2주차 결과보고서

전공: 아트엔테크놀로지 학년: 3학년 학번: 20191172 이름: 함승우

1. 연속 할당문, 절차형 할당문의 차이를 비교하여 설명하시오.

- 연속 할당문

연속할당(Continuous Assignment)은 net형 객체에 값을 할당하는 구문이다. 입력 피연산자의 값에 변화가 발생할 때마다 우변의 식이 평가되고, 그 결과값이 net에 저장된다.

예1)

wire y;

assign y = a & b;

예2)

wire y = a & b;

위의 예1과 예2는 동일한 동작을 수행하는데, 우변의 값에 변화가 발생하면(event) 발생하면, 좌변의 net형 객체에 그 값을 할당한다.

- 절차형 할당

절차할당(Procedural Assignment)은 문장이 나열된 순서대로 실행되어 할당 문 좌변의 변수 값을 갱신한다. procedural block을 통해 변수에 값을 할당한다. 위의 연속 할당문과는 다르게 우변의 값에 변화가 생길 때 값이 할당되는 것이 아니라, 해당 구문이 실행될 때 값이 할당된다. 절차형 할당문의 자주 쓰이는 always 문은 시뮬레이션이 실행되는 동안 반복적으로 무한히 실행된다. initial 구문은 시뮬레이션이 실행되는 동안 한번 만 실행이 된다. 아래에 설명할 Blocking statement과 NonBlocking statement는 절차형 할당문 내부에 포함된다.

2. Blocking 및 nonBlocking문법의 차이를 simulation을 통해 설명하시오.

Blocking과 nonBlocking의 가장 큰 차이점은 구문의 Line by Line을 순차적으로 계산하느냐, 동시에 저장 작업이 이루어지냐에 있다. Blocking은 변수를 할당할 때 ‘=’을 사용하고, 한 문장이 끝나면 다음 문장이 실행된다. nonBlocking은 변수를 할당할 때 ‘<=’를 사용하고, 동시에 저장 작업이 수행된다. 또한, HDL의 time에 관련하여 blocking 에서는 ‘#t 변수 = 연산’의 경우에는 t시간 이후, 연산하고 변수에 할당한다라고 해석해야 하지만, nonBlocking에서는 ‘변수 <= #t연산’ 연산하고 t시간 후 변수할당을 예약해 놓는다는 점에서 차이가 있다. 이렇듯 둘의 해석은 시간론적 관점에서 다르기 때문에, 두 가지를 모두 이해해야 올바른 해석이 가능하다.

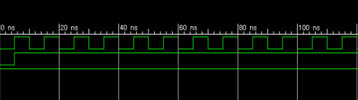
simulation을 사용하여 자세히 설명하겠다. 강의 자료의 nonblocking statement, blocking statement를 참고했다.

첫번째의 경우인 blocking statement인 경우,

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

initial begin~ 부분에서 a= 0. b= 1임을 알 수 있다. 그 이후 always 구문에서 a=b; b= a; 구문이 있는데, 이는 실행이 순차적으로 일어나기 때문에 a= b가 먼저 일어나, a= 1, b= 1이 되고, b = a가 일어나, a= 1, b= 1이 된다(변화가 없음). 아래의 시뮬레이션으로 확인할 수 있다.

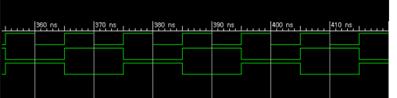


두번째의 경우인 nonblocking statement의 경우,

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

위의 예시와 마찬가지로, initial begin~에서 a= 0, b= 1임 알 수 있다. 이는 nonblocking statement기 때문에, 변수의 할당이 동시에 된다는 점을 기억하자. always 구문에 a<=b; b<=a; 이 두 식은 동시에 실행된다. 따라서, ‘a=b’ 식에 의해 변수 a에 b의 값인 1이 할당되고 이와 동시에 b=a 식에 의해 변수 b에 a의 변하기 전인 0이 할당된다. 따라서, a= 1, b = 0이 된다.



3. Verilog 의 for문, if문, while문, case문 을 C언어와 비교하여 설명하시오

|  |  |
| --- | --- |
| For문 in Verilog | For문 in C |
| for (초기값 ; 조건식 ; 변수 증감)  begin  처리문 or 블록 처리문  처리문 or 블록 처리문  end | For(초기값; 조건식; 변수증감)  {  연산식  연산식  } |

for문의 경우 (초기값 ; 조건식 ; 변수 증감)와 같은 형식은 동일하지만, verilog의 경우에는 2줄 이상이 되는 경우 block(begin, end)로 묶어줘야 하고, c의 경우에는 중괄호로 묶어줘야 한다. 또한 verilog의 경우에는 always, initial문 내에서만 사용이 가능하다.

|  |  |
| --- | --- |
| If문 in Verilog | If문 in C |
| If(조건)  Begin  처리문  처리문  end | If(조건){  처리문  처리문  } |

C언어의 경우에는 처리문이 두 줄 이상일 때 중괄호로 묶는 반면, Verilog는 begin-end의 block으로 묶는다. else문의 경우에도 동일하다.

|  |  |
| --- | --- |
| While문 in Verilog | While문 in C |
| While(조건)  Begin  처리문  처리문  End | While(조건문){  처리문  처리문  } |

위와 마찬가지로 두줄의 경우일 때 중괄호로 묶는 모습, 블록형태인 begin-end형식으로 묶는 모습으로 나눠 볼 수 있다.

|  |  |
| --- | --- |
| Case문 in Verilog | Case문 in C |
| case(x)  0 : a= 1;  1: b = 1;  Default: c =1;  endcase | switch(x)  {  Case 0:  A=1;  Break;  Case 1:  B=1;  Break;  Default:  C= 1  Break;  } |

c언어와 매우 유사하지만, endcase로 switch구문을 종료해야 하며, c언어와 다르게 break로 종료할 필요가 없다.

4. Verilog 의 net 형 자료형에 대해서 조사하시오.

net 자료형이란 모듈 또는 게이트 간의 물리적인 연결을 나타내는 자료형이다.

따라서 net 자료형 자체는 데이터를 저장하지 않는다.

net 자료형에는 wire, tri, wand, wor, triand, trior, supply0, supply1, tri0, tri1, trieg가 있다. 데이터가 다른 구조의 출력에 의해 결정되거나 연속 할당 구문에 사용되어야 할 경우에 net 자료형을 사용하여야 한다.

|  |  |
| --- | --- |
| wire | 함축된 논리적 동작이나 기능을 갖지 않는 단순한 연결을 위한 net |
| tri | 함축된 논리적 동작이나 기능을 갖지 않는 단순한 연결을 위한 net이며, 하드웨어에서 3상태(tri－state)가 되는 점이 wire와 다름 |
| wand | 다중 구동자를 갖는 net이며, ‘wired-and’(즉, open collector logic)의 하드웨어 구현을 모델링하기 위해 사용 |
| wor | 다중 구동자를 갖는 net이며, ‘wired-or’(즉, emitter coupled logic)의 하드웨어 구현을 모델링하기 위해 사용 |
| triand | wand와 동일하게 다중 구동자를 갖는 net이며, 하드웨어에서 3상태(tri-state)를 갖는 점이 다름 |
| trior | wor와 동일하게 다중 구동자를 갖는 net이며 하드웨어에서 3상태(tri state)를 갖는 점이 다름 |
| supply0 | 회로접지(circuit ground)에 연결되는 net |
| supply1 | 전원(power supply) (power supply)에 연결되는 net |
| tri0 | 저항성 pulldown (resistive pulldown)에 의해 접지로 연결되는 net |
| tri1 | 저항성 pullup (resistive pullup)에 의해 전원으로 연결되는 net |
| trireg | 물리적인 net에 저장되는 전하를 모델링하는 net |