CHƯƠNG 3: Mô tả IC số, hệ thống số dùng ngôn ngữ Verilog (12LT+3BT)

3.1. Các thành phần cơ bản của ngôn ngữ Verilog (Nhắc lại chương 1: Khái niệm module, port, chú thích; Khai báo tín hiệu, biến; hằng trong mạch; Các mô hình mô tả mạch: mô hình cấu trúc, mô hình dòng dữ liệu, mô hình hành vi) – 1LT

3.1.1. Khái niệm module và port

a. Module

- Thiết kế mạch bằng Verilog được khai báo bằng cú pháp module
- Cú pháp

```
module tên_module (
        khai_báo_cổng_vào_ra
);
        /* Khai báo biến, tham số sử dụng trong module; */
        /* Mô tả cấu trúc, hoạt động của module bằng các cú pháp Verilog;*/

endmodule
```

b. Cổng vào ra

- Cổng vào ra của thiết kế được khai báo cùng với module
- Cú pháp

  input/output/inout [khoảng_chỉ_số] tên_cổng,

  Trong đó:
  - **input**, **output**, **inout** được sử dụng để chỉ ra hướng tín hiệu (vào, ra, hoặc vào/ra) của cổng
  - [khoảng_chỉ_số] là tùy chọn dùng để khai báo một cổng là bit vector (gồm nhiều tín hiệu). khoảng_chỉ_số có cú pháp [chỉ_số_lớn:chỉ_số_nhỏ]
  - các cổng trong khai báo được phân cách nhau bởi dấu ,; cổng cuối cùng trong khai báo kết thúc bằng dấu) của khai báo module

- Chú ý: Có thể sử dụng cú pháp như sau để khai báo module và cổng
  Trong khai báo module chỉ liệt kê tên các cổng. Hướng và kích thước của cổng được chỉ rõ trong thân module.

```
module tên_module (danh_sách_cổng_vào_ra);

        /* Khai báo tham số sử dụng trong module */

        /* Khai báo cổng vào ra */
        input/output/inout [khoảng_chỉ_số] tên_port;
        ....
        /* Khai báo biến, hằng sử dụng trong module */
        /* Mô tả module */
```

endmodule

3.1.2. Tham số, hằng số, biến, tín hiệu

a. Tham số

- Tham số là một hằng số trong module được khởi tạo giá trị khi tạo ra/sử dụng module trong một thiết kế khác. Tham số được dùng để tái cấu hình lại module.
  Ví dụ: Bộ cộng n bit

- Cú pháp: Khai báo cùng module và cổng vào ra
  module tên_module #(**parameter** tên_tham_số = giá_trị_mặc_định,
  tên_tham_số2=giá_trị_mặc_định,...)
  (khai_báo_cổng_vào_ra);

- Cú pháp bên trong thân module, trước khai báo cổng
  parameter tên_tham_số = giá_trị_mặc_định;

Ví dụ: Khai báo module và tham số

```
module mux_21
    #(parameter n=8)
    (
        input [n-1:0] a,b,
        input s,
        output [n-1:0] y
    );

    ...
endmodule

module mux_21 (a, b, s, y);

    parameter n = 8;
    input [n-1:0] a,b;
    input s;
    output y;

    ...
endmodule
```

b. Hằng số

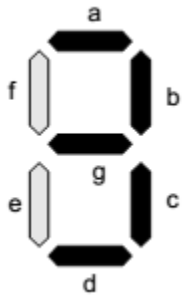
- Cú pháp
  - **localparam** tên_hằng = giá_trị;
  - `**define** tên_hằng = giá_trị;
- **localparam** thường được dùng để khai báo tên và giá trị mã hóa trạng thái của FSM
- **define** thường được dùng để khai báo các giá trị mã hóa đầu vào hoặc đầu ra

Ví dụ: Khai báo hằng số

- Khai báo 3 trạng thái của mạch IDLE, SEND, RECV:

  localparam IDLE = 2'b00;
  localparam SEND = 2'b01;
  localparam RECV = 2'b10;

- Khai báo hằng số giá trị đầu ra điều khiển LED 7 thanh



  `define BLANK = 7'b1111111;
  `define ZERO = 7'b0000001;
  `define ONE = 7'b1001111;
  `define TWO= 7'b0010010;

Biến và tín hiệu

- Cú pháp

  kiểu_biến [khoảng_chỉ_số] tên_biến;

  Trong đó:
  - kiểu_biến: có thể là **wire**, **reg**, **real**, **integer**, **time, wor, wand, tri, trireg, trior, triand, supply0, supply1**
    khoảng_chỉ_số: là khai báo tùy chọn để chỉ ra kích thước theo bit của biến (không áp dụng cho các kiểu integer, time, real)
- Các kiểu biến
  - Các kiểu biến net (**wire, wor, wand, tri, trior, triand, supply0, supply1**) được dùng để lan truyền giá trị, mô tả các dây dẫn kết nối thành phần mạch
  - Các kiểu biến reg (**reg, trireg, real, integer, time**) được dùng để lưu trữ giá trị. Trong đó **reg** thì được dùng để mô tả các cổng logic hoặc flip-flop, latch trong mạch. Các kiểu **real** (64 bit), **integer** (32 bit), **time** (64 bit) được dùng trong testbench khi mô phỏng-không có thành phần tương ứng trong phần cứng.
  - Các thành phần trong thiết kế phần cứng số đồng bộ thường được mô tả thông qua 2 loại biến chính là **wire** và **reg**
  - Phân biệt **wire và reg**

wire	**reg**

dùng để mô tả dây dẫn	dùng để mô tả cổng logic, flip-flop, latch
Không lưu trữ giá trị	lưu trữ giá trị cho đến khi được gán giá trị mới
Cần được nối với cổng, Nhận giá trị của cổng điều khiển nó	Không nhất thiết là thanh ghi hay flip flop
Kết nối đầu ra và đầu vào khi thực thể hóa các module	Có thể kết nối với 1 cổng vào của 1 module con Không thể kết nối với 1 cổng ra của 1 module con
Có thể khai báo tín hiệu input, output của module là wire	Có thể khai báo tín hiệu output là reg Không thể khai báo tín hiệu input là reg
Là kiểu duy nhất có thể nằm bên trái phép gán dùng assign	Không thể nằm bên trái phép gán assign
Không thể nằm bên trái phép gán = và <= trong khối always@	Là kiểu duy nhất nằm bên trái phép gán trong khối always và khối initial
Chỉ có thể dùng để mô tả logic tổ hợp	Dùng để mô tả logic tổ hợp và logic tuần tự

Ví dụ phân biệt wire và reg

module mux_21 (a, b, s, y);

module test_mux;
// Q: a_sim, b_sim, s_sim, y_sim có thể khai báo là kiểu gì?
// A: a_sim, b_sim, s_sim có thể là kiểu **wire**, **reg**
// y_sim phải là **wire**
// nếu a_sim, b_sim, s_sim được gán nhiều giá trị trong initial/always => phải là kiểu reg

```
mux_21 dut(.a(a_sim),
           .b(b_sim),
           .s(s_sim),
           .y(y_sim)
           );
```

endmodule

- Giá trị biến
  - Với các biến 1 bit: Nhận logic 4 trạng thái:1'b0, 1'b1, 1'bX, 1'bZ
  - Với các biến bit vector: Nhận chuỗi bit, mỗi bit có giá trị là logic 4 trạng thái
    - Cú pháp: <số_bit>'<cơ_số><chuỗi_chữ_số_giá_trị>
      Trong đo: <số_bít> xác định kích thước của giá trị

<cơ_số> nhận giá trị: b, d, o, h tương ứng với cơ số 2, 10, 8, 16
    - Ví dụ: 2'b01, 2'b0X, 16'hF0A5, -16'd10
- Mảng
    - Cú pháp: kiểu [khoảng_chỉ_số_1] tên_mảng [khoảng_chỉ_số_2];
    - Trong đó: kiểu là wire, reg, integer, ...
    - khoảng_chỉ_số_1 chỉ ra kích thước mỗi phần tử trong mảng
    - khoảng_chỉ_số_2 chỉ ra kích thước mảng