2주차 결과보고서

전공: 심리학과 학년: 3학년 학번: 20190345 이름: 김동현

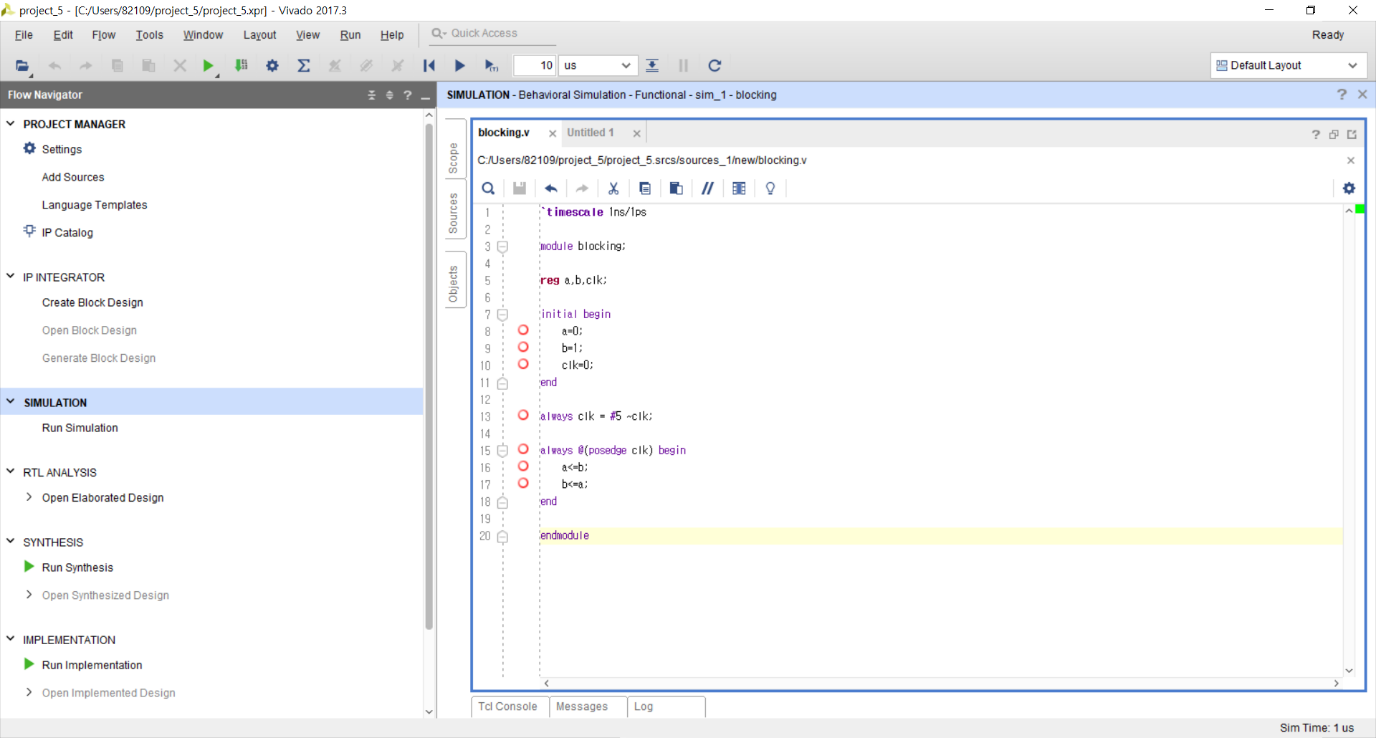
**1.**

Verilog 내에서 변수에 값을 지정하기 위한 할당문은 연속 할당문과 절차형 할당문이 존재한다. 연속 할당문은 디바이스의 물리적 연결을 나타내는 net형 변수에 논리 값을 할당하기 위해 사용된다. assign 구문을 사용하여 좌변의 net형 변수에 우변의 논리 값을 저장한다. 할당된 값을 제거하기 위해서는 deaasign 구문을 사용한다. 연속 할당문의 특징은 우변의 값이 달라진다면, 자동으로 좌변의 변수에 값이 할당된다. 절차형 할당문은 추상적인 저장 장치인 register의 reg변수에 값을 할당하기 위해 사용된다. 주로 always, initial구문 내에서 사용되며, 문장이 순서대로 실행되어 변수의 값이 할당된다. 절차형 할당문의 종류에는 blocking 구문과 nonblocking 구문이 있다. 연속 할당문과 절차형 할당문의 차이점은 다음과 같다. 우선 값의 할당에서 연속 할당문은 할당이 변수가 바뀔 때에 자동으로 이루어지는 반면, 절차형 할당문은 변수에 값을 할당하는 구문이 나오기 전까지는 그 값을 유지하고 있다. 이로 인해 연속 할당문은 문장의 순서가 결과값에 영향을 주지 않는 반면, 절차형 할당문은 문장의 순서가 결과값에 영향을 준다.

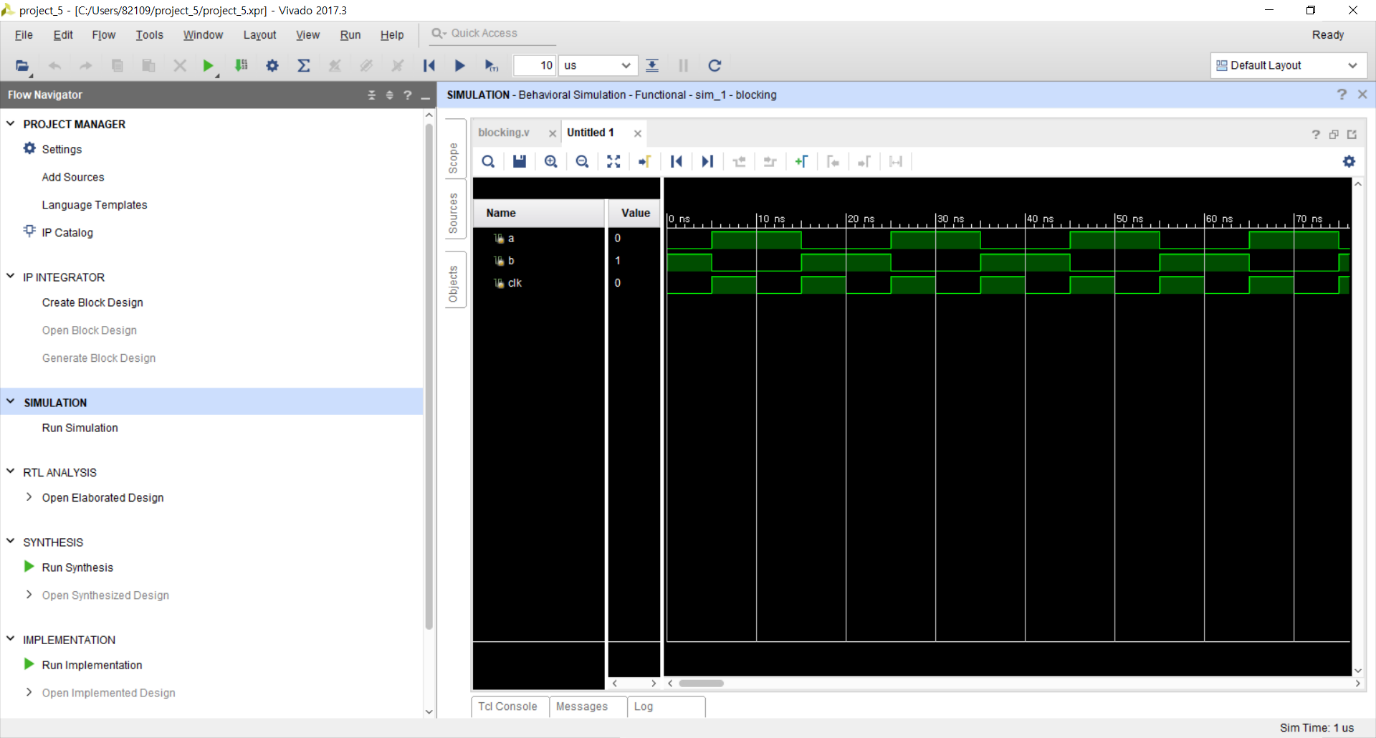
**2.**

Blocking 구문과 Nonblocking 구문의 차이는 다음과 같다. 우선 Blocking 구문은 단락 내의 문장들이 순차적으로 계산과 저장을 수행한다. 한 문장의 계산 및 저장이 완료된 이후, 다음 문장에 대한 계산 및 저장을 이어 가기 때문에 문장의 연산은 앞 문장들의 연산 결과에 영향을 받는다. 또한 Blocking 구문에서의 할당 연산자는 ‘=’을 사용한다.

Nonblocking 구문은 단락 내의 문장들이 한번에 계산과 저장을 수행한다. 문장들의 모든 계산을 수행한 뒤, 저장(할당)된다. 모든 연산이 이루어진 후, 저장을 수행하기 때문에 한 문장의 계산은 앞 문장들의 연산 결과와는 독립적이며 순서 또한 상관없다. 또한 Nonblocking 구문에서의 할당 연산자는 ‘<=’을 사용한다.

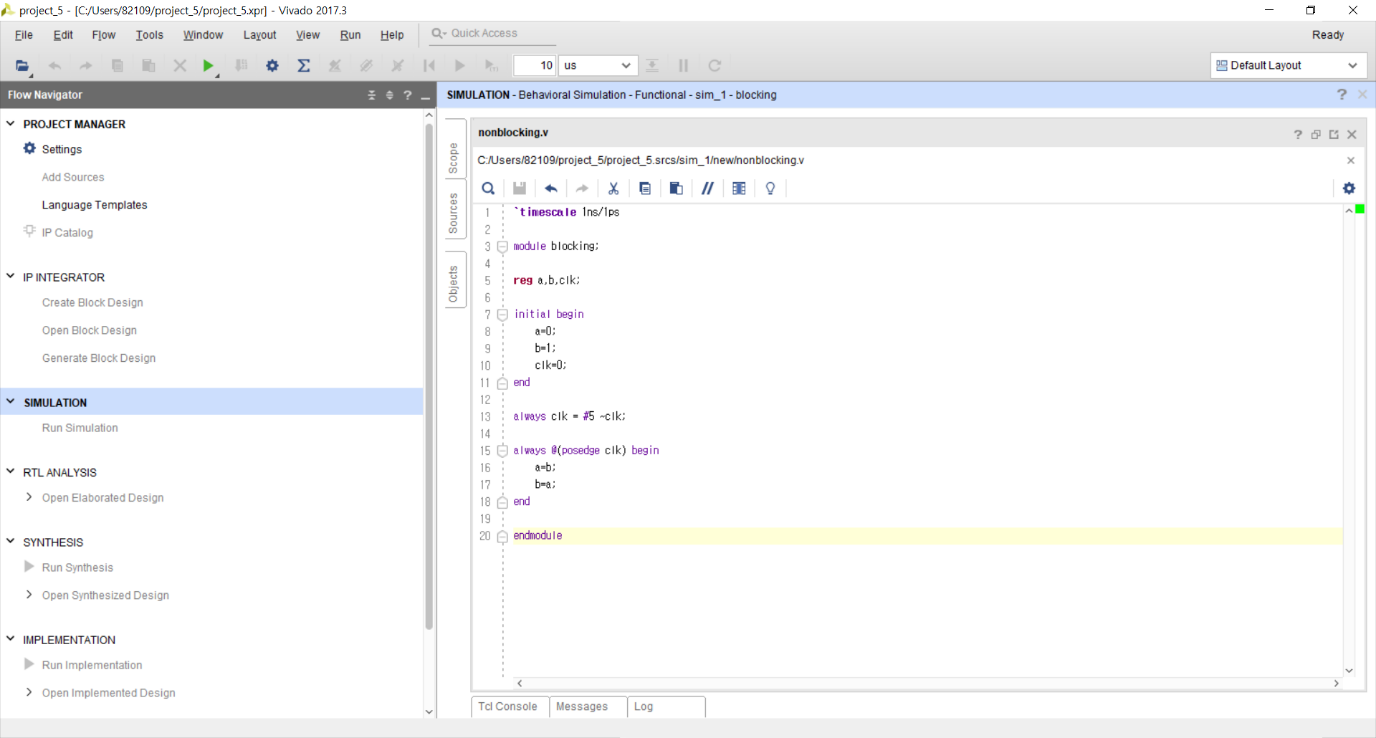


<그림1> Nonblocking statement source code

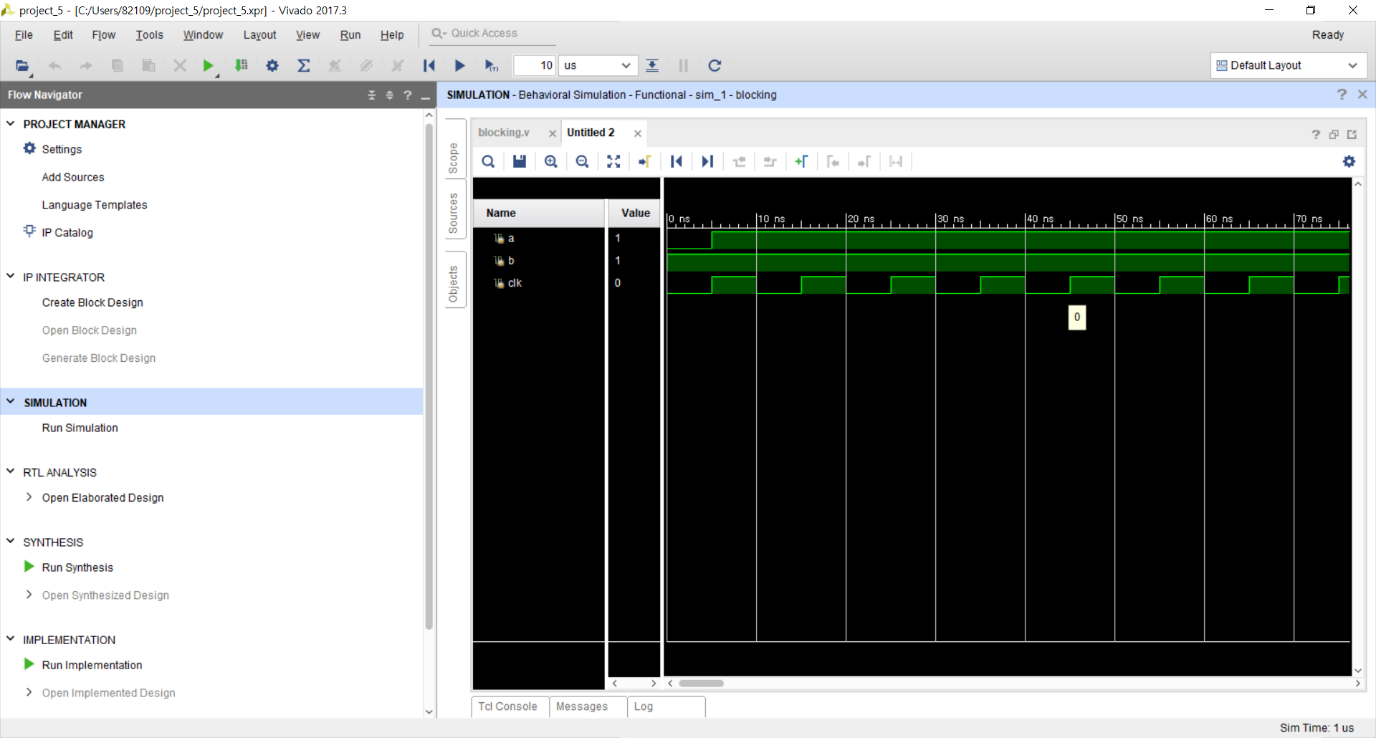


<그림2>Nonblocking statement simulation

위 <그림1>은 Nonblocking 구문을 구현한 source 코드이다. 변수의 초기값은 a=0, b=1이다. “always @(posedge clk) being”부터 “end”내의 구문이 매 클럭마다, clk의 값의 변화에 따라 실행된다. 변수 a는 b값으로, 변수 b는 a값이 할당되는데, 모든 연산 “a<=b”, “b<=a”가 실행된 후, a,b의 값이 저장되기 때문에 초기값에서 a=1,b=0이 되었다. <그림2>의 simulation 또한 매 클럭마다 a,b 값이 뒤바뀌는 모양으로 나타난다.



<그림3> Blocking statement source code



<그림4>Blocking statement simulation

위 <그림3>은 Blocking 구문을 구현한 source 코드이다. 변수의 초기값은 a=0, b=1이다. “always @(posedge clk) being”부터 “end”내의 구문이 매 클럭마다, clk의 값의 변화에 따라 실행된다. 변수 a는 b값으로, 변수 b는 a값이 할당되는데, “a=b”구문이 실행 및 저장이 된 후, “b=a”구문이 실행되기 때문에 변수의 값은 a=1, b=1이 되었다. 이후에는 a,b 모두 1인 상태가 유지된다. <그림4>의 simulation 또한 첫 클럭에서 a,b가 모두 1이 된 후, 값의 변화가 없다.

**3.**

For문의 경우, c언어에서는 함수 내 어느곳에서나 사용 가능하지만, 베릴로그 내에서는 always, initial 구문 내에서만 사용 가능하다. 또한 c언어는 for문 내의 구문을 중괄호(‘{‘,’}’)을 이용해 묶는 반면, 베릴로그에서는 for문 내의 구문을 “begin”, “end”로 묶어야 한다. 형식에서는 for(초기값; 조건식; 증감식)으로 구성되어 c언어와 베릴로그가 동일한 형식을 취한다.

If문의 경우, c언어에서는 함수 내 어느곳에서나 사용 가능하지만, 베릴로그 내에서는 always 구문 내에서만 사용 가능하다. 또한 c언어는 if문 내의 구문들을 중괄호로 묶지만, 베릴로그에서는 “begin”, “end”로 묶는다. 반면, if문 내의 구문이 하나인 경우는 굳이 묶지 않아도 된다는 점과, 중첩 if문이 가능하다는 점은 c언어와 베릴로그가 공통된 특성을 보인다.

While문의 경우, for문과 같이, c언어는 어느곳에서나, 베릴로그는 always, initial구문 내에서만 사용 가능하다. 또한 while문 내의 문장들을 c언어에서는 중괄호로, 베릴로그에서는 “begin”, “end”로 묶는다는 차이가 있다. 구현 방식에서는 while(조건문)의 구조로 동일하고, 조건문이 거짓이 되면 while문을 빠져나오는 규칙 또한 동일하다.

Case문의 경우, c언어는 case문 종료 시 break문을 사용하여 case문을 빠져나오지만, 베릴로그는 break문을 사용할 수 없다. 또한 case문을 c언어는 중괄호로 구분하는 반면, 베릴로그는 “begin”, ”end”을 사용하여 구분한다. 뿐만 아니라 베릴로그는 case문 종료시 “endcase’를 붙여야 하고, casex와 같은 case문 키워드도 존재한다.

4.

베릴로그의 net형 자료형은 하드웨어 요소들의 물리적 연결을 나타내기 위해 사용되는 변수이다. 연속할당문에서는 사용할 수 있지만, 절차형 할당문에서는 사용할 수 없다. Net형 자료형의 디폴드 값은 ‘z’이며, ‘z’는 high-impedance, 즉 어떤 전압도 드라이브 하지 않은 상태, 전기적으로 절연된 상태이다. Net형 자료형에는 wire, tri, wand, wor, triand, trior, supply0, supply1, trireg 등이 존재한다.

Wire와 tri는 회로를 구성하는 구성요소의 연결에 사용된다. Wire는 단일 게이트나 단일 연속 할당문에 의해 할당되는 net에 사용되며, tri는 0,1,고저항(z)의 3가지 상태를 가지는 3상태 net에 사용된다. Wand, wor은 단순 연결에만 사용되는 net인 wire에 and연산과 or연산을 추가한 것이다. Triand, trior 역시 3상태 회로의 단순 연결을 나타내는 tri에 and 연산과 or 연산을 추가한 것이다. Supply0는 회로의 접지에, supply1은 회로의 전원에 연결되는 net 이다.

5. 참고문헌

컴퓨터공학실험2 교재

If문, <https://peterfab.com/ref/verilog/verilog_renerta/source/vrg00019.htm>

Loop문, <https://peterfab.com/ref/verilog/verilog_renerta/source/vrg00023.htm>

Case문, <https://peterfab.com/ref/verilog/verilog_renerta/source/vrg00004.htm>