2주차 결과보고서

전공: 컴퓨터공학 학년: 2학년 학번: 20191629 이름: 이주헌

1. 연속 할당문, 절차형 할당문의 차이를 비교하여 설명하시오.

연속 할당문은 assign 문을 이용해서 net 자료형 객체에 값을 할당하는 것을 말한다. Verilog에서 assign 문은 우변의 값에 변화가 생기면 자동으로 다시 연속 할당문을 실행하여 좌변의 변수를 업데이트[1]한다는 특징이 있다.

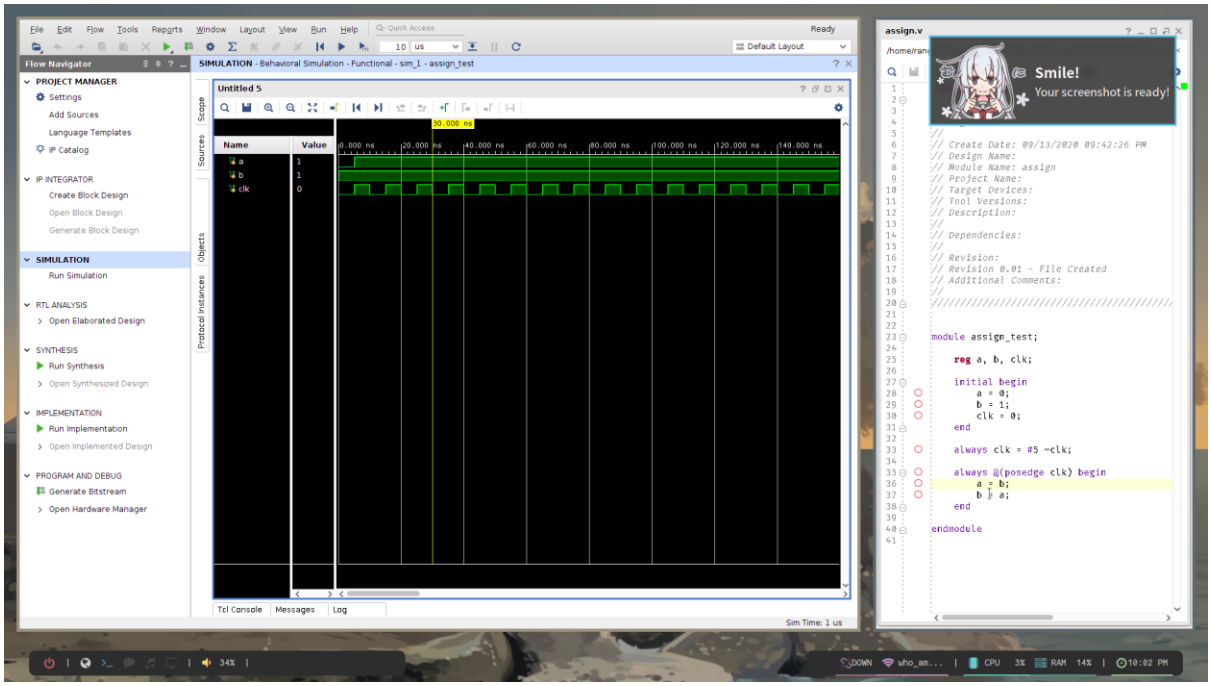
반면 절차형 할당문은 메모리 자료형의 변수에 값을 할당하는 할당문이다. 메모리에 값을 할당하는 것이기 때문에 절차형 할당문을 사용하면 이후 변수의 내용을 갱신하기 전까지는 이전의 값이 유지된다. Verilog에서 할당문은 보통 initial 또는 always 블록 안에 들어가며, blocking 할당문과 non-blocking 할당문의 두 종류가 있다. 할당문의 순서가 결과에 영향을 미치는 경우도 있다.

2. Blocking 및 non-blocking 문법의 차이를 simulation을 통해 설명하시오.

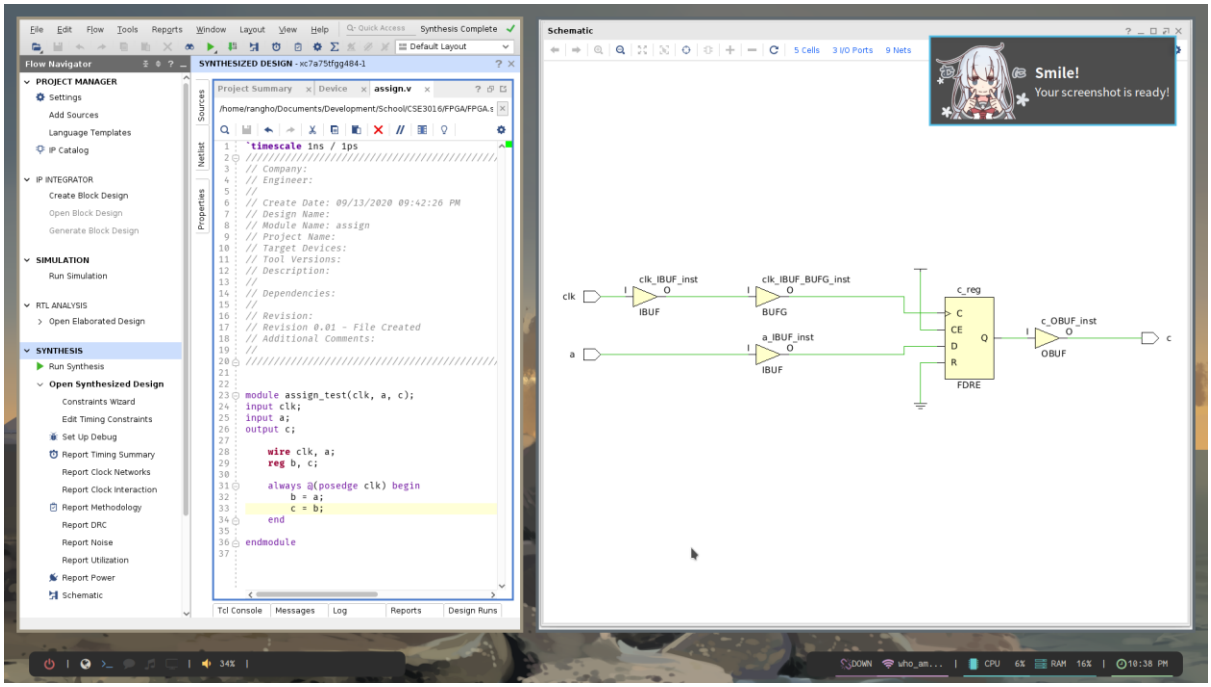
Blocking 문법과 같은 경우는 일반 컴퓨터의 할당 연산과 같이 순차적으로 할당이 실행된다. 즉, 현재 실행중인 할당문이 완료되기 전까지 다른 할당문이 실행되지 않으며,

[1] 정확히 말하면 값을 "할당"한다기보다는 두 객체를 "연결"한다는 것으로 해석하는 것이 정확하다. net 자료형의 특성상, 우변의 값이 변경되었다는 "이벤트"를 받아서 동작하는 것이 아니라 우변의 값 자체가 좌변과 연결되어 있기 때문이다.

코드에 작성한 순서대로 할당문이 실행된다.

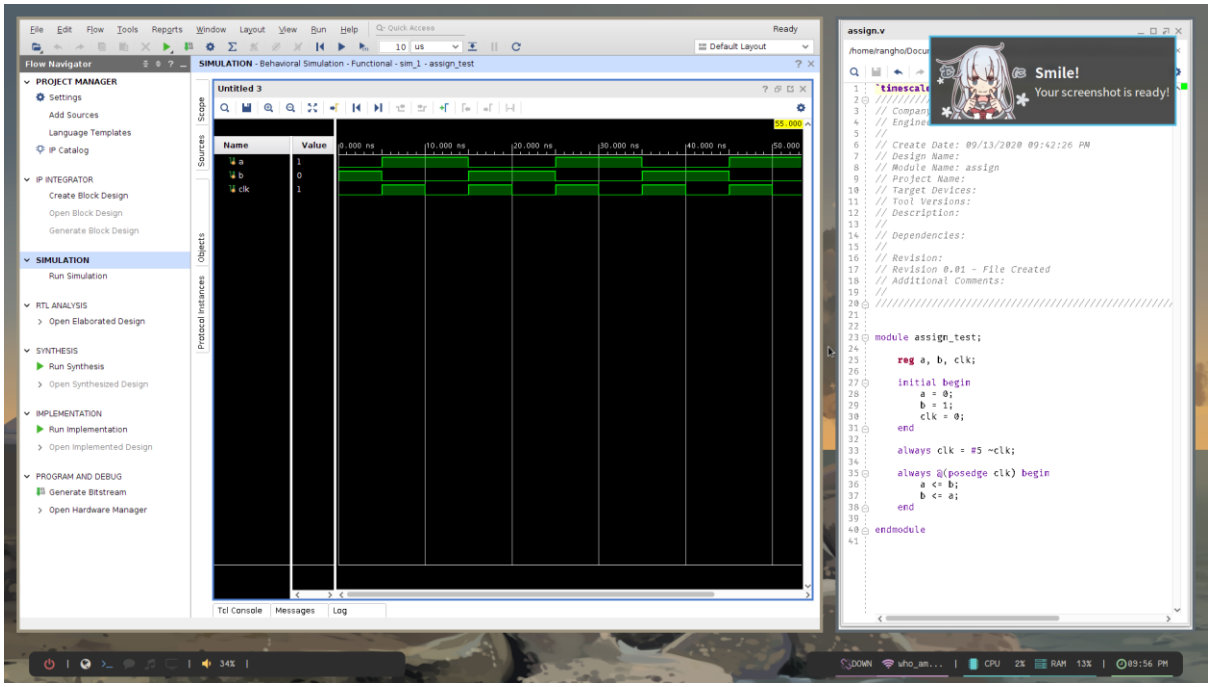


위 시뮬레이션에서 볼 수 있듯이, a = b; 할당문이 먼저 실행된 뒤 b = a; 할당문이 실행되어 결국 a와 b 모두 1이 된 것을 볼 수 있다. 따라서, 할당문이 의미가 없는 경우 컴파일러가 할당문을 삭제하여 최적화를 할 수 있다. 실제로, 아래 코드에서는 b 레지스터가 최적화되어 회로도에서 찾을 수 없다.

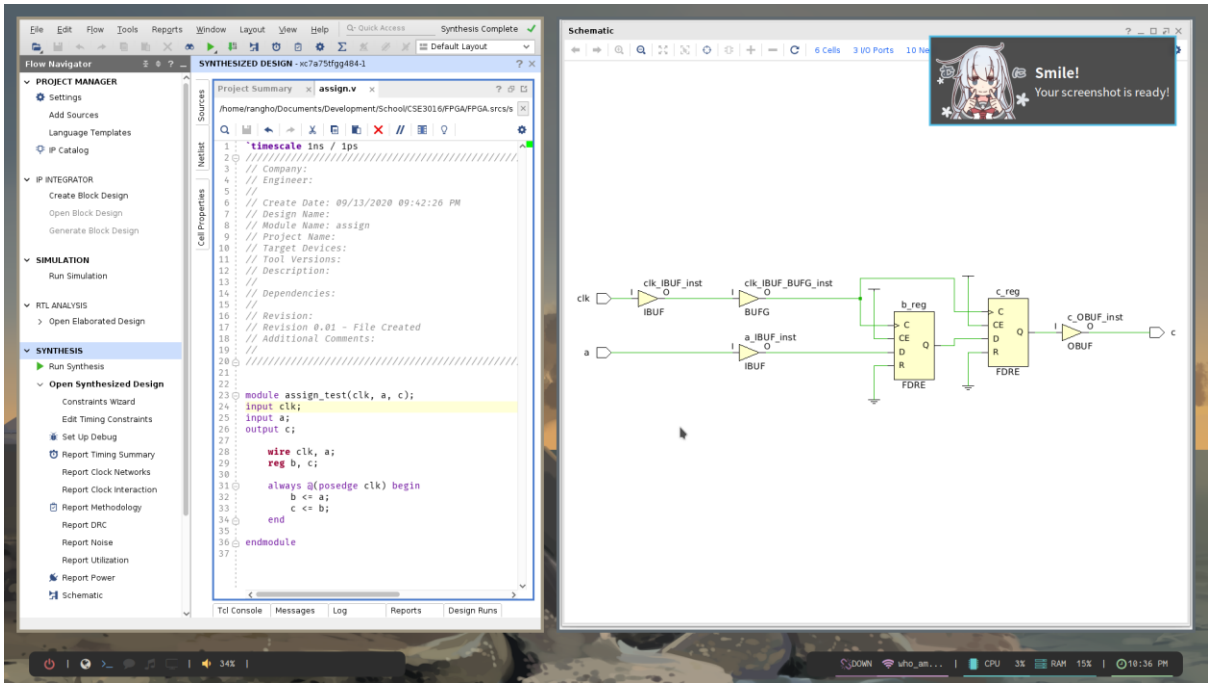


반대로 non-blocking 문법은 여러 개의 할당문을 동시에 평가하여 할당된 값을 결정한 후, 하나의 사이클에 할당을 진행한다. 실제로 하드웨어에 업로드하기 위해 non-blocking

할당문을 합성하면, 여러 레지스터를 합성하여 한 번에 할당하는 구조가 생기기 때문에 실제 하드웨어에 더 가까운 문법이라고 볼 수 있다.



시뮬레이션을 돌려 보면 클럭 사이클의 positive edge마다 a와 b의 값이 번갈아가며 바뀌고 있는 것을 볼 수 있는데, 이는 a <= b;와 b <= a;의 결과가 미리 평가되어 한 번에 할당이 진행되기 때문이다. Non-blocking 할당은 여러 레지스터를 사용하기 때문에, 위 blocking 할당에서 최적화 된 레지스터도 non-blocking 할당을 사용하면 남아 있게 된다.



2. Verilog의 for 문, if 문, while 문, case 문을 C 언어와 비교하여 설명하시오.

Verilog의 for, if, while 문은 C 언어에 존재하는 것과 같은 기능을 하며, 문법 또한 거의 비슷하다. for 문은 C 언어의 것과 같이, initial condition, condition, step assignment의 세 부분으로 이루어지며, 각 부분은 세미콜론으로 구분된다. for 문은 하나의 문(statement)을 가질 수 있으며, 여러 개의 문을 실행하고 싶다면 statement group으로 묶을 수 있다.

```
for (i = 0; i < 10; i = i + 1)
    $display("i: %d", i);

for (i = 0; i < 10; i = i + 1) begin
    a = b;
    b = a;
end

for (i = 0; i < 10; i = i + 1) fork
    a = b;
    b = a;
join
```

While 문 역시 C 언어의 것과 사용 방법이 같다. for 문과 마찬가지로 여러 개의 문을 실행하고 싶다면 begin-end나 fork-join statement group으로 묶어야 한다. 그러나 while 문을 사용하게 되면, 실제 FPGA에합성해서 업로드할 수 없는 경우가 생기기 때문에 조심해서 사용하여야 한다.

```
while (data < 10)
    $display("Hello!");'
```

If 문 역시 C언어의 것과 사용 방법이 같다. if 문 자체는 항상 FPGA에 합성 가능하므로 FPGA에 올리려는 용도로 사용해도 무방하다.

```
if (data < 10)
    $display("Data is less than ten!");
else if (data < 20)
    $display("Data is less than twenty!");
else
    $display("Data is massive!");
```

Verilog의 case 문은 C 언어의 switch-case문과 유사한 역할을 한다. 문법 역시도 C 언어와 유사한데, switch 키워드가 사라지고 그 자리에 case 키워드가 위치하며, C switch-case의

case 키워드가 사라진 형태로 사용할 수 있다. 또 한 가지 차이점이 있는데, C에서는 하나의 케이스 안에 여러 개의 문을 사용할 수 있었지만 Verilog에서는 하나의 문밖에 실행하지 못한다. 따라서 여러 문을 사용하려면 for, if 문과 마찬가지로 statement group을 사용해야 한다.

```
case (input_bits) begin
    2'b00: $display("None!");
    2'b10: begin
        $display("One!");
        $display("Two!");
    end
    default: $display("Something's wrong!");
endcase
```

4. Verilog의 net 형 자료형에 대하여 조사하시오.

Net 자료형은 다른 프로그래밍 언어의 자료형처럼 메모리 공간을 모델링하는 것이 아니라 하드웨어 구조 사이의 연결 자체를 모델링하는 자료형이다. 따라서, net 자료형 자체는 데이터를 저장하지 않으며, 데이터 "드라이버"의 값을 바로 반환한다. Verilog는 9가지의 net 타입을 구현하고 있는데, 그 목록은 다음과 같다.

- wire/tri　　두 구조를 연결하는 "와이어"
- wor/trior　　여러 구조의 OR 결과값
- wand/triand　　여러 구조의 AND 결과값
- tri0　　tri-state 구조의 값을 pull-down
- tri1　　tri-state 구조의 값을 pull-up
- supply0　　논리적 0 상수
- supply1　　논리적 1 상수
- trireg　　tri-state일 때 마지막 값을 저장

만약 데이터가 다른 구조의 출력에 의해 결정되거나, 연속 할당 구문에 사용되어야 할 경우 net 자료형을 사용하여야 한다.