2주차 결과보고서

전공: 컴퓨터공학과 학년: 3학년 학번: 20211547 이름: 신지원

**1.**

연속 할당문(continuous assignmen) 은 assign문을 통해 net형 객체에 값을 할당하는 구문을 말한다. 오른쪽에 변화가 생기면 왼쪽의 객체에 변화된 값을 할당하는 방식이다. 절차형 할당문(Procedural Assignment) 은 always, function 등을 통해 벡터나 스칼라 변수에 값을 할당하는 구문을 말한다. 문장이 나열된 순서대로 시행되어 변수 값을 갱신한다. 변수 값으로는 reg, integer, time, realtime 자료형과 메모리 변수가 있으며 blocking, nonblocking 할당문을 사용한다.

연속 할당문과 절차형 할당문의 가장 큰 차이는 module 내에 할당문의 기술순서의 영향이다. 연속 할당문은 우변의 이벤트에 따라 수행되기 때문에 기술 순서가 작동하는 데에 영향을 미치지 않지만 절차형 할당문은 순서대로 수행되기 때문에 기술 순서가 작동하는 데에 영향을 미친다. 따라서 연속 할당문은 결과 값을 좌변의 net으로 drive 하는 하드웨어적 특징을 가지고 있는 반면, 절차형 할당문은 순서대로 실행되는 소프트웨어적 특징을 가지고 있다. 아래 예시를 통하여 코드를 통한 차이를 볼 수 있다.

- 연속 할당문

**wire a;**

**assign a = b & c;**

첫 번째 줄에서 net 형 객체인 a를 명시적으로 선언하였고, 그 다음 줄에서 a의 값을 할당하는 것을 볼 수 있다. 따라서 만약 b나 c에게 이벤트가 생긴다면 a의 값도 변화하게 될 것이다.

**wire assign a = b & c;**

a를 암시적으로 할당한 연속 할당문이다. 앞서 명시적으로 선언한 명시적 연속 할당문과 같은 기능을 수행한다.

- 절차형 할당문

**reg a;**

**initial begin**

**a = 1’b0;**

**end**

**always begin**

**#20 a = ~a;**

**end**

첫 번째 줄에서 변수 a를 선언하고 그 다음 세 줄을 통해 a의 초기값을 설정하였다. 또한 나머지 코드를 통하여 a에게 주기 신호를 선언하였음을 알 수 있다. 따라서 이는 순서의 영향을 받으며 코드가 더 추가된다고 가정하여도 당연히 순서의 영향을 받을 것이다.

**2.**

Blocking 할당문은 현재 할당문의 실행이 모두 완료된 후에 다음 할당문이 실행되는 순차적 할당문이다. Non-blocking 할당문은 나열된 할당문들이 순차적인 흐름없이 정해진 스케줄에 따라 값이 할당된다. 예시를 들어보자면, 내가 ‘blocking’, ‘non-blocking’ 의 이름을 가지고 있는 두 직원에게 작업을 시키고자 하였다. 작업을 모두 마친 두 직원이 서류를 검토 받으러 나에게 왔을 때, ‘blocking’이라는 직원에겐 내가 서류 검토를 모두 마칠 때까지 앞에서 기다리라고 하겠지만 ‘non-blocking’이라는 직원에겐 내가 서류검토할 동안 다른 업무를 봐도 괜찮다고 할 것이다. 코드와 simulation으로 그 차이를 더 살펴보았다.

**텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명**

blocking 코드

텍스트, 스크린샷, 소프트웨어, 번호이(가) 표시된 사진

자동 생성된 설명

Non-blocking 코드

Blocking 과 Non-blocking 의 코드에서 가장 크게 나타나는 차이점은 blocking 은 “=” 를 사용하지만, non-blocking 은 “<=” 를 사용한다는 것이다.

스크린샷, 텍스트, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

blocking simulation

스크린샷, 텍스트, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

Non-blocking simulation

Simulation으로 알 수 있듯이, blocking의 simulation 은 clock 에 첫번째 신호가 가해졌을 때부터 끝까지 a의 값이 1이 된다. b의 값은 초기 설정되었던 값인 1이 끝까지 유지된다. 그 이유는 always 안에 있는 실행문인 a=b; b=a; 가 순서대로 실행되기 때문에 첫 번째 신호부터 a=b; 에 의해 이미 a의 값이 b와 동일한 1이 되었기 때문이다. 따라서 a와 b의 값은 끝까지 1로 유지된다. 반면 non-blocking 의 simulation을 살펴보았을 때, a와 b의 결과값이 항상 반대로 나온다. 그 이유는 always 안에 있는 실행문인 a<=b; b<=a; 가 동시에 실행되기 때문이다. a가 0이고 b가 1로 초기 설정되었지만 실행문에 따라 동시에 실행되어 첫 번째 clock 신호가 가해졌을 때 a는 1로 b는 0으로 변환하게 된다.

이처럼 non-blocking 구문은 2가지의 절차로 진행되며, 우변의 값이 event queue에서 평가되고 이후 좌편의 값이 업데이트된다. 이때 blocking 구문들과 $display 의 task 는 함께 처리되기 때문에 non-blocking 구문의 값을 확인하고 싶다면, $monitor, $strobe 등의 task 를 이용하여 알아볼 수 있을 것이다.

**3.**

우선 Verilog 와 C언어는 목적에서부터 차이를 지닌다. Verilog 는 하드웨어 설계에 사용되고, C언어는 소프트웨어 설계에 사용되기 때문에 제어 구조의 의미나 동작에서 다르게 표현되거나 해석될 수 있다. 따라서 for문을 비롯한 다른 문법에서도 차이를 지닌다.

- for 문

“for” 라는 키워드를 작성하여 반복문을 구성한다는 자체는 C언어와 Verilog 모두 동일하다. 하지만 구체적인 작성 방법에 차이가 있다. 일반적으로 Verilog 에서는 for문을 직접 사용하기보단 generate 안에 있는 for 문을 사용한다고 이해하는 것이 더 올바르다. 이를 generate loop 라고 말한다. (generate- endgenerate 구조를 사용하는 것이 필수는 아니다.)

generate loop 내부에 선언된 wire 등은 지역 변수로 인식되어loop iteration 간의 충돌이 일으키지 않는다. 보통loop index 로 사용하는 i 등은 c언어에서는 주로 int 로 사용한다. 하지만 verilog 에서는 genvar 라고 명시하여 loop index 임을 선언한다.

i) C언어

**int i;**

**for(i = 0; i < 3; i++) {**

**int w1,w2;**

**w1 = in1[i] & in2[i];**

**w2 = in1[i] | in2[i];**

**out[i] = w1 & w2;**

**}**

ii) Verilog

**genvar i;**

**generate**

**for(i = 0; i < 3; i = i + 1) begin : gen\_inst**

**wire w1, w2;**

**assign w1 = in1[i] & in2[i];**

**assign w2 = in1[i] | in2[i];**

**assign out[i] = w1 & w2;**

**end**

**endgenerate**

물론 C언어에서 ‘&’연산자는 비트 연산을 수행할 때 사용하지만 코드의 구조를 비교하기 위하여 할당문은 똑같이 작성하였다.

- if 문

Verilog에서 if 문은 하드웨어 설계에서 조건부 논리를 제어하는 데 사용되어 특정 조건이 충족될 때만 특정 논리 블록이 활성화되도록 설계하지만 C에서 if 문은 프로그램의 제어 흐름을 조건에 따라 변경하는 데 사용된다. 따라서 조건이 참일 때만 특정 코드 블록을 실행한다. 구조 측면에서 다른 점이 있다면 Verilog 에서는 하나의 조건에서 동작해야 할 문장이 여러 개라면, begin-end 로 묶어주어야 한다. 또한 if – else if 로 작성하였다면 마지막에 꼭 else 문을 넣어주어야 한다. 이는 원치 않은 latch의 발생을 방지하기 위한다고 한다. 마지막으로 if-else 문을 지나치게 많이 작성한다면 회로의 사이즈가 커져 하드웨어에 좋지 않다. 이유는 회로적으로 '2X1 Mux’의 회로가 표현되기 때문이다.

i) C언어

**if(a>0) {**

**data = 10;**

**address = 16;**

**{**

**else {**

**data = 20;**

**addres++;**

**}**

ii) Verilog

**if(a>0) begin**

**data = 10;**

**address = 16’hDEAD;**

**end**

**else begin**

**data = 20;**

**address = address + 1;**

**end**

- while 문

While 문도 마찬가지로 begin, end 가 추가되는 것과 크게 다르지 않다.

i) C언어

**int i=0;**

**while(i < 5) {**

**i++;**

**}**

ii) Verilog

**int i=0;**

**initial begin**

**clk = 1’b0;**

**while(i<5) begin**

**clk = #5 ~clk;**

**i=i+1;**

**end**

**end**

- case 문

Verilog에서 case 문은 여러 입력 조건을 통해 다양한 동작을 선택하는 데 사용된다. 다양한 입력 패턴에 따라 다른 동작을 수행하는 상태 머신을 설계할 때 사용된다. C에서의 switch 문은 정수 또는 열거형 변수의 값에 따라 여러 조건 분기를 처리하는 데 사용된다. C에서는 주로 프로그램의 제어 흐름을 변경하기 위해 사용된다는 점이 Verilog 와 다르다.

case 문도 마찬가지로 여러 개의 실행문을 담기 위해 begin ~ end 를 사용하여 작성할 수 있다.

i) C언어

**switch value {**

**case 1:**

**a = 1;**

**b = 2;**

**break;**

**case 2:**

**a = 2;**

**b = 3;**

**default:**

**a = 0;**

**b = 0;**

**}**

ii) Verilog

**case (value)**

**value 1 : begin**

**a = 1;**

**b = 2;**

**end**

**value 2 : begin**

**a = 2;**

**b = 3;**

**default : begin**

**a = 0;**

**b = 0;**

**endcase**

if-else 문의 설명에서 사이즈의 이유로 하드웨어에 좋지 않아 if문을 남발하지 말라고 언급하였다. if-else 문은 회로적으로 '2X1 Mux’의 회로가 표현되는데 마찬가지로 case 문은 ‘nX1 Mux’ 회로가 표현된다. 그 이유는 병렬적으로 고려하기 때문이다. 따라서 if-else 문보다 case 문이 하드웨어적으로 더 좋은 설계라고 할 수 있다. 또한 if 문과 마찬가지로 latch방지를 위하여 default 를 작성해주어야 한다.

**4.**

Net 자료형이란 element 간의 물리적인 연결을 추상화하여 나타낸 것을 말한다. Verilog 의 모든 신호는 데이터와 연결되어야 하는데, 이때 요소들을 연결짓고 값을 갖는 데이터를 net 데이터 유형이라고 말한다. Net의 신호는 항상 구동되어야 하고 값을 업데이트할 수 있어야 한다. Net의 종류에는 wire, tri, wand, for, triand, trior, supply, supply1, tri0, tri1, trireg 등이 있다.

wire – 연속적인 할당으로 코드를 구현할 때 사용된다. 함축된 논리적 동작이나 기능을 가지 않는, 단순한 연결을 위한 net이다.

tri – wire 와 마찬가지로 단순한 연결을 위한 net이지만 wire 과 달리 net의 값이 저장되지 않는다는 차이점을 지닌다.

wand – wired(wire 의 확장자) AND 형식을 의미하며 다중 구동자를 갖는 net이다.

wor – wired OR 형식을 의미하며 다중 구동자를 갖는 net이다.

triand – wand의 wire 를 tri 로 구현한 것이다. wand 와 달리 값을 저장하지 않는다.

trior - wor의 wire 를 tri 로 구현한 것이다. wor 과 달리 값을 저장하지 않는다.

supply0 – 접지에서 끌어오는 선을 표현하며 회로접지에 연결되는 net이다.

supply1 – 전원에서 끌어오는 선을 표현하며 전원에 연결되는 net 이다.

tri0 – 저항성 pulldown 에 의해 접지에 연결되는 net이다.

tri1 - 저항성 pulldown 에 의해 전워에 연결되는 net이다.

trireg – 이전에 구동된 값을 유지하는 net 이다.