2주차 결과보고서

전공: 신문방송학과 학년: 4학년 학번: 20191150 이름: 전현길

1. 연속 할당문, 절차형 할당문의 차이를 비교하여 설명하시오.

Verilog에서 값을 할당하는 구문은 연속 할당문(continuous assignment)과 절차형 할당문(procedural assignment)으로 두 가지가 존재한다. 할당문은 각각 net 자료형, variable 자료형과 연관지어 생각할 수 있다.

연속 할당문은 net 자료형 객체에 값을 할당하는 구문으로 **assign문**을 사용한다. 할당 구문은 다음과 같고, delay는 생략할 수 있다.

```
assign [delay] net_assignments;
```

연속 할당문의 경우 **우변에 값의 변화가 발생했을 때, 좌변의 객체에 값이 할당된다.** 할당된 값은 언제든지 변경될 수 있으며, 값이 변화할 때마다 값을 즉시 반영한다. 따라서 주로 조합 논리(combinational logic) 회로를 구현할 때 사용한다.

절차형 할당문은 variable 자료형(reg, integer, real, realtime 등) 객체에 값을 할당하는 구문으로, **always문, initial문, task/function 내부의 할당문**이 해당된다. assign문과 달리 코드의 순서대로 실행되어 할당문 좌변의 변수 값을 갱신하며, **문장이 실행되어야 좌변에 값이 할당된다.**

절차형 할당문은 다시 blocking 문법과 nonblocking 문법의 할당문으로 나뉜다. **blocking 할당문은 = 연산자로 값을 할당**하고, **non-blocking 할당문은 <= 연산자로 값을 할당**한다. blocking 문법은 현재 할당문의 실행이 완료된 후(blocking) 다음의 할당문을 실행하며, non-blocking 문법은 이와 반대로 정해진 할당 스케줄에 따라 코드의 순서와 관계없이 값을 할당한다.

2. Blocking 및 nonBlocking 문법의 차이를 simulation을 통해 설명하시오.

blocking 할당문과 non-blocking 할당문의 차이를 설명하는 예시 코드를 Vivado에서 다시 작성하고 시뮬레이션을 돌려 보았다, 다음과 같은 코드이다.

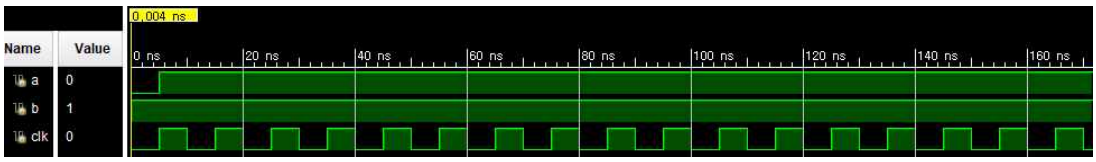
Non-Blocking 할당문	Blocking 할당문

`timescale 1ns / 1ps
 	
module inv;
reg a, b, clk;
 	
initial begin
  a = 0;
  b = 1;
  clk = 0;
end
 	
always clk = #5 ~clk;
always @(posedge clk) begin
  a <= b; // a = 0, 1, 0, ...
  b <= a; // b = 1, 0, 1, ...
end
endmodule
	module blk1;
	  wire out;
	  reg a, b, clk;
	
	  initial begin
	  a = 0;
	  b = 1;
	  clk = 0;
	  end
	
	  always clk = #5 ~clk;
	
	  always @(posedge clk) begin
	  a = b; // a = 1
	  b = a; // b = a = 1
	  end
	endmodule

코드의 주석에 작성된 것과 같이, Blocking 할당문의 경우 순차적으로 값을 실행하므로 a, b 값이 1로 유지되고 Non-Blocking 할당문의 경우 동시에 clk에 따라 할당을 실행하므로 값을 계속해서 바꾸게 된다. 실제로 그런지 위의 코드를 Vivado simulation을 통해 검증해 보았다.

Non-Blocking 할당문

Blocking 할당문



처음에는 양쪽 모두 원래의 값인 a = 0, b = 1을 출력하다가, 클럭의 상승 엣지(posedge clk)일 때 값이 변화하는 것을 확인할 수 있다. Non-Blocking 할당문에선 값이 교차해서 반복되며, Blocking 할당문의 경우 값이 순차적으로 할당되기 때문에 한 번 a가 1로 바뀐 뒤 그 값이 유지되는 것 또한 확인할 수 있다.

3. Verilog의 for문, if문, while문, case문을 C언어와 비교하여 설명하시오.

for문은 c언어에서의 형식과 동작 방식이 유사하다. for (초기화문; 조건문; 증감문) 형태에 begin ~ end를 추가로 붙여 작성한다는 점, for문 안의 변수는 integer로 미리 선언해야 한다는 점을 유의하면 된다.

if문은 c언어에서의 조건문과 형식이 거의 유사하지만, 동작 오류가 발생할 가능성을 줄이기 위해 if문에 대해 else문을 항상 포함하는 것이 권장된다. 조건이 여러 개 있을 경우 c언어와 유사하게 if ~ else if ~ else문을 사용한다.

while문 역시 c언어와 동작 방식이 유사하다. while (조건문) 형식에 for문처럼 begin ~ end문을 추가로 붙여 작성한다. 조건문이 만족되는 동안 반복문을 실행하고, 조건문이 만족되지 않으면 반복문을 탈출한다.

case문은 c언어의 switch문에 해당되며, 주로 디코더 등 특정한 값에 따라 다양한 결과값이 나올 수 있는 상황에 사용하면 유용하다. don't care를 포함하도록 구성할 수도 있는데, 이럴 경우에는 casex문, casez문 등을 사용하면 된다. 예시 코드는 https://lifelectronics.tistory.com/153 사이트를 참조했다.

case문	casex문

reg [15:0] rega;
reg [9:0] result;
 	
always@(rega)
begin
case(rega)
16'd0 : result = 10'b0111111111;
16'd1 : result = 10'b1011111111;
16'd2 : result = 10'b1101111111;
16'd3 : result = 10'b1110111111;
16'd4 : result = 10'b1111011111;
16'd5 : result = 10'b1111101111;
16'd6 : result = 10'b1111110111;
16'd7 : result = 10'b1111111011;
16'd8 : result = 10'b1111111101;
16'd9 : result = 10'b1111111110;
default : result = 10'bx;
endcase
end
	module pri_enc_casex(encode, enc);
	input [3:0] encode;
	output [1:0] enc;
	reg [1:0] enc;
	
	always@(encode)
	begin
	casex(1) //casex문 선언
	encode[3] : enc = 2'b11;
	encode[2] : enc = 2'b10;
	encode[1] : enc = 2'b01;
	encode[0] : enc = 2'b00;
	default : $display("Error")
	endcase
	end
	endmodule

4. Verilog의 net형 자료형에 대해서 조사하시오.

Verilog에는 크게 두 가지 자료형(data type)이 존재하는데, 추상적 저장 장치(=값을 저장)인 **register**와 디바이스의 물리적 연결을 나타내는 **net**이다.

net 자료형은 스스로 특정한 신호를 저장하지 않으며, 디바이스 간의 입출력 연결만을 의미한다. 때문에 주로 연속 할당문 등을 이용해 net 값을 유지하게 되며, 유도되는 값이 존재하지 않는다면 Z값을 기본적으로 가진다(high impedence). net 자료형을 선언할 때 초기값, 지연, 비트 폭(크기)을 정의할 수 있으며, 비트 폭을 정의하지 않을 경우 기본적으로 1bit net으로 선언된다.

net 자료형은 논리적 동작·기능이 있는 자료형과 없는 자료형으로 분류할 수 있다. 논리적 동작·기능이 없는 자료형으로 **wire, tri** 등이 있고, 논리적 동작·기능이 있는 자료형의 경우 **wand, wor, for, triand, trior** 등이 있다.

wire는 하드웨어 요소 간의 단순한 물리적 연결을 나타내기 위해 사용되며, tri는 wire처럼 단순한 물리적 연결을 나타내기 위해 사용되지만 추가적으로 3상 net에 사용된다. wand, wor은 다중 입력 and 게이트, or 게이트를 사용한 것과 유사하게 동작한다.