2주차 결과보고서

전공: 컴퓨터공학과 학년: 4학년 학번: 20191583 이름: 김태곤

**1. 연속 할당문, 절차형 할당문의 차이를 비교하여 설명하시오.**

1) 연속 할당문

연속 할당문은 주로 wire 타입의 신호에 사용되며, 신호 간의 연결이나 조합 논리 회로의 표현에 사용된다. 연속 할당문은 항상 always 블록 밖에서 사용되며, 할당된 신호의 값이 할당문에 나타난 소스 신호의 값이 변경될 때마다 자동으로 업데이트된다. 연속 할당문의 기본 형태는 다음과 같다.

*assign wireName = expression;*

이 형태를 통해 expression의 계산 결과가 실시간으로 wireName에 반영된다.

2) 절차형 할당문

절차형 할당문은 always 블록이나 initial 블록과 같은 절차형 블록 내에서 사용된다. 이 할당문은 순차 논리 회로의 동작을 모델링하는 데 주로 사용되며, 레지스터 타입의 변수에 값이 할당된다. 절차형 할당문은 블록 내에서 정해진 순서대로 실행되며, 특정 조건이나 이벤트가 발생했을 때만 값이 업데이트된다. 절차형 할당문에는 = Blocking 할당과 <= Non-Blocking 할당의 두 가지 유형이 있다. 절차형 할당문의 기본 형태는 다음과 같다.

*always @(Trigger Event) begin*

*대상\_신호 = 표현식; // Blocking 할당*

*대상\_신호 <= 표현식; // Non-Blocking 할당*

*end*

3) 연속 할당문과 절차형 할당문의 차이점

연속 할당문은 조합 논리를 표현하고 wire 타입의 신호에 할당된다. 이는 신호가 변경될 때마다 연속적으로 업데이트되어, 회로의 연결 상태를 나타낸다. 절차형 할당문은 순차 논리를 표현하는 데 사용되며, always 블록 내에서 실행된다. 레지스터 타입의 변수에 값을 할당하며, 특정 조건이나 이벤트에 의해 업데이트되는 것이 특징이다.

**2. Blocking 및 nonBlocking문법의 차이를 simulation을 통해 설명하시오.**

Verilog에서 blocking (=)과 non-blocking (<=) 할당은 신호값의 할당 방식에 있어서 중요한 차이를 가진다. 이들의 사용은 주로 always 블록 내에서 순차적(sequential) 또는 병렬적(parallel) 동작을 모델링하는 데 영향을 미친다.

1) Blocking

Blocking 할당은 할당문이 순차적으로 실행됨을 의미한다. 한 할당문이 완전히 끝난 후에야 다음 할당문이 실행된다. 이는 프로그래밍 언어에서 일반적으로 볼 수 있는 할당 방식과 유사하다. Blocking 할당은 주로 조합 논리를 모델링할 때 사용되지만, 순차 논리모델링에 사용될 때는 주의가 필요하다.

2) non-Blocking

Non-blocking 할당은 현재의 시뮬레이션 타임 스텝에서 모든 non-blocking 할당문이 병렬적으로 실행됨을 의미한다. 할당문의 실행 순서는 영향을 미치지 않으며, 모든 non-blocking 할당은 동일한 시뮬레이션 타임 스텝에서 동시에 일어난다. 이는 순차 논리를 모델링할 때 유용하며, 플립플롭(flip-flop)과 같은 순차적 요소의 동작을 정확히 표현할 수 있도록 한다.

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명 Blocking 할당 a = b;는 a에 b의 현재 값(0)을 할당하고 이 할당이 완료된 후 다음 문장이 실행된다.

b = 1;는 b에 1을 할당하여 이 시점에서 a=0, b=1, c=0, d=0가 된다.

Non-blocking 할당 c <= d;는 c에 d의 현재 값(0)을 할당하도록 예약하고, d <= 1;는 d에 1을 할당하도록 예약한다. 현재 시뮬레이션 타임 슬라이스의 끝에서 c와 d의 할당이 실행되므로, 최종적으로 c=0과 d=1이 된다. Non-blocking 할당으로 인해 c와 d의 값 변경이 동시에 발생한다.

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

Verilog는 하드웨어 설계 및 모델링을 위한 언어이고, C언어는 소프트웨어 프로그래밍 언어이기 때문에 동일한 제어 구조일지라도 사용되는 목적과 방식이 조금 다를 수 있다.

|  |  |  |
| --- | --- | --- |
| 예시 | Verilog | C언어 |
| for문 | always @(posedge clk) begin  for (i = 0; i < 10; i = i + 1)  out[i] = ~out[i];  end | for (int i = 0; i < 10; i++)  printf("%d ", i); |
| if문 | always @(a or b) begin  if (a == 1'b1)  result = b;  else  result = ~b;  end | if (a == 1)  result = b;  else  result = !b; |
| while문 | initial begin  int i = 0;  while (i < 10) begin  $display("i = %d", i);  i = i + 1;  end  end | while (i < 10) {  printf("i = %d\n", i);  i++;  } |
| case문 | always @(in) begin  case (in)  0 : out = 0;  1: out = 1;  2 : out = 2;  3 : out = 3;  endcase  end | switch (in) {  case 0: printf("0"); break;  case 1: printf("1"); break;  case 2: printf("2"); break;  case 3: printf("3"); break;  default: printf("Unknown\n");  } |

1) for문

Verilog의 for문은 주로 always 블록 내에서 반복적인 하드웨어 동작을 모델링하는 데 사용된다. C언오와 동일하게 for(초기식; 조건식; 변화식)으로 동일하게 선언한다. statement 부분에는 {}대신 begin과 end가 사용되며 statement가 한 줄인 경우 begin과 end를 생략할 수 있다.

2) if문

Verilog의 if문은 주로 always 블록 내에서 조건에 따라 하드웨어의 동작을 결정하는 데 사용된다. 전반적인 실행 방법은 C언어의 if문과 동일하며 if, else, else if 모두 지원한다.statement부분에는 {} 대신 begin과 end가 사용되며 statement가 한 줄인 경우 begin과 end를 생략할 수 있다.

3) while문

Verilog에서 while문은 주로 초기화나 시뮬레이션을 위해 사용되며, 합성 가능한 하드웨어 설계에서는 일반적으로 사용되지 않는다. while문의 조건이 참이면 begin과 end 사이의 명령문을 1회 실행한다. 전체적인 실행은 while문과 유사하다.

4) case문

Verilog의 case문은 하드웨어의 다양한 상태나 조건을 처리하기 위해 always 블록 내에서 사용된다. C언어의 case문과 달리 switch와 break의 사용 없이 case를 통해 나타낸다. case에서 받은 condition에 따라 condition : statements로 명령문을 실행한다. case문의 종료는 endcase를 통해 이루어진다.

**4. Verilog 의 net 형 자료형에 대해서 조사하시오.**

Verilog에서 net 형 자료형은 주로 연결된 회로 요소 간의 물리적 연결을 표현하는 데 사용된다. 주로 신호선을 모델링할 때 사용되며, 신호의 흐름을 나타낼 때 사용한다. net은 하드웨어의 와이어링에 해당하며, 연속 할당문에 주로 사용된다. 크기를 지정하지 않으면 기본적으로 1비트의 wire 값을 가진다. net형 자료형에는 wire, tri, wand, wor, tri1, tri0 등이 있다.

wire : wire는 가장 기본적인 net 타입으로, 입력과 출력 사이의 연결, 그리고 다른 요소들 사이의 논리적 연결을 표현하는 데 사용된다. wire는 기본적으로 단방향 신호 전달만을 지원하며, 기본값은 1비트다.

tri : tri는 세 가지 상태(1, 0, 고립(Z))를 가질 수 있는 신호선을 표현할 때 사용된다.

wand : wand는 wire-And net으로, 여러 드라이버가 연결된 경우에 모든 드라이버의 논리적 AND를 출력한다.

wor : wor는 wire-Or net으로, 연결된 모든 드라이버의 논리적 OR를 출력한다.

tri1, tri0 : tri1과 tri0은 tri net의 변형으로, 기본적으로 각각 1과 0의 강제 값을 가진다.