2018년 2학기 컴퓨터공학실험Ⅱ
CSE3016-05반 2주차 결과 보고서

학번: 20171665
이름: 이 선 호

2018. 09. 14

목 차

Ⅰ 할당문

1. 연속 할당문과 절차형 할당문의 차이

wire 또는 net의 값을 assign 키워드로 업데이트하는 문장을 연속 할당문 또는 진행형 할당문(continuous assignment)이라고 한다. 예시를 들어 설명하면 아래와 같다.

```
wire x, y;
assign y = ~x;
assign x = 1'b1;
```

[예제 1] 연속 할당문을 사용한 코드

assign 키워드의 의미는 좌변에 있는 net을 우변에 있는 수식으로 구동하라는 뜻이다. 우변에 올 수 있는 수식은 계산식, 함수, reg 형식의 신호 등이 있다. assign의 의미에 유의하여 예제 1에서 두 번째 문장을 해석하면 'y라는 net을 구동하는 신호는 x를 부정(반전)한 값이다'라고 말할 수 있다. 마찬가지로 세 번째 문장도 'x라는 net을 구동하는 신호는 1이다'라는 의미임을 알 수 있다. 참고로 예제 1에서 첫 번째와 두 번째 문장을 간단하게 'wire y = ~x;처럼 wire 형식의 net의 선언과 assign 키워드를 동시에 쓸 수도 있다.

논리회로의 소자를 연결하는 선의 개념인 wire 또는 net에 신호를 구동하는 데 assign 키워드를 사용한다. 또한, 우변의 값에 변화가 발생했을 때 좌변의 객체에 값의 할당이 발생하기 때문에 어떤 값을 저장하거나 순서가 중요한 경우에는 연속 할당문을 사용할 수 없다. 그리고 연속 할당문은 initial, always, function, task 등의 절차 처리기(procedure) 밖에서 사용한다.

절차형 할당문(procedural assignment)은 좌변의 변수의 데이터 형식이 reg, integer, real, time 등 일 때 우변의 값을 저장하거나 갱신한다. 또한 다음 절차형 할당문에 의해 값이 갱신될 때까지 그 변수에 할당된 값을 유지한다. 그래서 좌변의 객체에 저장되는 값은 우변의 값의 변화에 무관하기 때문에 할당문의 순서가 결과에 영향을 미칠 수 있다. 그리고 절차형 할당문은 절차 처리기 안에서 사용이 가능하다.

위의 내용을 바탕으로 연속 할당문과 절차형 할당문의 차이를 정리하면 아래와 같다.

연속 할당문	절차형 할당문
좌변의 객체가 wire 형식의 net이어야 한다.	좌변의 객체로 reg, integer, real, time 등이 올 수 있다.
우변의 값이 변화할 때 좌변의 객체의 값에 할당된다.	우변의 값의 변화와 상관 없다.
값을 저장할 수 없다.	값을 저장할 때 사용 가능하다.
할당문의 순서와 무관하다.	할당문의 순서가 결과에 영향을 미친다.
procedure 밖에서 사용한다.	procedure 안에서 사용한다.

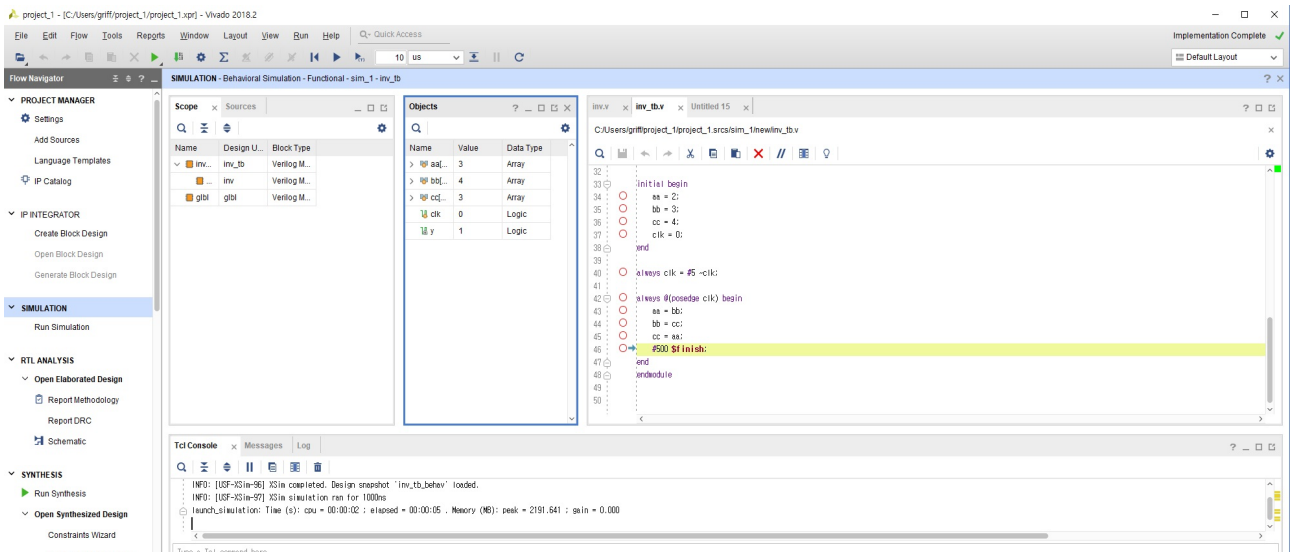
[표 1] 연속 할당문과 절차형 할당문의 차이

2. Blocking과 Nonblocking의 차이

Blocking은 할당 연산자 '='를 말하고, Nonblocking은 할당 연산자 '<='를 의미한다. Blocking은 클럭(clock)이 발생할 때 순차적으로 대입되며 위의 할당이 아래의 할당에 순차적으로 영향을 준다. 반면에 Nonblocking은 대입문의 처리 순서가 없으며 동시에 실행된다. 아래의 예시를 가지고 실제 Vivado 프로그램에서 simulation하여 Blocking과 Nonblocking에 따라 결과 값이 어떻게 나오는지 확인했다.

```
initial begin
    aa = 2;
    bb = 3;
    cc = 4;
    clk = 0;
end
always clk = #5 ~clk;
always @(posedge clk) begin
    aa = bb;
    bb = cc;
    cc = aa;
    #500 $finish;
end
```

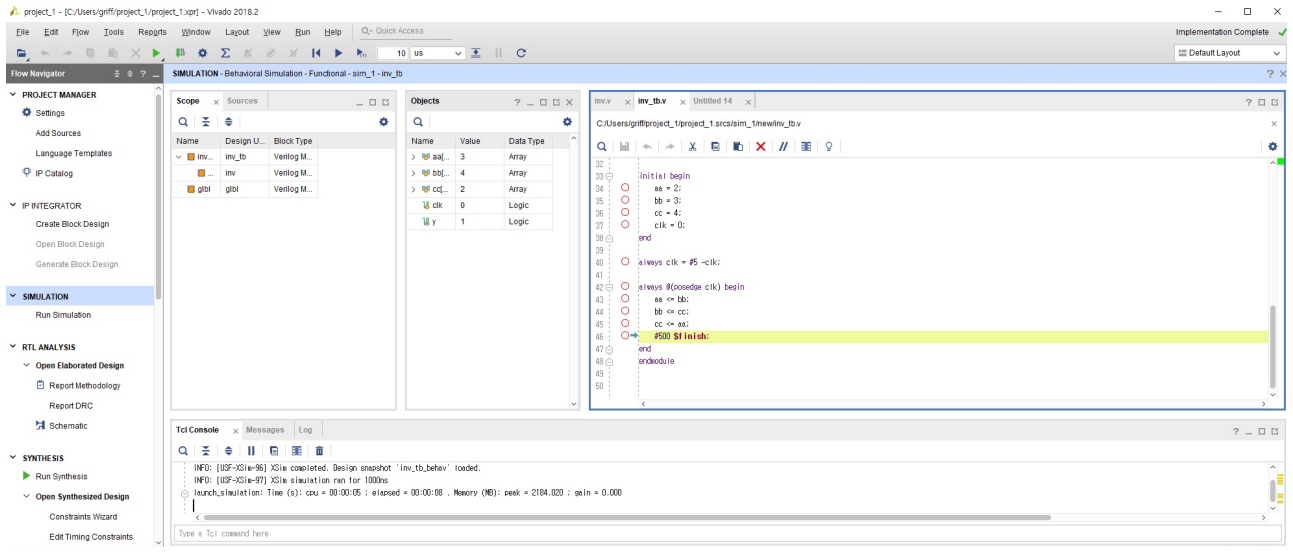
[예제 2] Blocking 절차형 할당문 예제 코드



[그림 1] Blocking 절차형 할당문 Simulation 결과

```
initial begin
    aa = 2;
    bb = 3;
    cc = 4;
    clk = 0;
end
always clk = #5 ~clk;
always @(posedge clk) begin
    aa <= bb;
    bb <= cc;
    cc <= aa;
    #500 $finish;
end
```

[예제 3] Blocking 절차형 할당문 예제 코드



[그림 2] Nonblocking 절차형 할당문 Simulation 결과

결과 값	Blocking	Nonblocking
aa	3	3
bb	4	4
cc	3	2

[표 2] 객체에 저장된 결과 값 비교

Blocking 할당문을 사용한 예제에서는 각 줄마다 순차적으로 계산과 동시에 저장이 일어나기 때문에 bb의 값이 aa에 할당이 되고난 뒤에 cc의 값이 bb에 할당되며, 그 다음에 aa의 값이 cc에 저장된다. 반면에 Nonblocking 할당문에서는 여러 구문이 동시에 evaluation된 후 한꺼번에 저장 작업이 수행되기 때문에 bb, cc, aa 값이 연산된 후 각각 aa, bb, cc에 할당이 된다. 그래서 cc에 할당된 값에서 서로 차이가 발생한다.

Ⅱ Verilog와 C언어의 비교

1. for문

Verilog에서의 for문은 C언어와 비슷하게 초기화, 종결 조건, 조건 변화식의 조건을 만족해야 한다. 대신 C언어와 다르게 중괄호를 쓸 때 begin과 end를 사용한다는 점에서 차이가 있다. 또한 Verilog에서는 반복 구문을 쓸 때 initial 또는 always 블록 안에서만 사용이 가능하다. 이는 아래에서 다룰 while문에서도 마찬가지이다. 예제를 들어 설명하면 아래와 같다.

```
always @(posedge clk) begin
    for (i = 0; i < ram_size; i = i + 1) begin
        data[i] = data[i] + 1;
    end
end
```

[예제 4] for문을 사용한 코드

이를 if문으로 쓴 동일한 코드로 바꾸면 다음과 같다.

```
always @(posedge clk) begin
    if (i < ram_size) begin
        data[i] <= data[i] + 1;
        i <= i + 1;
    end
end
```

[예제 5] for문 대신 if문을 사용한 동일한 코드

2. if문

Verilog와 c언어에서 조건문은 큰 차이가 없다. 조건문은 키워드 if와 else, 또는 필요하면 else if로 구성이 되며 if와 else if 키워드 뒤에는 반드시 조건을 써야 한다. 중괄호는 for문과 마찬가지로 begin과 end를 쓴다. 조건문의 수식을 한 줄만 쓴다면 begin과 end(C언어에서는 중괄호)를 쓸 필요가 없고, 여러 줄을 쓴다면 반드시 begin과 end로 묶어야 한다. 또한 조건문 안에 조건문을 사용하는 다중 조건문도 가능하다.

if문	if - else문	if - else if - else문

if (조건) begin
// statement
end
	if (조건) begin
	// statement
	end
	else begin
	// statement
	end
		if (조건) begin
		// statement
		end
		else if (조건) begin
		// statement
		end
		else begin
		// statement
		end

[표 3] 조건문 경우에 따른 형태

3. while문

while문은 while 뒤에 오는 조건이 만족하면 계속 반복 수행하고, 조건을 만족하지 않을 때 loop를 벗어난다. C언어와 큰 차이는 없으며, for문과 마찬가지로 initial 또는

always 블록 안에서만 사용 가능하다는 점을 유의해야 한다. 참고로 C언어와는 다르게 반복문을 시작할 때 begin에 레이블을 붙이면 'disable 레이블 이름'에 도달했을 때 레이블 이름이 붙인 반복문을 빠져나올 수 있다.

```
always @(posedge clk) begin
    while (index) begin
        index = index - 1;
    end
end
```

[예제 6] while문을 사용한 코드

4. case문

Verilog에서는 다중분기를 사용할 때 C언어에서 switch-case문과는 다르게 case문에서 break를 기술하지 않는다. 그러나 switch-case문에서 쓰는 default문은 사용 가능하다. 또한 case문을 끝낼 때에는 다른 구문들과는 다르게 endcase 키워드를 사용한다. 그리고 Verilog에서는 항상 always문에서만 사용 가능하다.

```
case (sel[1:0]) begin
    2'b00 : output <= 4'b1110;
    2'b01 : output <= 4'b1101;
    2'b10 : output <= 4'b1011;
    default : output <= 4'b0111;
endcase
```

[예제 7] case문을 사용한 코드

Ⅲ net형

1. net 자료형

Verilog에는 크게 net 자료형과 variable 자료형이 있다. variable 자료형은 앞에서 언

급한 절차형 할당문에서와 같이 임시로 값을 저장하는 데 주로 사용하며, 프로그래밍 언어의 변수와 유사한 개념이다. 반면에 net 자료형은 하드웨어 소자 간의 물리적인 연결을 추상화한 것이다. 연결된 장치 값으로 출력 값을 가지게 되며, 디폴트 자료형은 1비트의 wire이고 디폴트 초기 값은 z이다. 모든 assign문(연속 할당문)의 결과 값은 net이어야 한다는 특징이 있다. Verilog에서 사용하는 net 자료형에는 여러 가지가 있는데 이를 정리하면 표 4와 같다. 일반적으로는 wire 자료형을 주로 사용한다.

자료형 이름	의미
wire, tri	함축된 논리적 동작이나 기능을 갖지 않는 단순한 연결을 위한 net이다.
wor, trior	다중 구동자를 갖는 net이며, or 연산을 하는 wire이다.
wand, triand	다중 구동자를 갖는 net이며, and 연산을 하는 wire이다.
supply0	회로접지에 연결되는 net이다.
supply1	전원에 연결되는 net이다.
tri0	저항성 pulldown에 의해 접지로 연결되며, 아무 것도 인가되지 않으면 0이 되는 net이다.
tri1	저항성 pullup에 의해 전원으로 연결되며, 아무 것도 인가되지 않으면 1이 되는 net이다.
trireg	물리적인 net에 저장되는 전하를 모델링하며, 특이하게도 값을 저장하는 net에 사용한다.

[표 4] 객체에 저장된 결과 값 비교