2018년 2학기 컴퓨터공학실험Ⅱ
CSE3016-05반 2주차 예비 보고서

20171665 이 선 호

2018. 09. 14

목 차

Ⅰ HDL(하드웨어 기술 언어)

1. HDL이란?

HDL은 Hardware Description Language의 약자이며, 논리회로의 구조와 동작을 정밀하게 기술하는 데 사용하는 컴퓨터 언어이다. HDL은 크게 두 가지 종류의 시스템을 설계하기 위해 만들어졌는데, 그 중에서 앞서 1주차 실습에서 배운 FPGA 같은 설계 가능 논리 소자)를 프로그램하기 위해서도 사용된다. 논리 회로가 간단한 경우에는 논리 회로도만을 보고 논리 회로를 쉽게 기술하거나 이해할 수 있지만, 기술의 발달로 인해 칩 하나에 트랜지스터 또는 게이트가 수없이 많이 들어가면서 이는 불가능하게 되었다. 그러나 인간이 쉽게 읽을 수 있는 HDL 언어를 사용하면 상대적으로 알아보기 쉽다. 그래서 설계뿐만이 아니라 실물로 구현해서 검증하기 이전에 컴퓨터 시뮬레이션을 이용해 어디에 문제가 있는지도 쉽게 파악할 수 있다.

HDL에는 대표적으로 거의 표준으로 자리 잡은 VHDL과 Verilog가 존재하는데, 이번 실습에서는 특히 Verilog에 초점을 두어 실습을 진행할 것으로 보인다.

2. Verilog 이외의 HDL

HDL에는 Verilog 이외에 VHDL(VHSIC Hardware Description Language)가 존재한다. 두 언어 모두 논리 회로 프로그램을 작성하는데 사용되는 하드웨어 설명 언어이지만 서로 차이가 있다. VHDL은 두 언어 중 오래된 언어이며 Ada와 Pascal을 기반으로 하는 언어의 특성을 이어받지만, Verilog는 상대적으로 최근에 만들어진 언어이며 C 프로그래밍 계열의 언어 코딩 방식을 따른다. 또한 VHDL은 다른 클래스와 함께 변수의 혼합 또는 연산을 허용하지 않지만, Verilog는 이보다 상대적으로 약한 형식을 따른다. 그리고 Verilog는 대소문자를 구분하지만, VHDL은 대소문자를 구분하지 않는 특성이 있다.

Verilog와 VHDL이외에도 여러 HDL이 존재한다. 디지털 회로 설계에 활용되는 HDL에는 ABEL(Advancded Boolean Expression Language), AHDL(Altera가 만든 상용 언어), JHDL(Java에 기반한 언어), MyHDL(Python 기반) 등이 있다. 그리고 아날로그 회로 설계에 활용되는 HDL에는 Verilog-AMS, VHDL-AMS 등이 있다.

Ⅱ Verilog의 역사와 발전 과정

1. Verilog의 역사와 발전 과정

Verilog는 1983년 Gateway Design Automation이라는 회사의 창업자인 프라부 고엘(Prabhu Goel)에 의해서 만들어졌다. 그 당시 초창기 HDL은 미 국방성 주도로 만들어진 VHDL가 시장을 주도하고 있었다. 그러나 Verilog가 VHDL보다 사용하기 상대적으로 쉽다는 특징으로 인해 HDL 시장이 Verilog에 의해 잠식되기 시작했다. 그래서 1987년 Verilog 역시 기존의 VHDL와 마찬가지로 IEEE의 표준 하드웨어 언어로 정립되었다. 이후 1989년에 Cadence라는 회사가 Gateway사를 인수했고 이듬해인 1990년 Cadence사는 Verilog를 공개했다. 이에 따라 Open Verilog International가 설립되면서 Verilog를 적극 지원하게 되었다. 현재 Verilog는 VHDL와 마찬가지로 Accellera라는 단체가 운영하고 관리하고 있다.

Ⅲ Verilog의 기본적인 구조와 문법

1. Verilog의 기본적인 구조와 문법

Verilog는 Ada 기반인 VHDL과는 달리 C언어와 비슷한 문법을 지니고 있다. 그러나 C언어와 완전히 일치하지는 않기 때문에 기본적인 구조와 문법을 익혀두는 것이 필요하다. Verilog는 **대소문자를 구별**하며 begin과 end와 같은 예약어를 제외하면 한 문장을 끝낼 때 반드시 세미콜론(;)으로 끝난다.

Verilog에는 **모듈(module)**이라는 개념이 있다. 모듈을 C언어에 대응하여 설명하자면 프로그램의 일부이며 C언어의 함수보다 좀 더 포괄적인 개념으로 이해할 수 있다. 엄밀하게 말하자면 모듈은 잘 정의된 한 가지 일을 수행하는 프로그램의 논리적인 일부분이며, 여기서 수행하는 일은 어떤 입력에 대해 어떤 출력을 수행하는 것을 뜻한다. 그래서 Verilog에서 일반적인 모듈은 **입출력 단자(포트)**가 존재한다. 여기서 단자는 전자기기 내에서 발생한 전력을 밖으로 내보내거나 또는 밖으로 전력을 공급하기 위한 전류의 출입구 역할을 하며, Verilog에서 이러한 포트는 크게 input(입력 값 포트),

inout(양방향성 데이터 버스), 그리고 output(출력 값 포트)가 있다.

Verilog는 언어구조를 정의하기 위해 사전에 예약된 식별자를 사용하는데, 이를 **키워드(keyword)**라고 한다. 대표적으로 always('@' 뒤에 오는 조건이 참일 때 실행, '@'이 붙지 않으면 항상 실행), assign, begin과 end(한 줄 이상 코드를 실행할 때 중괄호처럼 묶어주는 역할), initial(시뮬레이션을 할 때 순차적으로 신호를 인가할 때 사용) 등이 있다.

식별자(identifier)는 프로그램 언어에서 쓰이는 일반 변수를 의미하며, 객체를 참조할 수 있게 사용자가 부여하는 이름이다. 영문자 알파벳이나 숫자를 사용해야 k며, 특수문자는 언더스코어(_), 달러($) 기호만 사용 가능하다. 단, 식별자를 시작할 때는 반드시 알파벳이나 언더스코어만 써야한다.

수를 표현할 때는 기본적으로 '**<비트 수><진수><숫자>**'로 표현한다. '<비트 수>'에는 10진수로만 써야하며 8비트, 11비트처럼 수의 크기를 의미한다. '<진수>'에는 2진수이면 'b' 또는 'B', 10진수이면 'd', 'D', 그리고 16진수이면 'h', 'H'로 표현한다. 만약 비트 수를 적지 않으면 일반적으로 32비트 수가 되며, 진수를 적지 않으면 기본적으로 10진수로 표현된다.

Verilog에서는 디지털 회로의 기본적인 요소인 **논리 게이트(logic gate)**를 사용한다. 대개 2개의 입력과 하나의 출력으로 되어 있으며, 회로가 처리하는 데이터에 따라 각 단자는 2진수 중 하나인 0 또는 1 상태가 된다. 여기서 0은 전압이 발생하지 않고 1은 일정 전압이 유지된다. 기본적인 논리 게이트에는 논리곱(AND), 논리합(OR), 배타적 논리합(XOR), 부정 논리(NOT), 부정 논리합(NOR), 부정 논리곱(NAND) 등이 있다. 예를 들어, OR 게이트의 경우 입력 값 중 하나라도 1이 존재하면 출력 값이 1이 된다. 이처럼 입력 값들의 기본적인 논리 연산을 통해서 이진수 값의 형태를 출력한다. 논리 게이트 외에도 Verilog에서는 비트 처리 연산자('~', '&', '|' 등)도 사용한다.

Verilog의 기본적인 데이터 종류에는 wire, reg, int, real 등이 존재한다. 여기서 **wire**는 두 개의 다른 물체를 전기적으로 연결하는 전선과 같은 역할을 한다. 그래서 wire를 이용해 모듈의 입력이나 출력을 설정하거나 모듈 간의 입력과 출력을 연결할 때 사용할 수 있다. wire로 선언된 값은 assign문에서만 사용 가능하다. 반면에 **reg**는 논리 상태를 저장할 수 있고, 누군가 이를 바꾸기 전까지 논리 상태를 유지할 수 있다. 그래서 새로운 이벤트가 발생하기 전까지 기존 값을 유지한다. reg로 선언된 값은 always문에서만 사용 가능하다. **parameter**는 모듈 내에서 사용하는 상수를 정의할 수 있으며, 합성 또는 시뮬레이션 시에 값이 치환된다. 자주 사용하는 상수의 경우 수를 직접 사용하는 것보다 parameter를 이용하는 것이 추천된다.

Verilog에서도 여타 프로그램 언어와 마찬가지로 산술 연산, 조건 연산, 조건문(if-else, case), 반복문(while, for, repeat) 등 비슷한 문법을 사용한다.

위에서 배운 바를 정리하여 예시에 적용하면 module문의 형식은 대개 아래와 같이 표현할 수 있다.

```
module 모듈 이름 (포트 목록)
        // 포트
        포트 선언;
        // 기본 선언
        레지스터 선언;
        와이어 선언;
        파라미터 선언;
        // 회로 기능 표현
        연산자
        assign문
        always문
        하위 모듈 호출 등
        ...
endmodule
```