2주차 결과보고서

전공: 물리학과 학년: 3학년 학번: 20191286 이름: 김나현

1. 연속 할당문, 절차형 할당문의 차이를 비교하여 설명하시오.

연속 할당문은 net 자료형에 값을 할당하는 것이고, 절차형 할당문은 변수에 값을 할당하는 것입니다. 연속 할당문에는 assign 구문이 해당되고, 절차형 할당문에는 always 블록, initial 블록, task, function 내부에 존재하는 할당문인 blocking문과 nonblocking문이 해당됩니다.

연속 할당문은 assign문을 이용하여 net 자료형 객체에 값을 할당하는 것으로, 우변의 값에 변화(event)가 생겼을 때 좌변 객체에 값의 할당이 이루어집니다. 연속 할당문은 암시적 연속 할당문과 명시적 연속 할당문으로 구분할 수 있는데

wire mynet = enable & data;

wire mynet;

assign mynet = enable & data;

위의 두 상자 안에 있는 할당문들은 같은 의미이지만, 첫 번째 상자 안에 있는 할당문은 암시적 연속 할당문으로, net 선언문 안에 연속 할당문을 포함시킨 경우이고, 두 번째 상자 안에 있는 할당문은 net 선언문 안에 연속 할당문이 포함되지 않았기 때문에 명시적 연속 할당문이라고 합니다.

연속 할당문에는 지연을 사용할 수 있는데 정규지연은 명시적 연속 할당문에서 assign 뒤에 지연 연산자 #을 사용하여 연속 할당문의 지연 값을 지정하는 것이고, 우변 피연산자 값의 변화가 좌변에 할당되기까지의 시간 간격을 지정하는 것입니다. 명시적 연속 할당문에서는 net 선언문에서도 지연 값을 지정할 수 있는데, 지정된 net 지연이 경과된 후에 할당이 이루어지게 됩니다. 암시적 연속 할당문에서도 마찬가지로 지연 값 지정이 가능한데, 선언과 지연 할당을 동시에 할 수 있다는 특징이 있습니다. 아래 상자는 순서대로 명시적 연속 할당문에서의 정규지연, net 지연, 암시적 연속 할당문에서의 지연을 나타낸 것입니다.

wire out;

assign #10 out = in1 & in2;

wire #10 out;

assign out = in1 & in2;

wire #10 out = in1 & in2;

절차형 할당문은 절차형 블록, 즉 always, initial, task, function 등의 프로시저 내부에서 사용되는 할당문으로, register 변수에 값을 할당하는 것이 목적이므로 절차형 할당문의 좌변에는 reg, integer, real, time 등의 register 자료형(variable 자료형)을 사용합니다. 절차형 할당문은 피연산자 값에 변화가 발생할 때마다 우변의 식이 평가되고, 그 결과 값이 좌변의 net를 구동(drive)하는 하드웨어적인 특성이 있는 연속 할당문과 달리, 문장이 나열된 순서대로 실행되어 할당문 좌변의 변수 값을 갱신하는 소프트웨어적인 특성이 있습니다. 절차형 할당문의 예시인 blocking문과 nonblocking문은 질문 2에서 자세히 다루었습니다.

1. Blocking 및 nonblocking 문법의 차이를 simulation을 통해 설명하시오.

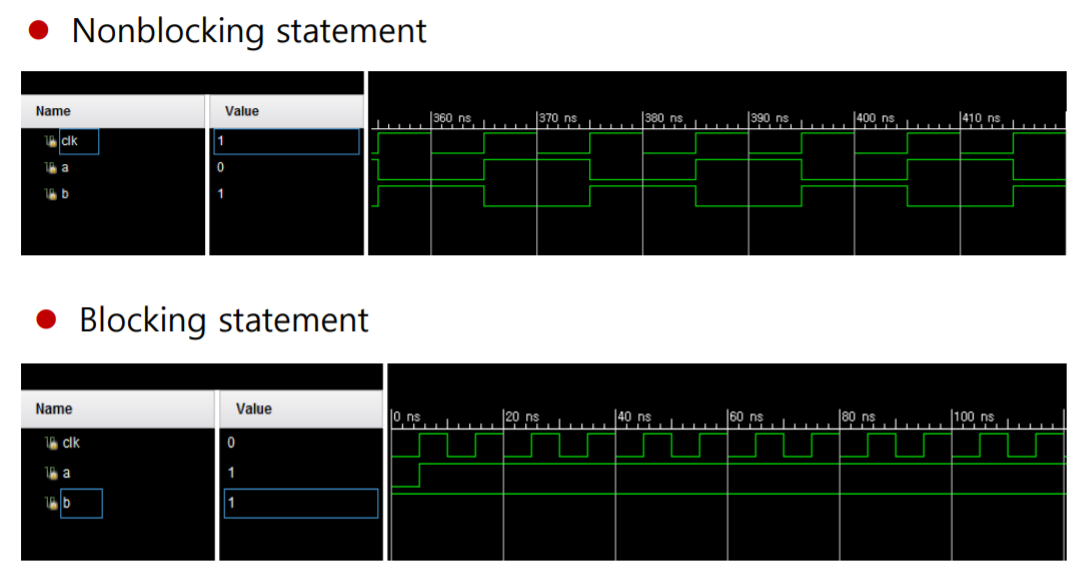
Blocking문과 nonblocking문은 Verilog 절차형 할당문의 예시이고, 문장이 나열된 순서대로, 순차적으로 실행되면서 할당문 좌변의 변수 값을 새롭게 갱신하는 구문을 의미합니다. 아래 <Figure 1>의 nonblocking문과 blocking문을 비교함으로써 알 수 있듯이, nonblocking문에서 사용되는 기호는 ‘<=’이고, blocking문에서 사용되는 기호는 ‘=’d입니다. Nonblocking문은 always 구문이나 initial 구문 등 내에서 begin~end 사이에 있는 모든 계산을 수행한 후, 한 번에 저장 작업이 수행되는 반면, blocking문은 마찬가지로 always 구문이나 initial 구문 등 내에서 begin~end 사이에 있는 모든 문장들이 line by line으로, 계산과 동시에 저장이 이루어집니다. Blocking, nonblocking이라는 표현을 사용한 것도 blocking 구문에서는 앞줄에 있는 내용이 실행되지 않을 경우 뒷줄이 실행되는 것을 막아(block) 순차적으로 실행될 수 있게 하기 때문이고 반대로 nonblocking 구문에서는 앞줄의 실행 여부와 상관없이 뒷줄의 실행을 막지 않아(nonblock) 동시에 여러 개의 문장을 평가하고 한 번에 저장하기 때문입니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

<Figure 1> nonblocking문과 blocking문의 예시

<Figure 1>에서 두 절차형 할당문에 공통적으로 들어있는 initial문으로 인해, a는 0, b는 1, clk은 0으로 초기화 되었음을 알 수 있습니다. 또한 always 구문 중 두 번째로 나오는 always문을 보면, clk이 달라질 때마다 always문 안에 있는 begin-end 블록이 실행되므로 line by line으로, 순차적으로 계산과 동시에 저장이 이루어지는 blocking statement의 예시에서는 a 변수 값이 b의 값인 1로 바뀌고 저장되므로 b 변수 값은 다시 a의 값인 1로 바뀌므로 b는 변함없이 1입니다. 이후에도 clk이 달라질 때마다 해당 always문이 실행되어도 a는 1, b는 1로 변하지 않으므로 시뮬레이션 결과 또한 그럴 것임을 예상할 수 있습니다. 하지만, begin~end 사이에 있는 모든 문장이 동시에 실행되고 그 다음에 저장되는 nonblocking statement의 예시에서는 a 변수 값이 b로 바뀌는 동시에, b변수 값은 a로 바뀌고 저장되므로, clk이 바뀔 때마다 a와 b가 0과 1, 1과 0이 됨을 반복할 것입니다.



<Figure 2> <Figure 1> nonblocking문과 blocking문의 시뮬레이션 결과

실제로 <Figure 1>의 시뮬레이션 결과인 <Figure 2>를 보면, blocking statement에서는 a가 1로 바뀌고 이후 clk이 달라져도(상승 엣지에서도) a와 b가 1로 유지되는 모습을 확인할 수 있고, nonblocking statement에서는 a와 b가 초기에 0과 1로 초기화된 후, clk이 달라질 때마다(상승 엣지마다) 1과 0, 0과 1로 반복되는 모습을 확인할 수 있습니다. 이 때, clk이 상승 엣지일 때마다 always 구문이 실행되는 이유는 <Figure 1>에서 posedge라고 설정하였기 때문이므로 하강 엣지에서 a와 b가 바뀌도록 설정하고 싶다면 posedge가 아닌 negedge를 사용하면 됩니다.

1. Verilog의 for문, if문, while문, case문을 c언어와 비교하여 설명하시오.

Verilog의 if-else문은 아래와 같이 c언어의 if-else문과 동일하게 조건식이 참이면, statement\_true 부분을 실행하고 조건식이 거짓이면 statement\_false 부분을 실행합니다.

if (expression)

statement\_true;

else

statement\_false;

이때, verilog는 c언어와 달리 0과 1 외에도 unknown을 의미하는 x와 high-impedence를 의미하는 z라는 논리 값을 갖기 때문에 verilog에서 조건식이 참이라는 것은 0이 아닌 값을 의미하고, 조건식이 거짓이라는 것은 0과 x, z를 의미합니다. Verilog에서도 c언어와 동일하게 else 부분이 존재하지 않을 수 있는데 이때는 조건식이 거짓이면 c언어와 동일하게 변수들이 달라지지 않고 할당받은 값을 유지하게 됩니다. 또한 verilog에서도 if문 안에 if문이 있는, nested if문을 사용할 수 있지만 두 줄 이상의 실행문이 있을 때, 중괄호 { }를 사용하는 c언어와 달리, verilog에서는 begin~end로 묶어 다음 상자와 같이 표현하게 됩니다.

if ( a ) begin

if ( x > y )

max = x;

else

max = y;

end

else

max = x+y;

else

statement\_false;

Verilog의 case문은 c언어의 switch문과 같은 역할을 하는 것이라고 생각할 수 있습니다. switch문과 동일하게 case 조건식의 값과 일치하는 case\_item의 문장을 실행하고, 조건식은 각 비트가 0, 1, x, z 중 하나로 정확히 같은 경우에만 일치한다고 판단합니다. If-else문과 동일하게 각 조건에서 두 개 이상의 실행문이 있을 때에는 begin~end로 묶어 표현하며 case문 안에 case문이 있는, nested case문이 가능하고 조건식의 값과 일치하는 case\_item이 존재하지 않는 경우 default항이 실행됩니다.

case (expression)

case\_item {, case\_item} : statement\_or\_null;

default : statement\_or\_null;

endcase

Verilog에서도 c언어와 동일하게 for문이난 while문과 같은 반복문이 존재합니다. c언어에서 반복할 횟수를 알고 있을 때 주로 사용하는 for문은 verilog에서도 반복 횟수를 제어하는 변수에 의해 조건식의 값이 거짓이 될 때까지 반복적으로 실행됩니다. 마찬가지로 c언어에서 반복할 횟수를 알지 못하지만 반복을 멈출 조건을 알고 있는 경우 사용하는 while문은, verilog에서도 조건식의 값이 거짓이 될 때까지 반복적으로 문장을 실행하기 위해 사용됩니다. c언어에서도 while문의 조건식이 참이 아니면 한 번도 while문을 실행하지 않고 뛰어넘는 경우가 있듯이 verilog에서도 조건식의 초기값이 거짓이면 문장이 아예 단 한 번도 실행되지 않을 수 있습니다.

for (initial\_assignment; expression; step\_assignment)

statement or statement group

while (expression)

statement or statement group

1. Verilog의 net형 자료형에 대해서 조사하시오.

Verilog 데이터 타입은 크게 입출력, register 자료형(variable 자료형)과 net 자료형으로 구분할 수 있습니다. 이 때, net 자료형은 논리 게이트나 모듈 등 하드웨어 요소들의 물리적인 연결을 나타내기 위한 것으로, wire나 tri가 대표적인 net 자료형의 예시입니다. Net 자료형은 연속 할당문과 같은 구동자의 값에 의해 net 값이 연속적으로 유지되며 값을 저장하지 않습니다. 또한 구동자가 연결되지 않으면 default 값인 high-impedence(z)로 표현되지만 예외적으로 trireg net의 경우, default 값이 알 수 없는 값을 의미하는 x로 표현됩니다. wire는 변수들이 모듈 내에서 어떻게 연결되어 있는지를 나타내 주는 변수이고 앞서 언급하였듯이 데이터를 전달할 수는 있지만 저장을 할 수는 없습니다. 버스 형태의 레지스터 등을 wire로 재정의하여 사용할 수 있으며, 1비트 wire는 생략이 가능합니다. 즉, 다음과 같은 상자 안에서, 첫 번째 줄과 같이 1비트 wire는 생략이 가능하고, 두 번째 줄과 같이 8비트 버스와 세 번째 줄과 같이 초기 값이 0인 와이어를 선언하는 것이 가능합니다.

wire w1, w2;

wire [7:0] bus;

wire enable = 1’b0;

tri는 선을 서로 연결할 때 사용하는 것으로 하드웨어에서 3상태(tri-state)가 된다는 점에서 wire와 차이가 있고 이 외에도 verilog의 net 자료형에는 wand, wor, triand, trior, supply0, supply1, tri0, tir1, trireg이 있습니다. Wand와 wor, triand, trior가 다중 구동자를 갖는 net이라는 공통점이 있지만, wand, triand와 wor, trior는 각각 wire-and(open collector logic), wire-or(emitter coupled logic)의 하드웨어 구현을 모델링하기 위한 것이라는 점에서 차이가 있고, wand, wor와 달리, triand와 trior는 tri와 동일하게 하드웨어에서 3상태(tri-state)가 된다는 차이가 있습니다. Supply0은 회로접지(ground)에 연결되는 net이며 supply1은 전원(power supply)에 연결되는 net이고, tri0은 저항성 pulldown에 의해 접지로 연결되는 net이며 tri1은 저항성 pullup에 의해 전원으로 연결되는 net입니다. 마지막으로 trireg는 물리적인 net에 의해 저장되는 전하를 모델링하기 위한 net이고, 앞서 언급하였듯이 net 자료형 중 유일하게 default 값이 알 수 없는 값을 의미하는 x로 표현되는 자료형입니다. 다음은 wire, tri net과 wand, triand net의 진리표, wor, trior net의 진리표를 순서대로 나열한 것입니다.

<Table 1> wire, tri net 진리표

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Wire/tri | 0 | 1 | x | z |
| 0 | 0 | x | x | 0 |
| 1 | x | 1 | x | 1 |
| x | x | x | x | x |
| z | 0 | 1 | x | z |

<Table 2> wand, triand net 진리표

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Wand/triand | 0 | 1 | x | z |
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | x | 1 |
| x | 0 | x | x | x |
| z | 0 | 1 | x | z |

<Table 3> wor, trior net 진리표

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Wor/trior | 0 | 1 | x | z |
| 0 | 0 | 1 | x | 0 |
| 1 | 1 | 1 | 1 | 1 |
| x | x | 1 | x | x |
| z | 0 | 1 | x | z |