xung nhịp thì phép gán blocking sẽ được tổng hợp thành mạch tuần tự sử dụng mạch chốt.
  - Một số phép toán trong Verilog sẽ không thể tổng hợp thành mạch hoặc tổng hợp thành mạch có kích thước lớn hoặc/và chậm.

a. Câu lệnh điều kiện if/else

- Cú pháp

  if (biểu_thức_điều_kiện)
  lệnh/khối_lệnh;
  else
  lệnh/khối_lệnh;
  Trong đó: khối_lệnh gồm nhiều lệnh thủ tục bao bởi từ khóa **begin**...**end**
- Hoạt động: Nếu biểu_thức_điều_kiện có giá trị khác 0 hoặc x, khối lệnh nhánh if sẽ được thực hiện, nếu biểu thức điều kiện có giá trị 0, x thì khối lệnh nhánh else sẽ được thực hiện. (Thông thường, nên tránh trường hợp biểu_thức_điều_kiện có giá trị x).
- Tổng hợp:
  - Các câu lệnh trong cả nhánh if và else sẽ được tổng hợp và chúng được ghép nối thông qua bộ mux với đầu điều khiển là biểu_thức_điều_kiện
  - Nếu một biến không được gán giá trị trong cả 2 nhánh if, else thì sẽ được tổng hợp thành mạch chốt
  - **Chú ý: Luôn luôn viết đủ nhánh cho câu lệnh điều khiển**
  - Các lệnh if lồng nhau sẽ tạo ra mạch lựa chọn có ưu tiên trong đó điều kiện của lệnh if phía trước sẽ có độ ưu tiên cao hơn
  - Với các lệnh if song song, lệnh if phía sau sẽ có độ ưu tiên cao hơn
  - Khi điều kiện của các nhánh if lồng nhau không loại trừ nhau (overlapped conditions - có thể cùng đúng) thì kết quả mạch có thể khác với specification
  - Chú ý: Kích thước và tốc độ mạch không phụ thuộc đơn giản vào kiểu viết if dạng nested-if, parallel-if, hay điều kiện loại trừ, điều kiện bao hàm.

```
module encoder83(
        input [7:0]     data,
        output [2:0] code
    );
     reg [2:0]  code;

     always @(data)
          begin
                if (data == 8'b0000_0001) code = 0; else
                    if (data == 8'b0000_0010) code = 1; else
                        if (data == 8'b0000_0100) code = 2; else
                            if (data == 8'b0000_1000) code = 3; else
                                if (data == 8'b0001_0000) code = 4; else
                                    if (data == 8'b0010_0000) code = 5; else
                                        if (data == 8'b0100_0000) code = 6; else
                                            if (data == 8'b1000_0000) code =
                                            7; else code = 3'bx;
          end
endmodule
```

a. Câu lệnh lựa chọn case

- Cú pháp

  case/**casex**/**casez** (biểu_thức)

giá_trị_1: khối_lệnh_1
giá_trị_2: khối_lệnh_2
...
default: khối_lệnh_n;
endcase

- Hoạt động
  - Giá trị của biểu_thức sẽ được so sánh với các giá trị lựa chọn giá_trị_1, giá_trị_2, ... tùy từng loại case:
    - case: so sánh sử dụng phép toán === (phân biệt giữa các giá trị 0, 1, x, z);
    - casex: các giá trị x, z, ? trong giá trị lựa chọn sẽ được coi là bằng với bít giá trị 0 và 1. Ví dụ: 0x? có thể coi là bằng 000, 001, 010, 011
    - casez: các giá trị z, ? trong giá trị lựa chọn sẽ được coi là bằng với bit giá trị 0 và 1.
    - Khối lệnh có giá trị lựa chọn bằng giá trị biểu thức sẽ được thực hiện.
    - Nếu không có giá trị lựa chọn bằng giá trị biểu thức, khối lệnh n tương ứng với nhánh default sẽ được thực hiện
    - Nếu có nhiều hơn 1 khối lệnh có giá trị lựa chọn bằng giá trị biểu thức thì khối lệnh đứng trước sẽ được thực hiện (có mức ưu tiên cao hơn)
- Tổng hợp:
  - Các khối lệnh của tất cả các nhánh lựa chọn đều được tổng hợp và được ghép nối qua khối mux
  - casex, casez sẽ được tổng hợp thành mạch có mức ưu tiên khi có 2 nhánh lựa chọn cùng đúng
  - nếu không có default và có biến không được gán giá trị trong mọi nhánh giá trị thì sẽ tổng hợp thành mạch tuần tự sử dụng chốt
- Ví dụ: Bộ tính toán số học và logic cho MIPS32

ALUOp	Funct	ALUControl
00	X	010 (add)
X1	X	110 (subtract)
1X	100000 (add)	010 (add)
1X	100010 (sub)	110 (subtract)
1X	100100 (and)	000 (and)
1X	100101 (or)	001 (or)
1X	101010 (slt)	111 (set less than)

- Chú ý:
  - Tham khảo thêm: Clifford Cummings: "full_case parallel_case, the Evil Twins of Verilog Synthesis"

a. Câu lệnh lặp

- Cú pháp
  - **for** (lệnh_khởi_tạo; điều_kiện;lệnh_chỉ_số) **begin** ... **end**
  - **repeat** (số_lần_lặp) **begin** ... **end**

- **while** (điều_kiện) **begin** ... **end**

- Hoạt động
  - for:
    - Khi bắt đầu vòng lặp, lệnh_khởi_tạo được thực hiện để gán giá trị bắt đầu cho biến chỉ số điều khiển vòng lặp
    - Thực hiện lặp các lệnh:
      - Kiểm tra biểu thức điều_kiện, nếu biểu thức sai thì kết thúc lặp
      - Nếu biểu thức điều_kiện đúng thì thực hiện các lệnh trong vòng lặp bao bởi begin ... end
      - Thực hiện lệnh_chỉ_số để thay đổi giá trị biến chỉ số
  - repeat: Thực hiện lặp số_lần_lặp lần các lệnh bao giữa begin...end
  - while: Thực hiện các lệnh bao giữa begin...end đến khi nào biểu thức điều_kiện sai.
- Tổng hợp
  - Với các lệnh lặp có số lần lặp cố định (vòng lặp tĩnh - static loop), phần mềm tổng hợp sẽ trải vòng lặp thành khối lệnh thủ tục thông thường và thực hiện tổng hợp
  - Với các lệnh lặp có số lần lặp phụ thuộc biến (vòng lặp động non-static loop): phần mềm tổng hợp sẽ thường không tổng hợp thành mạch được. Tuy nhiên có một số trường hợp đặc biệt với phần mềm tổng hợp mức cao (high level synthesis), vòng lặp động có thể tổng hợp được.
- Ví dụ: Mạch majority

```
module majority
    # (parameter data_width = 8,
       cnt_width = 4,
       majority_value = 4
    )
    (
    input [data_width-1:0] data,
    output reg y
    );

    reg [cnt_width-1:0] cnt;
    integer i;

    always @(data)
    begin
        cnt = 0;
        for (i = 0;i<data_width;i=i+1)
        begin
            if (data[i]) cnt = cnt+1;
        end
        if (cnt > majority_value)
            y = 1;
        else
            y = 0;
    end
```

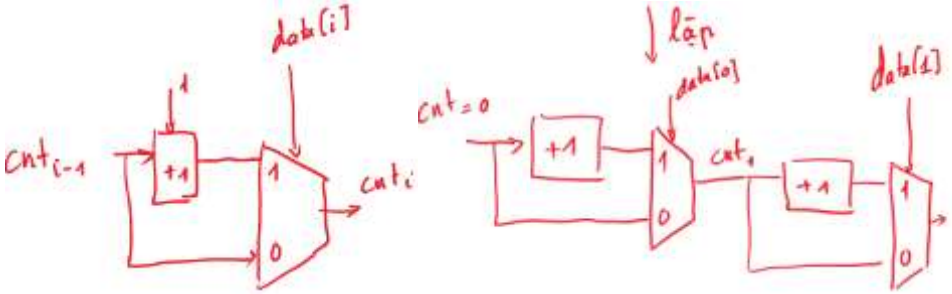
```
endmodule

module test();

    reg [7:0] data;
    wire      y;

    majority_for duv(.data(data), .y(y));

    initial
        begin
            $monitor ("%t: data=%b, y=%b", $time, data, y);
            repeat (20)
                begin
                    data = $random();
                    #5;
                end
        end
endmodule // test
```



a. Chương trình con

- Chương trình con được dùng để đóng gói 1 đoạn mã và tái sử dụng nó
- Chương trình con được dùng để mô tả một khối mạch => chương trình con sẽ có đầu vào, đầu ra, và có thể là mạch tổ hợp hoặc tuần tự.
- Trong Verilog có 2 loại chương trình con là
  - Thủ tục (Task)
    - Được sử dụng như một câu lệnh trong khối always hoặc khối initial
    - Không trả về giá trị, có thể có nhiều đầu ra, nhiều đầu vào
    - Trong thủ tục dùng để mô tả mạch tổ hợp, có thể sử dụng tất cả các câu lệnh như trong khối always trừ các câu lệnh điều khiển thời gian (@, wait)
    - Cú pháp:

```
task tên_thủ_tục;
    khai_báo_đầu_ra;
    khai_báo_đầu_vào;
    khai_báo_biến;

    begin
        ....các lệnh....
```

```
        end
    endtask
```

- Ví dụ:

```
module adder4bit (
    input [3:0] a, b,
    input ci,
    output [3:0] s,
    output co);

    task full_adder;

        input a, b, ci;
        output s, co;

        begin
            {co, s} = a+b+ci;
        end
    endtask

    reg [2:0] c;

    always @(a, b, ci)
    begin
        full_adder (a[0], b[0], ci, s[0], c[0]);
        full_adder (a[1], b[1], c[0], s[1], c[1]);
        full_adder (a[2], b[2], c[1], s[2], c[2]);
        full_adder (a[3], b[3], c[2], s[3], co);
    end
endmodule
```

- Hàm (Function)
  - Được sử dụng như một biến trong các biểu thức
  - Trả về 1 giá trị: có 1 đầu ra, nhiều đầu vào
  - Hàm thì chỉ được sử dụng để mô tả mạch tổ hợp, có thể sử dụng tất cả các câu lệnh như trong khối always trừ các câu lệnh điều khiển thời gian (@, wait)
  - Cú pháp

```
function [msb:lsb] tên_hàm;
    khai_báo_đầu_vào;
    khai_báo_biến;

    begin
        .....các_lệnh....

        tên_hàm = giá_trị_trả_về;
    end
endfunction
```

- Ví dụ

```
module adder4bit (
    input [3:0] a, b,
    input ci,
    output [3:0] s,
    output co);

    function [1:0] full_adder;

        input a, b, ci;

        begin
            full_adder = a+b+ci;
        end
    endfunction

    reg [2:0] c;

    always @(a, b, ci)
    begin
        {c[0], s[0]} = full_adder (a[0], b[0], ci);
        {c[1], s[1]} = full_adder (a[1], b[1], c[0]);
        {c[2], s[2]} = full_adder (a[2], b[2], c[1]);
        {co, s[3]} = full_adder (a[3], b[3], c[2]);
    end
endmodule
```