2주차 예비보고서

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

**1.**

**HDL이란?**

HDL (Hardware Description Language)은 하드웨어를 모델링하고 설계하는데 사용되는 프로그래밍 언어이다. HDL은 디지털 논리 게이트, 레지스터, 메모리 등과 같은 하드웨어 구성 요소를 정의하고, 이러한 요소들 간의 상호 작용을 기술하는 데 사용된다. HDL을 사용하여 디지털 회로를 설계하고 시뮬레이션하며, 이를 FPGA 또는 ASIC와 같은 디지털 논리 디바이스에 구현할 수 있다.

**Verilog이외의 HDL**

Verilog 이외의 HDL은 AHDL, CUPL, JHDL, MyHDL, VHDL 등 여러 언어가 있다. 이 중 몇 가지를 살펴보자면, JHDL은 자바, MyHDL은 파이썬, VHDL은 파스칼 기반 언어이다.

**VHDL**은 주로 FPGA 및 집적회로(IC) 디자인과 같은 전자 공학 회로의 자동화된 설계에 사용된다. Pascal과 유사한 문법을 가지고 있어 엄격한 구조적 프로그래밍 스타일을 따른다. 이로 인해 VHDL 코드는 상대적으로 구조화된 형태를 보이며, 모듈화와 계층화에 적합하다는 특징이 있다. VHDL은 복잡한 디지털 시스템, 고속 통신 시스템, 신호 처리, 무선 통신 및 아날로그 및 디지털 혼합 시스템 설계와 같이 고수준의 응용 분야에서 주로 사용된다.

**JHDL**은 자바 프로그래밍 언어를 기반으로 하기에 자바의 객체 지향 프로그래밍 기능을 활용할 수 있다. 따라서 하드웨어 요소를 객체로 모델링하고 다양한 객체 간의 상호 작용을 표현할 수 있다. 이로써 개발자는 코드를 보다 모듈화하고 재사용 가능한 구성 요소로 설계할 수 있다.

**MyHDL**은 Python 언어를 기반으로 하기 때문에 JHDL과 마찬가지로 객체 지향 설계를 지원한다. 따라서 하드웨어 요소를 Python의 클래스로 모델링할 수 있어 모듈화와 재사용성을 향상시킬 수 있다. MyHDL은 하드웨어 설계 교육 및 프로토타이핑에 자주 사용되는데, 프로그래밍 입문 언어인 Python을 기반으로 하기에 개발 초보자가 접근하기 쉽기 때문이다.

**2.**

**Verilog의 역사와 발전 과정**

1980년대 중반: Verilog는 1980년대 중반에 프라부 고엘(Prabhu Goel)이 설립한 회사 Gateway Design Automation에 의해 처음 개발되었다. 초기 버전은 Verilog-X로 알려져 있으며, 회로 설계를 목적으로 만들어졌다.

1985년: 새로운 버전의 verilog인 verilog-XL과 그 simulator가 공개되었다.

1989년: Gateway를 인수한 Cadence 회사가 Verilog HDL을 공개했다.

1995년: IEEE표준이 만들어져 Verilog-95(IEEE 1364-1995)로 자리매김 하였고, 이로써 Verilog는 업계에서 널리 사용되는 언어로 성장할 수 있게 되었다.

2002년: Verilog를 확장하여 시스템 레벨 디자인 및 검증에 적합한 기능이 추가된 SystemVerilog가 발표되었다.

2005년: Open Verilog International (OVI)는 Verilog와 SystemVerilog를 통합해 새로운 표준(IEEE 1800)을 공식화했다. 이로써 다양한 기능과 향상된 라이브러리를 제공하게 되었다.

현재: Verilog는 FPGA 및 ASIC 디자인 분야에서 매우 널리 사용되며, 현재까지도 계속 발전하고 있다. 하드웨어 설계와 검증에서 중요한 역할을 하며, 업계 표준으로 자리 잡아 현대 디지털 하드웨어 설계 프로세스의 핵심 부분으로 자리 잡고 있다.

**3.**

**Verilog의 기본적인 구조**

모듈(Module)은 Verilog의 기본 설계 단위이며, 하나의 독립된 하드웨어 블록을 나타낸다. 이 블록은 다른 모듈에서 인스턴스화할 수 있다. 모듈은 머리부, 선언부, 몸체부 등 크게 세 부분으로 구성 된다.

1. 모듈의 머리부는 키워드 "module"로 시작하며 모듈의 이름, 포트 목록 및 세미콜론 ";"으로 끝난다. 모듈의 이름은 모듈을 식별하는 데 사용되기에 모듈 이름과 동일한 이름을 다른 곳에서 사용할 수 없다.

**\*** 포트(Port): 모듈은 입력과 출력 포트를 가진다. 입력 포트는 모듈로 데이터를 가져오고, 출력 포트는 모듈로부터 데이터를 내보낸다.

1. 모듈의 선언부에서는 포트 목록의 포트들에 대한 방향, 비트 폭, 그리고 "reg" 및 "wire"와 같은 데이터 형식의 선언, 그리고 "parameter"와 같은 매개 변수의 선언 등을 포함한다.
2. 모듈의 몸체부는 회로의 동작, 구조, 그리고 기능을 표현하는 다양한 구문들로 구성된다. 예를 들어 논리 게이트, 레지스터, 외부 모듈 인스턴스, 조건문, 순서 블록 등 회로의 동작을 제어하는 모든 코드가 포함될 수 있다.

**Verilog의 기본적인 문법**

1. 데이터 타입(Data Type): 레지스터(Register)는 추상적 저장 장치로서 값을 저장하는 역할을 하고, 넷(Net)은 디바이스의 물리적인 연결을 나타낸다. 예를 들어 게이트 간의 연결을 나타낸다. 레지스터에는 reg(절차형 할당문에 의해 값을 받는 객체), integer(정수형 변수) time/realtime(시간형 변수), real(실수형 변수) 등이 있다. 넷에는 wire(변수들이 모듈내에서 어떻게 연결되어 있는지 나타내 주는 변수), tri(선을 서로 연결할 때 사용, 주로 tri-state net에 사용) 등이 있다.
2. 상수 선언:

* 비트 수 제한이 있는 reg 값 선언: 비트 수를 명시하고 입력 형식과 값을 지정한다. 예를 들어, 8비트 크기의 정수 12를 선언하려면 다음과 같이 작성한다. -> 8'b12
* 크기를 지정하지 않은 값: 크기를 특별히 정하지 않아도 선언이 가능하다. 예를 들어, 정수 123을 선언하려면 (123)과 같이 사용할 수 있다. 또한 16진수인 35를 선언하려면 ('h35)와 같이 사용하며, 8진수인 234는 ('o234)로 표현한다.
* 크기를 정한 값: 값을 선언할 때 비트 크기를 명시적으로 지정할 수 있다. 예를 들어, 4비트 크기의 2진수 1111은 4'b1111로 표현된다. 마찬가지로, 4비트 크기의 16진수 f(=2진수 1111)는 4'hf, 4비트 크기의 10진수 15(=2진수 1111)는 4'd15로 표현된다.
* 부호가 있는 수: 음의 값을 나타내려면 부호 있는 수로 선언합니다. 예를 들어, 8비트 크기의 -7은 -8'd7으로 표현된다. 이때 음의 값은 2의 보수로 처리되어 저장된다.

1. 연산자:

- 산술 연산자: +, -, \*, /, \*\*

- 결합 및 반복 연산자: {}, {{}}

- 관계 연산자: >, >=, <, <=

- shift 연산자: <<, >>

- 조건 연산자: exp1 ? exp2 : exp3

- 논리 연산자: &&, ||, !, &, |, ~, ^ 등이 있다.

1. Verilog 구문:

* timescale: Verilog에서 시뮬레이션 및 시간 지연을 다룰 때 사용된다.

timescale {시간 단위}/{정밀도}

시간 단위(Time Unit): 이 부분을 선언하면 해당 Verilog 파일 내에서 사용되는 모든 시간 단위가 지정된 값으로 설정된다. 예를 들어, "1ns"로 설정하면 나머지 파일에서 모든 시간 값이 나노초 단위로 해석된다.

정밀도(Precision): 정밀도는 주어진 시간 단위 내에서 사용 가능한 소수점의 허용 범위를 나타낸다. 정밀도를 높이면 시뮬레이션의 정확도가 향상된다.

* assign: 연속할당문으로 피연산자의 값이 변경될 때마다 우변의 식을 평가하고 그 결과 값을 할당문 주변의 net에 전달한다. 변수에 대한 assign문의 영향을 제거하기 위해서 deassign문을 사용한다.
* always: 시뮬레이션이 동작하는 동안 반복적으로 수행된다. 따라서 타이밍 제어와 관련된 표현에 자주 쓰인다. @(sensitivity\_list)의 신호 중 하나 이상에 이벤트가 발생했을 때 구문 내부의 begin-end 블록이 실행된다.
* initial: 위의 구문과 달리 시뮬레이션이 진행되는 동안 한번만 실행된다. 이 구문의 절차형 문장들로 begin-end 블록이 구성되며 절차형 문장들은 순차적으로 실행된다.

1. Verilog 절차형 할당문:

나열된 문장을 순서대로 실행하여 할당문 좌변의 변수 값을 업데이트한다. 주로 always, initial, task, function과 같은 프로시저(Procedure) 내부에서 사용된다.

* Blocking Statement (직렬): 할당 연산자로 ‘=’를 사용한다. Blocking Statement는 begin부터 end까지 line by line으로 계산과 동시에 저장한다. 한 문장이 다른 문장을 blocking하고 진행하다가 현재 문장의 수행이 완료되면 다음 문장을 실행하는 것이다.
* Non Blocking Statement (병렬): 할당 연산자로 ‘<=’를 사용한다. Non Blocking Statement는 begin과 end 사이에 있는 모든 코드를 병렬로 실행하고, 한번에 값을 저장한다. 이는 공통의 사건이 발생된 후에 여러 개의 데이터를 동시에 전송하거나 업데이트해야 할 때 유용하게 활용된다.