2주차 예비보고서

전공: 물리학과 학년: 3학년 학번: 20191286 이름: 김나현

1. HDL이 무엇인지 조사하고 Verilog이외의 HDL에 대하여 조사하시오.

HDL이란 hardware description language의 약자로, 하드웨어 기술 언어를 의미합니다. HDL은 일반적인 프로그래밍 언어와 비슷하지만 같은 것은 아닙니다. 프로그래밍 언어가 소프트웨어를 만드는 데에 사용된다면, HDL은 디지털 논리 회로의 행동 특성(behavioral characteristics)을 묘사하거나 표현하는 데에 사용됩니다. 예를 들어, 프로세서나 CPU, 마더보드(motherboard)를 디자인하는 데에 HDL을 사용할 수 있습니다. 또한, 시간의 개념이 없이 순차구문(sequential statements)로 구성된 프로그래밍 언어와 달리, HDL은 clock, delay 등 일반적인 프로그래밍 언어에는 없는 개념이 존재하며 순차구문과 함께 병렬구문(concurrent statements)를 사용할 수 있습니다. HDL을 배우면 전체 회로도를 그리는 것보다 훨씬 빠르게 디지털 시스템을 지정할 수 있기 때문에 현대 디지털 디자인에 필수적입니다. 또한, 회로의 재배선 대신 간단한 코드 변경을 통해 수정을 할 수 있기 때문에 빠르게 디버그할 수 있습니다. HDL은 시뮬레이션 단계와 합성 단계에서 모두 사용될 수 있습니다. 이때, 논리 시뮬레이션 단계는 하드웨어로 전환되기 이전, 컴퓨터에서 시스템을 테스트하는 것으로, 시뮬레이터를 통해 물리적인 하드웨어에서는 불가능할 수도 있는 내부 신호 값까지 체크할 수 있습니다.

HDL의 모델링은 크게 세 가지로 나눌 수 있는데, 구조적 모델링(structural modeling), 데이터플로우 모델링(dataflow modeling), 동작적 모델링(behavioral modeling)이 바로 그것입니다. 각각에 대해 간략히 설명하자면, 구조적 모델링은 논리 게이트와 플립플롭 등을 사용한 연결을 표현하는 것이고 이때, 기존에 설계한 회로를 포함한 네트리스트를 사용할 수 있다는 특징이 있습니다. 또한 데이터플로우 모델링은 데이터의 이동을 표현하는 것이고 이때, 연산자를 사용한 연속할당문을 사용하게 됩니다. 마지막으로, 동작적 모델링은 if-else, case, for loop 등과 같은 구문을 사용하는 것입니다.

가장 흔하게 사용되는 HDL에는 Verilog와 VHDL이 있습니다. 이 두 HDL 모두 디지털 회로 생성을 용이하게 하도록 특별히 설계된 반도체인 FPGA와 함께 사용됩니다. HDL은 사용자들로 하여금 단어나 기호를 이용해서 회로를 묘사할 수 있게 해주고 개발 소프트웨어는 원하는 기능을 구현할 수 있도록 사용자가 묘사한 텍스트 설명(textual description)을 FPGA에 load할 수 있는 구성 데이터(configuration data)로 전환해줍니다.

이 중, VHDL은 very high-speed integrated circuit hardware description language의 약자로, 전자 설계 자동화에서 집적 회로(IC) 및 FPGA와 같은 혼합 신호 및 디지털 시스템을 표현하는 데에 사용됩니다. VHDL을 사용하여 논리 회로를 설명하거나 표현하는 텍스트 모델을 작성하면 이 텍스트 모델이 논리 설계의 일부인 경우, 합성 프로그램에 의해 처리됩니다. 그 다음, 시뮬레이션 프로그램을 통합하여 논리 설계를 테스트합니다. 이 단계에서 시뮬레이션 모델을 활용하여 설계에 인터페이스하는 논리 회로를 특성화합니다. 이때, 이러한 시뮬레이션 모델 모음을 테스트 벤치(test bench)라고도 합니다. VHDL은 nor과 nand를 포함하는 광범위한 Boolean 연산자이 가능하다는 특징이 있습니다. 또한 VHDL은 합성 도구(synthesis tool)가 디자인을 실제 게이트 및 와이어, 즉 하드웨어로 변환하는 합성 과정에 앞서 필수 시스템의 동작을 미리 검증하고 모델링할 수 있다는 점입니다. 또 다른 VHDL의 장점은 동시 시스템(concurrent system)에 대한 설명을 제공한다는 점인데, VHDL은 C나 어셈블리 코드와 같은 절차적 컴퓨팅 언어(procedural computing language)와 다르게 모든 명령문이 실행과 동시에 고려되는 데이터 흐름 언어(dataflow language)이기 때문입니다. VHDL 프로젝트의 multipurpose 특성으로 인해, 한 번 생성되면 해당 calculation block은 다른 다양한 프로젝트에도 사용될 수 있습니다.

1. Verilog의 역사와 발전 과정을 조사하시오.

Verilog HDL은 Phil Moorby와 Prabhu Goel에 의해 1983년 말, 1984년 초 사이에 발명되었습니다. Verilog는 Gateway Design Automation Inc의 독점 하드웨어 모델링 언어로 사용되었습니다. 하지만 그 당시 Verilog는 표준화되지 않았기 때문에 1984년부터 1990년 사이, 수정을 거쳤습니다. 1990년에 Gateway Design Automation Inc가 전자 설계 자동화(EDA) 산업에서 전자 설계 시술 및 엔지니어링 서비스의 최대 공급 업체 중 하나인 Cadence Design System에 인수되면서 Cadence가 Verilog를 독점하게 되었고 초기에는 시뮬레이션을 설명하는 용도로만 사용되었지만 Verilog의 활용도가 점차 다양해지면서 해당 언어를 이용하여 gate 등과 같은 물리적인 실현 가능한 구조들을 자동 합성할 수 있게 되었습니다. Cadence는 Verilog를 공개 표준화하기 위해 OVI(Open Verilog Internation) 조직의 공개 도메인으로 이전하였고 이후 IEEE에 제출되어 흔히 Verilog-95라고 불리는 IEEE 표준 1364-1995가 되었습니다. 이후에도 Verilog 표준에서 발견되는 결함을 보완하여 IEEE에 다시 제출하여 Verilog-2001으로 알려진 IEEE 표준 1364-2001이 되었습니다. 2005년에는 Verilog-2005가 게시되었고, 이 확장판의 상위 집합인 SystemVerilog가 발표되었습니다. 2009년에는 SystemVerilog와 Verilog 언어 표준이 오늘날 IC 설계와 검증에 널리 사용되는 언어 중 하나인 SystemVerilog 2009(IEEE 표준 1800-2009)으로 병합되었습니다.

1. Verilog의 기본적인 구조와 문법에 관하여 조사하시오.

Verilog의 기본적인 설계 단위는 모듈(module)로, verilog는 모듈의 계층 구성으로 이루어져 있습니다. 모듈은 설계 계층을 캡슐화하고 선언된 입력(input), 출력(output) 및 양방향(inout, bi-directional) 포트(단자쌍) 세트를 통해 다른 모듈과 통신합니다. 내부적으로 모듈은 net/variable 선언, 동시 및 순차 명령문 블록, 다른 모듈의 인스턴스의 모든 조합을 포함할 수 있습니다. 순차 명령문은 시작/종료 블록 내에 배치되고 블록 내에서 순차적으로 실행됩니다. 그러나 블록 자체는 동시에(concurrently) 실행됩니다.

[[1]](#endnote-1)Verilog는 머리부, 선언부, 몸체부, 크게 세 부분으로 나뉘고, 머리부는 키워드 module로 시작하여 모듈의 이름, 포트 목록 등을 적은 후, 세미 콜론(;)으로 끝납니다. 선언부는 포트 목록에 나열된 포트들의 방향, 비트 폭, reg 및 wire 선언, parameter 선언 등 모듈에서 필요로 하는 것들을 선언하고, 몸체부는 회로의 기능, 동작, 구조 등을 표현하는 다양한 Verilog 구문들로 구성됩니다. 블록의 끝을 나타낼 때는 end를 적고, 다 끝나면 endmodule을 적어줍니다.

다음은 Verilog code를 작성할 때에 유의해야 할 문법들을 간략히 다뤄보겠습니다.

-여백

빈칸, 탭, 줄바꿈 등이 해당되고, 어휘 토큰을 분리하기 위해 의미있게 사용된 여백를 제외하고는 무시합니다. 다만, 공백과 탭은 문자열에서 유의미하게 취급됩니다.

-주석

c언어와 동일하게, 단일 라인 주석문은 \(이하 \는 모두 back slash을 의미함.)\로 시작되어 해당 라인 끝까지 주석처리가 가능하고 블록 주석문은 \\*~\*\으로 표시할 수 있습니다. 이렇게 주석 처리된 문장은 HDL 소스코드의 설명을 위해 사용되며 컴파일 과정에서 무시됩니다.

-연산자

c언어와 비슷하게 단항 연산자, 2항 연산자, 3항 연산자를 사용합니다.

-식별자

객체에 고유 이름을 지정하기 위해 사용되고, 대소문자를 구별하여 인식합니다. 단순 식별자는 일련의 문자, 숫자, 기호 $, 밑줄 등으로 구성되고 확장 식별자는 \로 시작되며, 여백 등으로 끝납니다.

-키워드

Verilog 구성 요소를 정의하기 위해 미리 정의된 식별자로, 확장문자가 포함된 키워드는 키워드로 인식되지 않는다는 것에 유의해야 합니다.

-수 표현

형식: [size\_constant]’<base\_format><number\_value>

[size\_constant]는 값의 비트 크기를 나타내는 상수로, 0이 아닌 10진수가 사용되며 생략될 수도 있습니다. 비트 크기가 지정되지 않은 수는 32비트로 표현되며 음수를 표현하고 싶으면 [size\_constant] 앞에 음수 부호를 붙이면 됩니다. [base\_format]은 d(10진수), h(16진수), b(2진수), o(8진수), 이 네 가지 중 하나로 표현되고 아무것도 쓰지 않으면 자동으로 10진수로 취급됩니다. [number\_value]는 unsigned 숫자를 사용하여 값을 표현하는 것입니다.

[[2]](#endnote-2)이들을 이용하여 Verilog 구문 예시 몇 가지를 보이고, 설명해보도록 하겠습니다.

구문의 맨 위에는 timescale을 적기 위해 ‘timescale <시간단위>/<정밀도>를 적는데, 이 문장에서 시간 단위는 파일 내의 모든 시간 단위가 해당 시간 단위로 변경되는 것이고, 정밀도는 주어진 시간 단위로 구성할 수 있는 가장 작은 지연을 나타내는 것입니다.

Assign 구문은 입력 피연산자의 값에 event가 발생할 때마다 우변의 식이 평가되고 그 결과 값이 할당문 주변의 net를 구동하는 하드웨어적인 특성을 갖습니다. Deassign 구문은 변수에 대한 assign 구문의 영향을 제거시킬 때 사용합니다.

Always 문은 시뮬레이션이 실행되는 동안 무한히 반복적으로 실행되며, 타이밍 제어와 연관된 표현에 유용하게 사용됩니다. @(sensitivity\_list)는 always 문의 실행을 제어하며 sensitivity\_list에 나열된 신호 중 하나 이상에 event가 발생하면 always 내부에 있는 begin-end 블록이 실행됩니다.

시뮬레이션이 진행되는 동안 무한히 반복되는 always 문과 다르게, initial 구문은 시뮬레이션이 실행되는 동안 한 번만 실행되고, initial 구문 안에 있는 begin-end 블록은 절차형 문장들로 구성되어 나열된 순서대로 차례차례 실행됩니다.

1. 해당 부분은 컴퓨터설계실험2 ppt 4 page에서 발췌하였습니다. [↑](#endnote-ref-1)
2. 해당 부분은 컴퓨터설계실험2 ppt 10-13 page에서 발췌하였습니다. [↑](#endnote-ref-2)