2주차 결과보고서

전공: 아트&테크놀로지학과 학년: 4학년 학번: 20191048 이름: 김도솔

**1.**

**연속 할당문 (Continuous Assignment Statement):**

연속 할당문은 주로 net 자료형 변수(디바이스의 물리적인 연결을 나타내는 변수)에 논리값을 할당하는 경우 사용된다. assign 구문은 입력 피연산자의 값이 변경될 때마다 우변의 식을 평가하고, 그 결과 값은 할당문 주변의 net 변수를 구동(drive)시킨다. 즉, 입력 값의 변화에 따라 연결된 net 변수가 즉시 업데이트되는 것을 말한다. deassign 구문은 변수에 대한 assign 문의 영향을 제거하는 데 사용된다. 즉, 할당된 값을 해제하여 변수가 더 이상 업데이트되지 않도록 한다. 연속 할당문을 사용하면 게이트의 기본 요소만을 사용할 때보다 더 추상적이고 복잡한 조합 논리 회로를 설계할 수 있으며, 코드의 길이를 줄일 수 있다. 그러나 연속 할당문은 절차형 할당문과 달리 프로시저(Procedure) 내부에서 사용할 수 없다.

**절차형 할당문 (Procedural Assignment Statement):**

절차형 할당문은 주로 variable 형 객체(reg, integer 등)에 값을 할당할 때 사용된다. 이러한 할당문은 always, initial, task, function과 같은 procedure 내부에서 사용이 가능하다. 할당문에 의해 한 번 값이 할당되면 다음 절차형 할당문에 의해 값이 갱신될 때까지 변수에 할당된 값이 유지되는 특성을 가진다. 따라서 assign문과 달리 순서에 변화가 생기면 회로가 바뀔 수 있으며, 문장이 실행이 된 후에 변수에 값이 할당되게 된다. 절차형 할당문은 Blocking Statement와 Non-blocking Statement로 나뉘는데, Blocking Statement는 할당 연산자로 ‘=’를 사용하고 Non Blocking Statement는 할당 연산자로 ‘<=’를 사용한다. 절차형 할당문을 통해 상태 기계, 조건부 동작, 회로 동작 등을 모델링할 수 있으며, 프로시저 내에서 사용되므로 시뮬레이션 동안 동작을 제어하거나 변수 값을 저장하는 데 유용하다.

**2.**

**Blocking Statement (직렬):**

할당 연산자로 ‘=’를 사용한다. Blocking Statement는 begin부터 end까지 line by line으로 계산과 동시에 저장이 이루어진다. 한 문장이 다른 문장을 blocking하고 진행하다가 현재 문장의 수행이 완료되면 다음 문장을 실행하는 것이다.

|  |
| --- |
| Blocking 문법 예시 코드 |
| `timescale 1ns / 1ps  module inv\_tb;  reg a, b, clk;  initial begin  a=0;  b=1;  clk=0;  end  always clk = #5 ~clk;  always @(posedge clk) begin  a=b;  b=a;  end  endmodule |
| simulation 결과 |
| Blocking Statement를 사용해 값을 할당했기 때문에 begin부터 end까지 한 줄씩 계산과 저장이 이루어진다. 따라서 begin-end 구문에서 먼저 a=b; 수행 시 a에 1이 저장되고, 다음으로 b=a; 수행 시 b에 위의 문장에서 업데이트된 a의 값 1이 저장된다. |

**Non Blocking Statement (병렬):**

할당 연산자로 ‘<=’를 사용한다. Non Blocking Statement는 begin과 end 사이에 있는 모든 코드를 병렬로 실행하고, 한번에 값을 저장한다. 즉 할당문이 수행될 때 begin-end 내부 변수들의 값은 end에 도달하기 전까지 변화하지 않는다. 주로 공통의 사건이 발생된 후에 여러 개의 데이터를 동시에 전송하거나 업데이트해야 할 때 유용하게 활용된다.

|  |
| --- |
| Nonblocking 문법 예시 코드 |
| `timescale 1ns / 1ps  module inv\_tb;  reg a, b, clk;  initial begin  a=0;  b=1;  clk=0;  end  always clk = #5 ~clk;  always @(posedge clk) begin  a<=b;  b<=a;  end  endmodule |
| simulation 결과 |
| Non Blocking Statement를 사용해 값을 할당했기 때문에 being-end 구문이 종료된 후 각 변수의 값이 저장된다. 즉 할당문이 수행될 때 begin-end 내부 변수들의 값은 end에 도달하기 전까지 변화하지 않는다. 따라서 being-end 구문이 끝나면 a<=b; 에 의해 a에 1이 할당되고, 동시에 b<=a; 에 의해 b에는 0이 할당된다. |

**3.**

**1) for문:**

Verilog에서 for문은 초기값, 종료 조건 및 제어 변수를 설정하여 특정 동작을 반복 실행할 수 있다는 점에서 C언어와 공통점을 가진다. 그러나 for문의 범위를 지정할 때 중괄호가 아닌 begin-end를 사용한다는 것과 제어 변수에 ++, -- 등의 연산자를 사용할 수 없다는 점에서 차이가 있다.

|  |  |
| --- | --- |
| Verilog 예시 코드 | C 언어 예시 코드 |
| for (i = 0; i < 10; i = i + 1) begin  // 코드 내용  end | for (int i = 0; i < 10; i++) {  // 코드 내용  } |

**2) if문:**

Verilog에서 if문은 조건에 따라 특정 문장의 수행 여부가 결정되고, 동일하게 if (조건)의 형태로 사용되며, if, else, else if 등을 사용한다는 점에서 C언어와 비슷하다. 그러나 if문의 범위를 지정할 때 중괄호가 아닌 begin-end를 사용한다는 점에서 차이점을 지닌다. 조건문 내부 코드가 한 줄만 있는 경우에는 begin-end 생략이 가능하다.

|  |  |
| --- | --- |
| Verilog 예시 코드 | C 언어 예시 코드 |
| if (condition) begin  // 코드 내용  end else begin  // 코드 내용  end | if (condition) {  // 코드 내용  } else {  // 코드 내용  } |

**3) while문:**

Verilog의 while문은 조건을 검사하고 해당 조건이 참인 경우 코드 블록을 계속해서 반복 실행하는 역할을 한다. 조건이 거짓이 되면 루프를 빠져나오고, C 언어와 같이 while (조건)의 형태로 작성된다. 그러나 이 또한 범위 지정 시 중괄호가 아닌 begin-end를 사용한다는 점에서 차이점이 있다.

|  |  |
| --- | --- |
| Verilog 예시 코드 | C 언어 예시 코드 |
| while (condition) begin  // 코드 내용  end | while (condition) {  // 코드 내용  } |

**4) case문:**

Verilog에서 case문은 특정 동작을 선택하여 수행하는 데 사용된다. C언어와 달리 case-endcase로 범위를 지정하며, switch와 break를 사용하지 않는다. break가 없어도 해당 case만 실행 후 종료된다는 특징이 있다.

|  |  |
| --- | --- |
| Verilog 예시 코드 | C 언어 예시 코드 |
| case (input\_condition)  2'b00: // 입력이 00인 경우  // 코드 내용  2'b01: // 입력이 01인 경우  // 코드 내용  default: // 기본 동작  // 코드 내용  endcase | switch (input\_condition) {  case 0: // 입력이 0인 경우  // 코드 내용  break;  case 1: // 입력이 1인 경우  // 코드 내용  break;  default: // 기본 동작  // 코드 내용  break;  } |

**4.**

**Verilog에서 net 자료형 변수**는 디바이스의 물리적인 연결을 표현하기 위해 사용되는 변수이다. 이러한 변수들은 디바이스 내의 논리 게이트 간의 연결 또는 모듈의 포트 간의 연결을 나타내는 데 사용된다. net 변수는 주로 연속 할당문(assign)을 통해 값을 할당하고 논리 회로의 입력과 출력을 연결하는 데 활용된다. Verilog에서 주로 사용되는 net 자료형 변수의 종류는 다음과 같은 것들이 있다.

1. Wire (wire): wire 변수는 주로 논리 회로의 출력을 나타내는 데 사용된다. wire는 연결된 다른 wire나 reg 변수의 값을 읽을 수 있지만, 직접적으로 값을 변경할 수는 없다.
2. Tri-state (tri): tri 변수는 Tri-state 로직을 모델링하는 데 사용된다. Tri-state 로직은 출력을 높은 임피던스(Z) 상태로 만들 수 있는 논리 회로를 나타낸다. tri 변수는 0, 1, 또는 Z 상태 값을 가질 수 있으며, 다른 회로의 wire나 reg와 연결하여 출력을 제어한다.
3. Wand (wand): wand 변수는 연결된 다른 wire나 reg와의 & 연산 결과를 저장하기 위해 사용된다.
4. Wor (wor): wor 변수는 연결된 다른 wire나 reg와의 | 연산 결과를 저장하기 위해 사용된다.