2주차 예비보고서

전공 : 컴퓨터공학과 학년 : 3학년 학번 : 20211558 이름 : 윤준서

**1. HDL**

Hardware Description Language의 약자로, 직역하면 "하드웨어 기술 언어"인 HDL은 하드웨어를 설계하는 언어로, 회로 정보를 코드 형태로 표현하고 구조와 작동을 설계할 수 있게 한다. HDL은 알고리즘을 통해 디지털 시스템의 설계와 시뮬레이션, 검증이 모두 가능하며 FPGA 등의 전자 회로를 사용자가 직접 설계할 때 사용된다.

HDL의 종류는 Verilog 이외에도 여러 종류가 있다. **VHDL**은 디지털 회로와 혼합 신호를 표현하는 집적회로 기술언어다. 그 특징은 Verilog와 동일하나, Verilog보다 구체적 상황을 표현하기에 더 적합한 언어다. **AHDL**은 FPGA를 프로그램하기 위해서 알테라 사(altera)에서 개발한 하드웨어 기술 언어이다. Verilog, VHDL, AHDL은 모두 C에 기반한 언어이다. 이외에도 자바에 기반한 **JHDL**, 파이썬에 기반한 **MyHDL**이 있다.

**2. Verilog의 역사와 발전 과정**

1983년 - 미국 Gateway Design Automation사의 창립자인 Prabhu Goel이 HiLo와 C를 기반으로 **Verilog**를 최초로 개발했다.

1985년 - 상위 버전 **Verilog-XL**이 개발되었다.

1990년 ~ 1991년 - Cadence Design System사가 Gateway Design Automation사를 인수하고 년 **Verilog HDL**을 공개했다.

1993년 - IEEE Working Group을 구성하여 Verilog HDL의 표준화 작업이 시작되었다.

1995년 - **IEEE Std. 1364-1995**가 공개되었다.

2001년 ~ - 개정 버전인 **IEEE Std. 1364-2001**이 공개되었다. 이후 그 확장 형태인 **SystemVerilog**가 개발되었다.

**3. Verilog의 구조와 문법**

Verilog의 **기본** **구조**는 크게 머리부, 선언부, 몸체부 세 가지로 나뉘어진다.

**머리부** - 모듈 이름과 포트 목록들을 지정한다. 형태는 **<키워드 모듈> (모듈 이름) [포트 목록];** 이다. 모듈 이름은 키워드와 동일하게 사용할 수 없다.

**선언부** - 포트 목록에 나열된 포트들의 방향, 비트 폭, reg 및 wire 선언 , parameter 선언 등 모듈에서 필요로 하는 것들을 선언한다.

**몸체부** - 회로의 기능, 동작, 구조 등을 표현한다. Verilog 구문인 논리 합성용 구문과 시뮬레이션용 구문, 라이브러리용 구문으로 구성된다.

Verilog의 **기본** **문법**은 다음과 같다.

**자료형** - 크게 Register와 Net으로 나뉜다.

**Register**은 추상적 저장 장치로, 다음 네 가지가 있다.

**reg** - 절차형 할당문에 의해 값을 받는 객체.

**integer** - 정수형 변수.

**real** - 실수형 변수.

**time**, **realtime** - 시간형 변수. 시간을 체크할 때 시뮬레이션을 처리한다.

**Net**은 디바이스의 물리적인 연결을 의미하는데, 다음 두 가지가 있다.

**wire** - 변수들이 모듈 내에서 어떻게 연결돼 있는지 나타내는 변수다.

**tri** - 선을 서로 연결할 때 사용하는 변수다. wire와 달리 3상태 net에 사용된다.

**상수 선언** - 다음 세 개의 방식으로 상수 선언을 할 수 있다.

1. 크기를 정하지 않음 - **'(입력형식)(입력값)** ex). '12 (정수 12)

2. 크기를 정함 - **(비트 수)'(입력형식)(입력값)** ex). 4'b1111 (이진수 1111, 정수 15)

3. 음수 - **-(비트 수)'(입력형식)(입력값),** two's complement 형식으로 저장된다.

ex). -4'b1101 (이진수 1101, 정수 -3)

**연산자** - Verilog의 연산자는 C의 연산자와 유사하다.

**{}**, **{{}}** : 결합, 반복 **^** : 비트 xor

**+**, **-**, **\***, **/**, **\*\*** : 산술 **^~** (**~^**) : 비트 xnor

**%** : 나머지 **&** : 축약 and

**>**, **>=**, **<**, **<=** : 관계 **~&** : 축약 nand

**!** : 논리 부정 **|** : 축약 or

**&&** : 논리 and **~|** : 축약 nor

**||** : 논리 or **^** : 축약 xor

**==** : 논리 등가 **^~** (**~^**) : 축약 xnor

**!=** : 논리 부등 **<<** : 논리 왼쪽 시프트

**===** : case 등가 **>>** : 논리 오른쪽 시프트

**!==** : case 부등 **<<<** : 산술 왼쪽 시프트

**~** : 비트 부정 **>>>** : 산술 논리 시프트

**&** : 비트 and **? :** : 조건

**|** : 비트 or **or** : Event or

**Verilog 구문** - 앞선 C와 유사한 연산자와 달리 Verilog만의 구문들이 존재한다.

**Timescale** - **'timescale <시간단위>/<정밀도>** 의 형태로 사용된다. <시간단위>는 값을 지정하면 해당 파일 내의 모든 시간의 최소 단위는 그 값이 된다. <정밀도>는 주어진 시 간 단위로 구성 할 수 있는 가장 작은 지연을 나타내며, <시간 단위>와 관련하여 사용할 수 있는 소수점의 허용범위를 나타낸다.

**assign** - net 변수에 특정 논리 값을 지정할 때 사용된다. **조건 ? 참 :** **거짓**의 형태로도 쓰인다.

**always** - 시뮬레이션이 실행되는 동안 반복적으로 실행되기에, 타이밍 제어와 연관된 표 현에 유용하게 사용된다. if, else, case 등의 조건문은 always 구문 안에서 사용한다. @(sensitivity\_list)는 always 문의 실행을 제어하며 , sensitivity\_list의 신호들 중 하나 이 상에 변화가 발생했을 때 always 내부의 begin-end 블록이 실행된다. begin-end 블록은 C의 {}와 같은 역할을 한다.

**initial** - always 구문과 달리 시뮬레이션 진행 동안 한 번만 실행된다. begin-end 블록 내 부의 문구들이 순서대로 실행된다.

**반복문** - C에도 존재하는 for, while 문과 Verilog만의 반복문으로 나뉜다.

**forever** - 무한히 반복한다.

**repeat** - 지정 횟수만큼 반복한다.

**블록문** - C의 {}처럼 사용되며, begin-end과 fork-join 등이 있다. 주로 always 구문과 initial 구문에서 사용되는데, 블록문이 사용되면 Blocking statement, 사용되지 않으면 Non blocking statement로 불린다. Blocking statement는 순차적으로 계산과 저장이 이루 어진다. Non blocking statement는 우선 계산을 모두 시행한 후 동시에 저장이 이루어진 다. 보통 공통적인 사건이 발생 후 여러 데이터를 동시에 전송할 때 사용된다.