2주차 결과보고서

전공: 컴퓨터공학과/생명과학과 학년: 9학기 학번: 20181435 이름: 박다희

**1.**

Verilog에서 객체에 값을 줄 때에 연속 할당문이나 절차형 할당문을 사용한다. 객체의 종류에 따라 쓰이는 할당문이 다른데 연속 할당문은 net 자료형 객체에 값을 지정할 때 사용되고, 절차형 할당문은 reg, realtime, 메모리 변수 등 variable형 객체에 값을 지정할 때 사용된다.

연속 할당문은 AND, OR, XOR등과 같은 논리 연산자를 이용해 값을 계산하고 물리적 회로에 연결한다. 연속 할당문은 항상 작동하며 assign문을 통해 디바이스의 물리적인 연결을 의미하는 변수인 net 자료형 객체에 값을 지정하고, 입력 피연산자의 값이 변경될 때마다 연산이 다시 이루어지며 출력 값이 업데이트 되는 하드웨어적인 특징을 갖는다. 변수에 대한 assign 문의 영향을 제거할 때는 deassign 구문을 사용한다. 연속 할당문은 procedure 내부에서는 사용할 수 없다.

절차형 할당문은 연속 할당문과 달리 blocking statement와 non-blocking statement로 나뉘어 variable에 값을 할당하며, procedure 내부에서 사용할 수 있지만 문장의 실행에 의해서만 변수의 값이 갱신되기 때문에 다음 절차형 할당문에 의해 값이 바뀔 때 까지 기존의 값을 유지된다는 차이­­점이 있다. 실행 순서는 문장이 나열된 순으로 진행되며 소프트웨어적인 특성을 가지고 있다.

**2. ­**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

위의 사진은 reg 변수로 a, b, clk을 선언하고 각각의 변수를 0, 1, 0으로 초기화했다. 그 다음 always 구문 내부에 a는 b의 값으로, b는 a의 값으로 바꿔주는 코드를 non-blocking statement로 작성해준 뒤 simulation을 했다. 밑의 사진은 simulation의 결과다.

**스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명**작성해준 timescale 구문에서 1ns/1ps로 선언하였으므로 #5는 5ns을 의미하고 5ns마다 always 구문이 실행되는 것을 알 수 있다. 따라서 simulation 결과를 보면 5ns마다 그 값이 계속 바뀌는 것을 볼 수 있다. Nona-blocking statement는 값을 할당할 때 ‘<=’ 기호를 사용하며 begin~end 내부에서 end를 만날 때 까지 값이 바뀌지 않고 계산을 다 수행한 후에 end를 만나면 모든 값들의 저장이 이루어진다. 따라서 end를 만날 때에 a는 b의 값인 1로 바뀌고 b는 a의 값인 0으로 바뀐다. 반면에 blocking statement의 결과를 확인해보면 다음과 같다.

**텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명**

앞선 non-blocking statement를 사용한 코드와 비슷해 보이지만 always 구문을 보면 ‘=’기호로 값을 할당해 줌으로써 blocking statement로 짰음을 확인할 수 있다. 이 코드에 대한 simulation 결과는 다음과 같다.

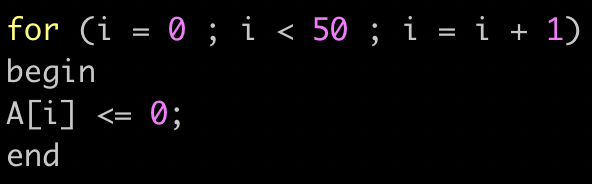
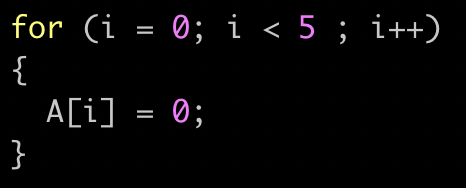
**스크린샷, 라인, 폰트, 텍스트이(가) 표시된 사진

자동 생성된 설명**

Timescale구문은 앞선 코드와 똑같으므로 단위 시간인 5ns마다 clk의 값이 바뀌는 것은 확인할 수 있지만 a값과 b값의 경우는 그 변화가 non-blocking statement와 다르다는 것을 볼 수 있다. 그 이유는 blocking statement는 계산과 동시에 저장이 이루어지므로 변수에 값을 할당까지 끝낸 후에 다음 줄로 넘어가 할당문을 실행하기 때문이다. 따라서 위의 코드에서는 a의 값이 1로 바뀌고 난 후에 b=a구문이 실행되어 b의 값이 1이 되는 것이다.

**3.**

1) for문



왼쪽의 사진은 c언어의 for문이고, 오른쪽의 사진은 Verilog의 for문이다. Verilog의 for문은 C언어의 for문처럼 변수 초기설정, 반복문이 돌아가는 조건, 한 번 수행한 후에 수행되는 문까지 조건식을 적지만 i++같은 연산은 불가능하다. 따라서 i = i + 1로 적어줘야 하며 c언어와 다르게 중괄호 대신에 begin~end를 사용한다. Verilog에서는 always문이나 initial문 내에서만 반복문을 사용한다.

2) if문

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

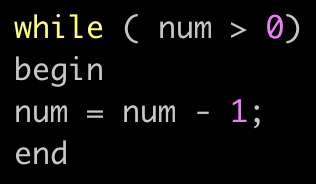
자동 생성된 설명텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

왼쪽 사진의 C언어의 if문처럼 오른쪽 사진의 Verilog의 if문도 비슷하지만 차이점은 앞의 for문에서와 같이 중괄호대신에 begin~end을 사용한다. 실행 순서는 c언어와 같이 if문 조건식에 만족하지 않으면 else문으로 내려간다. 또한 조건식이 만족할 경우 수행하는 수식이 한 줄만 있다면 begin~end는 사용하지 않아도 된다.

3) while문

텍스트, 폰트, 스크린샷, 그래픽이(가) 표시된 사진

자동 생성된 설명

왼쪽 사진이 c언어의 while문이고, 오른쪽 사진이 Verilog의 while문이다. Verilog에서의 while문도 c언어와 비슷하게 사용하면 되는데 중괄호 대신에 begin~end를 사용해야한다. Verilog에서도 while문 안의 조건이 거짓이 되면 반복문을 종료한다.

4) case문

**텍스트, 폰트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명폰트, 텍스트, 스크린샷, 그래픽이(가) 표시된 사진

자동 생성된 설명**

왼쪽 사진이 c언어의 case문이고 오른쪽 사진이 Verilog의 case문이다. C언어와 달리 Verilog에서는 break나 switch를 사용하지 않고, case문이 switch역할을 수행한다. 따라서 case의 값에 따라 해당하는 명령문을 실행한 뒤 종료한다. Verilog에서 case문을 작성할 때는 case(값)으로 시작하고 (값 : 명령문)으로 해당하는 값의 수행문을 적어준 뒤 마지막에 endcase를 적어줘야 한다.

**4.**

Net 자료형은 디바이스 간의 물리적인 연결을 나타내는 데 사용되는 자료형이다. 논리 회로에서 신호를 연결 또는 전달할 때 사용되며 연속 할당문과 같은 구동자에 의해 값이 지속적으로 유지되고 값을 변수에 저장하지 않는다.

Net 자료형은 다양한 유형으로 정의될 수 있으며 이 유형들에 따라 여러 신호 유형을 가질 수 있다. 주된 유형으로는 wire, wand, wor, tri, triand, trior, supply, trireg 등이 있다. Wire은 단순 연결할 때 사용하고, tri는 wire과 기능은 똑같지만 3상태 net에서 사용한다. Wand와 triand는 and기능으로, 2개 이상의 디바이스들의 출력을 연결한다. Wor과 trior은 wand와 triand와 비슷하지만 or기능을 한다. Supply는 0은 ground, 1은 Vdd를 나타내고 전압공급기의 세기이다. Trireg는 물리적인 net에 저장된 전하를 모델링한다.

Net 자료형 변수는 trireg를 제외하고는 구동자와 연결되어 있지 않으면 1비트의 wire값인 high-impedance(z)값을 가진다.