1/10/2013

Tuesday, October 01, 2013 7:53 AM

3.1.3. Các mô hình mô tả mạch

a) Mô hình cấu trúc:
b) Mô hình dòng dữ liệu

- Mô hình dòng dữ liệu trong Verilog mô tả các biểu thức Bool của hàm truyền đạt của mạch.
- Mô hình dòng dữ liệu sử dụng câu lệnh gán liên tục **assign** để gán biểu thức Bool cho 1 biến wire

- Thông thường, mô hình dòng dữ liệu được dùng để mô tả logic tổ hợp. (Có thể dùng mô hình dòng dữ liệu để mô tả phần tử chốt - latch)
- Hay được dùng để mô tả đường dữ liệu
- Mô hình dòng dữ liệu sẽ được phần mềm tổng hợp chuyển đổi thành mạch tự động
- Mô hình dòng dữ liệu sẽ đơn giản, dễ hiểu hơn mô hình cấu trúc. Tuy nhiên trong các trường hợp cần mạch tốc độ rất cao, mô hình cấu trúc vẫn được sử dụng
- Ví dụ: bộ mux_21_nbits

```
module mux_21_nbits #(parameter n=8) (
        input [n-1:0] a,b,
        input s,
        output [n-1:0] y);

        assign y = (s==0)?a:b;
endmodule
```

Ví dụ: bộ nhân 8 bit

```
module mult(
        input [7:0] a,b,
        output [15:0] y
        )

        assign y = a*b;
endmodule
```

c) Mô hình hành vi

- Mô hình hành vi mô tả hành động tính toán đầu ra của mạch khi đầu vào thay

đổi giá trị.

- Sử dụng khối lệnh always và các câu lệnh thủ tục để xây dựng mô hình hành vi. Câu lệnh thủ tục: if else, case, for, repeat, do...while, phép gán biểu thức
- Mô hình hành vi có thể dùng để mô tả logic tổ hợp (always không có xung nhịp) và logic tuần tự (always có xung nhịp).
- Mô hình hành vi đơn giản gần với ngôn ngữ bậc cao tương tự như ngôn ngữ lập trình.
- Chú ý: Mô hình hành vi sẽ được chuyển đổi (tổng hợp) thành phần cứng nhờ phần mềm tự động; nhưng không phải bất cứ mô hình hành vi nào cũng có thể tổng hợp thành phần cứng. Luôn cần liên kết giữa mô hình hành vi và cấu trúc phần cứng tương ứng.
- Ví dụ: Mô hình hành vi của bộ mux n bit

```
module mux_21_nbits #(parameter n=8) (
    input [n-1:0] a,b,
    input s,
    output [n-1:0] y);

    always @(a or b or s)
    begin
        if (s==0) y = a;
        else y = b;
    end
endmodule
```

3.2. Mô tả mạch logic tổ hợp (Mô hình dòng dữ liệu: Biểu thức Bool và phép gán liên tục; Trễ lan truyền; Các phép toán cơ bản; Mô hình hành vi: Cấu trúc always; Khái niệm về sự kiện; Phép gán blocking) – 3LT

Mạch logic tổ hợp có thể mô tả bằng mô hình cấu trúc, mô hình dòng dữ liệu và mô hình hành vi.

3.2.1. Mô hình dòng dữ liệu

a) Phép gán liên tục

- Cú pháp:

  assign tên_biến = biểu_thức;

  - tên_biến là một biến kiểu **wire** hoặc tên cổng đầu ra của module
  - biểu_thức: biểu thức trên các biến sử dụng các toán tử Boolean hoặc số học

b) Biểu thức Verilog - phép toán

- Phép toán số học

  Nhân (*), Mũ(**), Chia(/), Phần dư(%), Cộng(+), Trừ (-)

Chú ý:
  - Kích thước biểu thức
  - Có thể tổng hợp

- Phép toán Boolean
  - Bit-wise: and (&), or (|), xor (|), ~
    Chú ý: Kích thước biểu thức, mở rộng kích thước toán hạng ngắn
  - Bit-reduction (rút gọn bit): and(&), nand(~&), or (|), nor (~|), xor(^), xnor (~^)
  - Phép dịch: Dịch trái logic (<<), dịch phải logic (>>), dịch trái số học (<<<), dịch phải số học (>>>)
    Chú ý: Kết quả tổng hợp phụ thuộc số bít dịch là hằng số hay không
- Phép toán điều kiện (phép toán logic)
  - Phép toán logic: and (&&), or (||), not (!)
    Chú ý: Nếu một toán hạng chứa x/z => kết quả là x. Nếu không kết quả là 0 (false), 1 (true).
  - Phép so sánh
    lớn hơn (>), lớn hơn bằng (>=), nhỏ hơn (<), nhỏ hơn bằng (<=)
    bằng (==), khác (!=)
    bằng 4 giá trị (===), khác 4 giá trị (!==)
    Chú ý: Phép so sánh sẽ cho kết quả x nếu toán hạng chứa x
    Phép so sánh 4 giá trị sẽ so sánh sự giống nhau (khác nhau) của từng bit trong toán hạng kết cả x và z.

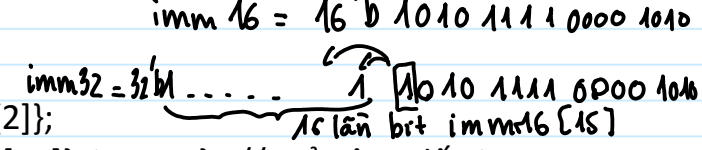
Ví dụ: Phép toán điều kiện

```
module relational_operators();

  initial begin
                $display (" 5    <=  10 = %b", (5    <= 10));
                $display (" 5    >=  10 = %b", (5    >= 10));
                $display (" 1'bx  <=  10 = %b", (1'bx  <= 10));
                $display (" 1'bz  <=  10 = %b", (1'bz  <= 10));

                // Case Equality
                $display (" 4'bx001 ===  4'bx001 = %b", (4'bx001 ===  4'bx001));
                $display (" 4'bx0x1 ===  4'bx001 = %b", (4'bx0x1 ===  4'bx001));
                $display (" 4'bx001 ==  4'bx001 = %b", (4'bx001 ==  4'bx001));
                $display (" 4'bz001 ==  4'bz001 = %b", (4'bz001 ==  4'bz001));
                $display (" 4'bz0x1 ===  4'bz0x1 = %b", (4'bz0x1 ===  4'bz0x1));
                $display (" 4'bz0x1 ===  4'bz001 = %b", (4'bz0x1 ===  4'bz001));
                // Case Inequality
                $display (" 4'bx0x1 !==  4'bx001 = %b", (4'bx0x1  !==  4'bx001));
                $display (" 4'bz0x1 !==  4'bz001 = %b", (4'bz0x1  !==  4'bz001));
                // Logical Equality
                $display (" 5      ==   10     = %b", (5      ==   10));
                $display (" 5      ==   5      = %b", (5      ==   5));
```

```
                    // Logical Inequality
                    $display (" 5     !=  5     = %b", (5     !=  5));
                    $display (" 5     !=  6     = %b", (5     !=  6));

                    if (4'bx001 ==  4'bx001)  // khi so sánh thông thường, kết quả biểu thức điều
                    kiện là x tương đương với false => nhánh else được thực hiện
                        $display ("4'bx001 == 4'bx001");
                    else
                        $display ("4'bx001 != 4'bx001");

                    if (4'bx001 ===  4'bx001) / khi so sánh 4 giá trị, kết quả biểu thức điều kiện là
                    true  => nhánh if được thực hiện
                        $display ("4'bx001 === 4'bx001");
                    else
                        $display ("4'bx001 !== 4'bx001");
        end
endmodule

run
#  5    <=  10 = 1
#  5    >=  10 = 0
#  1'bx  <=  10 = x
#  1'bz  <=  10 = x
#  4'bx001 ===  4'bx001 = 1
#  4'bx0x1 ===  4'bx001 = 0
#  4'bx001 ==  4'bx001 = x
#  4'bz001 ==  4'bz001 = x
#  4'bz0x1 ===  4'bz0x1 = 1
#  4'bz0x1 ===  4'bz001 = 0
#  4'bx0x1 !==  4'bx001 = 1
#  4'bz0x1 !==  4'bz001 = 1
#  5      ==   10     = 0
#  5      ==   5      = 1
#  5      !=   5      = 0
#  5      !=   6      = 1
# 4'bx001 != 4'bx001
# 4'bx001 === 4'bx001
```

- Phép toán trên bit vector
  - Slice: Sử dụng để lấy ra bit vector con của 1 toán hạng bit vector.
    Cú pháp: biến[index1:index2];
  - Nối bit vector: Sử dụng để nối các bit vector
    Cú pháp: {biến_1, biến_2, ...};
  - Sao chép bit vector: Sử dụng để ghép n bản sao của một bit vector
    Cú pháp: n{biến}

    Ví dụ:
    assign out = {a[1:0],b,c,d,a[2]};
    assign imm32 = {16{imm16[15]}, imm16}; // mở rộng dấu imm16



thành số 32 bit

- Phép toán điều kiện
  Cú pháp (biểu_thức_điều_kiện)?biểu_thức_1:biểu_thức_2;

  assign y = (s==0)?a:b;

3.2.2. Mô hình hành vi

a) Khối lệnh always không có xung nhịp

- 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**
- 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*
  - Các khối always sẽ hoạt động song song 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 = #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 sau thời gian 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
- 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.

c) Câu lệnh điều kiện if/else

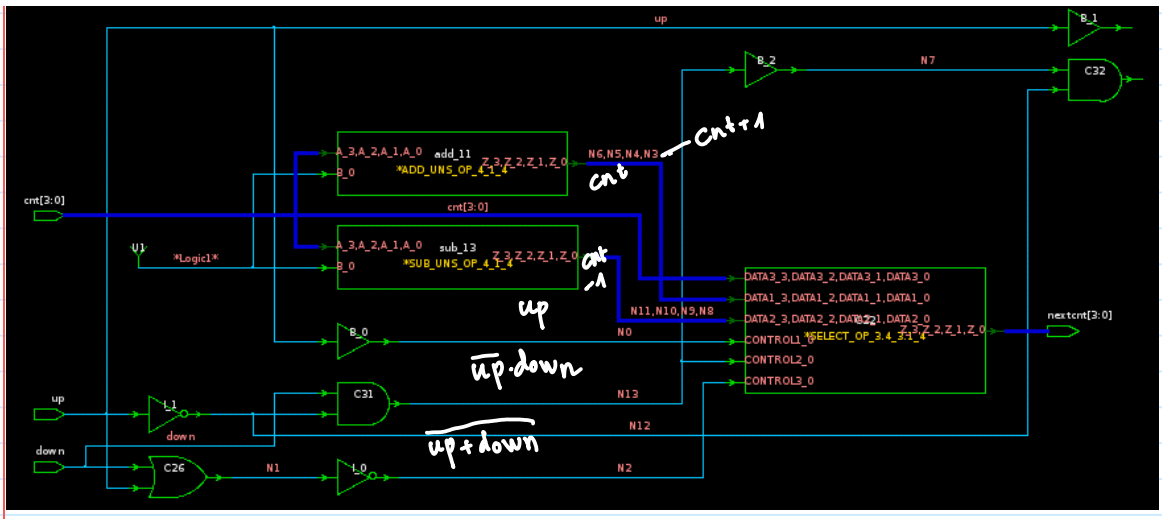
- Cú pháp

  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
- Tổng hợp:
  - Các câu lệnh trong cả nhánh if và else sẽ được tổng hợp và chúng được ghép nối thông qua bộ mux
  - 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 (bộ mux tương ứng sẽ đứng gần đầu ra 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
- Ví dụ: câu lệnh điều kiện if/else
  Mạch encoder 8-3

```
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
```

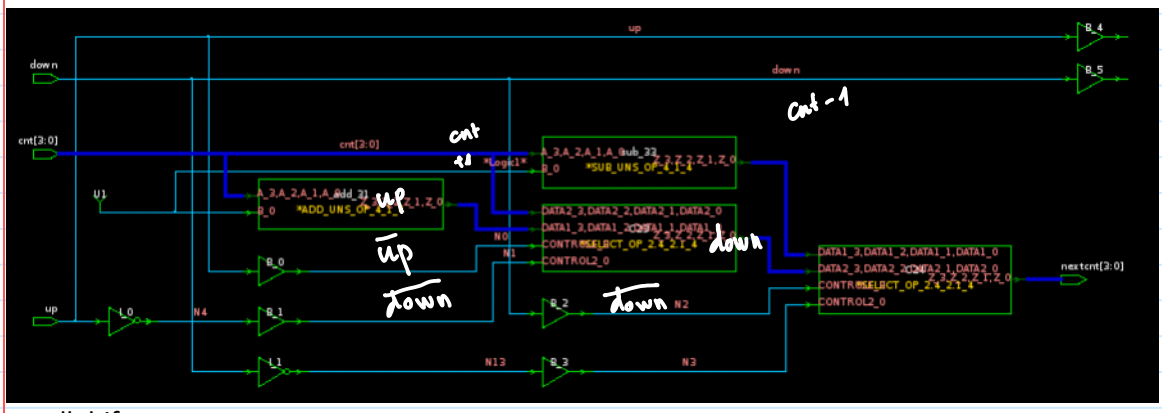
```
                    if (data == 8'b0000_1000) code = 3; else
                        if (data == 8'b0001_0000) code = 4; else
                            if (data == 8'b0010_0000) code = 5; else
                                if (data == 8'b0100_0000) code =
                                6; else
                                    if (data == 8'b1000_0000)
                                    code = 7; else code = 3'bx;
        end
endmodule

module encoder83_latch(
                    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;
        end
endmodule

module encoder83_priority(
                    input [7:0]     data,
                    output [2:0] code
                    );
    reg [2:0]                                                   code;

    always @(data)
        begin
            if (data[0]) code = 0; else
                if (data[1]) code = 1; else
                    if (data[2]) code = 2; else
                        if (data[3]) code = 3; else
                            if (data[4]) code = 4; else
                                if (data[5]) code = 5; else
                                    if (data[6]) code = 6; else
                                        if (data[7]) code = 7; else
                                        code = 3'bx;
        end
endmodule
```



v

Ví dụ: updown

nested-if:



parallel-if

d) 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); 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
    - casex: các giá trị x, z, ? trong giá trị lựa chọn sẽ được coi là bằng với bít giá trị 0 và 1. 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)
    - casez: các giá trị z, ? trong giá trị lựa chọn sẽ được coi là bằng với bit giá trị 0 và 1. 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
    - 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 giải mã
    - Bộ tính toán số học logic ALU

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)

- Tham khảo thêm: Clifford Cummings: "full_case parallel_case, the Evil Twins of Verilog Synthesis"
- Lời khuyên: chỉ sử dụng case (hạn chế dùng casex và casez), các giá trị lựa chọn trong các nhánh loại trừ nhau.
- Bài tập: Thiết kế lại bộ encoder83, encoder83_priority sử dụng case

e) Lệnh lặp tĩnh (Số lần lặp cố định không phụ thuộc biến số)

- Cú pháp
- Hoạt động

- Tổng hợp
-