Verilog HDL

Lecture 02

Verilog 문법

- C문법과 매우 유사
  - But! 회로를 그린다는 느낌으로 코드 작성 (병렬)
- 대소문자 구별
- Keyword는 반드시 소문자
- 문장의 마지막은 세미콜론(;)으로 끝남
  - end~ 와 같은 것들은 예외
- 주석은 /* 로 시작해서 */ 로 끝난다
- 한 줄 주석은 // 을 문장 앞에 서술합니다

```
//이 문장은 동작하지 않습니다.

/*
  이곳은 주석 처리된 영역입니다.
*/
```

수 표현

- 형식 : <비트수> <기본형식> <숫자>
  - <비트수>
    - 10진수로 표기
  - <기본형식>
    - ‘b : 2진수
    - ‘o : 8진수
    - ‘d : 10진수
    - ‘h : 16진수
  - <숫자>
    - 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F를 사용
    - <기본형식> 보다 낮은 숫자를 사용
    - Ex) ‘o 일 때는 0~7 사이 숫자만 사용
- x : Unknown
- z : High Impedance

수 표현

구 분	표현방식	의 미
일반적인 표 현	4'b1111 12'hABC 16'd255	4-bit 2진수 12-bit 16진수 16-bit 10진수
<크기>가 지정되지 않은 경우	23456 'hC3 'o21	32-bit 10진수 32-bit 16진수 32-bit 8진수
음수 표현	-6'd3	6-bit 3의 2의 보수
x, z 의 표 현	12'h13x 6'hx 32'bz	12-bit 16진수, 마지막 4bits는 Unknown 6-bit 16진수 32-bit 2진수, High-Impedance

Tip! 8'b11001010 == 8'b1100_1010

논리 값 수준

논리값 표기	회로에서의 상태
0	논리적 0, 거짓 상태, GND
1	논리적 1, 참 상태, VDD
x	Unknown
z	High-Impedance

Data Type

- Nets
  - 장치의 출력으로 유도되는 연속적인 값
  - wire 라는 키워드로 정의

```
wire a;                 // Net a를 정의
wire b, c;// Net b, c를 정의
wire d = 1'b0;          // Net d는 0에 연결
```

- Registers
  - Data를 저장
  - 다른 값이 들어오기 전까지 현재 값 유지
  - reg 라는 키워드로 정의

```
reg[1:0] state;       // Register 2bits state를 정의
always begin          // always block 시작
    state = 2'b00;    // state에 2bits 0을 대입
End                   // always block 종료
```

Vector & Array

- Vector
  - Nets과 Registers Data Type의 2개 이상의 Bits

```
wire[31:0] Bus;                 // 32bits Net
reg[15:0] Count;                // 16Bits Register
```

- Array
  - C언어의 Array와 유사
  - 다차원 배열도 가능

```
reg[7:0] buf[0:1023];           // 8bits buf가 1024개, 총 1 KBytes
```

Module

- Module 정의는 module로 시작하여 endmodule로 끝

```
module CA_AND2(In1,In2,Out);
    input In1, In2;
    output Out;

    wire Out;

    assign Out = In1 & In2;
endmodule
```

- Port
  - input / output / inout
  - Inout은 사용시, 주의할것! High-Impedance 사용

Initial

- 보통 Test Bench를 만들 때 사용하고,
  각 initial block 들은 병렬적으로 동작함.
- Simulation 동안 한 번만 실행하고, 다시는 실행하지 않음.
- Example

```
initial
  A = 1'b0;
  #50 A = 1'b1;
```

- 2줄 이상일 경우, begin과 end로 내용을 묶어줌.

Always

- 보통 Module을 만들 때 사용하고,
  각 always block 들은 병렬적으로 동작함.
- Module이 종료될 때 까지, 계속 실행됨.
- always @(Sensitivity List)
  - Sensitivity List 의 신호가 바뀔 때, always block 동작.
- Example
  - always @(posedge clk)
  - always @(State)
- 2줄 이상일 경우, begin과 end로 내용을 묶어줌.
- Tip!
  - always @(*) 라고 할 경우, 모든 신호를 자동으로 고려함.

Blocking

- = 를 사용하여 값을 대입.
- Block 내부에 열거된 순서대로 실행.
- 아래 문장을 모두 실행한 최종 결과는?

A is 1.
B is 2.
C is 3.

A is 0.
B is 0.
C is 0.

Verilog 문법 X, 개념을 알기 위한 코드

- 열거된 순서대로 실행하지만, A,B,C의 변화는 동시에 발생.
- C언어와 유사함.

Non-Blocking

- <= 를 사용하여 값을 대입.
- Block 내부에 있는 문장을 한번에 실행.
- 아래 문장을 모두 실행한 최종 결과는?

A is 1.
B is 2.
C is 3.

A is 0.
B is 1.
C is 2.

Verilog 문법 X, 개념을 알기 위한 코드

- 실행되는 시점의 값을 기준으로 동작함.
- Shift Register를 연상시킴.

Example

```
always @(posedge clk or negedge rst)
begin
    if(!rst)
        count <= 4'b0000;
    else
        count <= count + 1;
end
```

FlipFlop을 **이용한** Counter**를 구현한 회로**

```
always @(State)
begin
    case(State)
        4'b0100: Out = 4'b1101;
        4'b1010: Out = 4'b1001;
        4'b0110: Out = 4'b1101;
        4'b1111: Out = 4'b0001;
        default: Out = 4'b1111;
    endcase
end
```

임의의 규칙의 Combinational Circuit