2주차 예비보고서

전공: 컴퓨터공학과 학년: 3학년 학번: 20191599 이름: 송경호

**1.**

**1-1) HDL의 정의**

HDL(Hardware Description Language)은 디지털 하드웨어 설계를 위한 특수한 프로그래밍 언어이다. 즉 HDL을 통해 전자 회로 및 디지털 시스템 설계, 모델링, 시뮬레이션을 할 수 있다. 이때, HDL은 두 가지 주요 활용분야가 있는데, 첫번째가 직접회로 디자인 및 시뮬레이션이고 두번째가 FPGA 및 PLD(Programmable Logic Device) 프로그래밍이다. 먼저 직접회로 디자인의 경우 HDL은 특정한 직접 회로를 설계하기위 해 모델을 제공한다. 이는 회로의 실제 구현 전에 해당 회로의 동작을 테스트하는데 사용할 수 있으며 이를 통해 오류를 식별하고 수정할 수 있다. 다음으로 PLD 프로그래밍의 경우 FPGA를 예로 들겠다. FPGA에 HDL 코드를 넣으면 FPGA가 해당 코드를 실행하며 코드에 따른 논리 회로 및 동작을 구현한다. 1주차\_예비보고서에서 FPGA는 설명했듯이 다양한 기능의 구현이 가능하며 이를 통해 하드웨어의 재사용성을 높이고 수정이 용이하게 한다.

HDL의 설계과정은 설계 캡슐화 -> 논리 시뮬레이션 -> 논리 합성 -> 타이밍 검증 -> 결함 시뮬레이션의 순서로 이어진다. 설계 캡슐화 단계는 HDL 언어를 사용하여 하드웨어의 기능을 작성하는 단계이다. 논리 시뮬레이션 단계는 HDL로 작성된 설계를 시뮬레이션을 통해 확인하고 검증하는 단계이다. 시간 순서와 신호 파형을 통해 시각화된 동작이 올바른지 판단한다. 다음으로 논리 합성 단계에서는 HDL 표현에 기반한 구성 요소들을 최적화하고 회로 구성 정보를 뜻하는 넷리스트를 생성하는 단계이다. 이 가정을 통해 하드웨어의 구조가 데이터베이스로 생성된다. 이후 타이밍 검증과정은 합성된 결과물이 규정된 속도 내에서 동작이 가능한지를 검증하는 단계이다. 게이트 전파 지연과 같은 요인에 의해 의도와 다른 동작이 발생할 수 있기 때문에 이 검증 단계를 통해 올바른 타이밍 동작을 보장한다. 마지막 결함 시뮬레이션 단계에서는 결함이 있는 회로와 무결함 회로 사이의 차이를 확인하기 위한 테스트 패턴을 사용하야 검증을 수행하는 단계이다.

결론적으로 HDL을 사용하면 하드웨어 설계와 검증이 효율적으로 수행되며, 동시에 디지털 시스템의 모델링 및 분석이 가능해진다.

**1-2) Verilog 이외의 HDL**

HDL에는 Verilog 이외에도 여러 다른 언어가 존재한다. 그 중에서 가장 대표적인 예시인 VHDL과 ABEL에 대해 설명하겠다.

먼저 VHDL은 디지털 회로 및 혼합 신호를 표현하는 하드웨어 기술 언어로 VHSIC Hardware Description Language의 약자이다. VHDL은 회로 설계에서 상위 동작 레벨부터 하위 게이티 레벨까지 하드웨어를 기술하고 설계하는 데 사용된다. 또한 VHDL은 원래 특수용도 직접 회로인 ASIC의 문서화를 사용하기 위해 개발되었으나 이후 회로 디자인에서 시뮬레이션을 수행하는 데 사용되었다. 현재는 디지털 회로 설계 및 검증, 구현 등 다양한 용도로 사용되고 있다. VHDL의 장점은 EDA 소프트웨어나 ASIC 라이브러리를 이용하지 않더라도 설계가 가능하다는 점이다. 또한 내부 회로부터 시스템의 설계까지 다양한 단계에서 활용될 수 있다는 점도 있다. 설계 과정은 Design Entry -> Fuction Simulation -> Synthesis -> Timing Simulation의 단계로 구성된다. Design Entry는 VHDL의 동작을 정의하는 단계이며, Function Simulation은 동작을 검증하는 단계이다. 다음으로 Synthesis 단계에서 논리 게이트로 변환되고 Timing Simulation 단계에서 마지막으로 타이밍을 검증한다. VHDL 설계의 기본 구성 요소에는 package. entity, architecture, configuration이 있으며, 이 구성 요소들을 올바르게 활용하고 조합하여 설계를 구출할 수 있다.

ABEL은 Advanced Boolean Expression Language의 약자로 PLD를 프로그래밍하기 위한 언어이다. 1983년 데이터 I/O에서 개발되었으며 ABEL은 논리 함수를 표현하고 하드웨어 프로그래머에게 다운로드 가능한 테스트 벡터를 생성하는 데 사용된다.

HDL에는 다양한 설계 도구가 있으며, 각 언어는 특정 응용 분야의 요구 사항에 알맞게 선택하는 것이 중요하다.

**2.**

Verilog은 처음에 Phil Moorby가 모의 시험용 언어로 개발했다. 이 후 1983년에 Prabhu Goel이 Gateway Design Automation이라는 회사를 설립하여 verilog와 시뮬레이터를 공개하였다.

이후 1985년에 새로운 버전인 Verilog-XL과 그 시뮬레이터가 공개되었다. 1987년에 VHDL 언어가 IEEE1076 표준 HDL로 공개되었다. 1989년에는 Cadence라는 회사가 Gateway Design Automation사를 인수하였고 이후 1990년에 Verilog HDL을 공개한다. 이를 통해 Open Verilog International이 설립되었으며 현재까지 계속해서 Verilog의 사용자 수가 늘어나고 있다.

1995년에는 Verilog 언어가 1364-1995로 공개되었으며 2001년에 개정된 이후로 현재까지 IEEE-1364 표준을 지키고 있다. 또한 현재 VHDL과 Verilog는 Accellera라는 단체가 유지 관리를 맡고 있다.

**3.**

**3-1) Verilog의 기본적인 구조**

Verilog의 기본 단위는 모듈이다. 키워드 module에서부터 키워드 endmodule을 통해 하나의 module을 정의한다. module은 머리부, 선언부, 몸체부 세 부분으로 구성된다. 머리부는 키워드 module로 시작하여 모듈이름, 포트 목록을 나열하고 ‘;’로 종료된다. 이 때 모듈의 이름은 case sensitive하다. 선언부는 포트 목록에 나열된 포트들의 방향, 비트 폭, reg, wire parameter 등 모듈에 필요한 것들을 선언하는 부분이다. 마지막으로 몸체부에서는 회로의 기능, 동작, 구조 등을 표현하는 다양한 Verilog 구문들이 작성된다. Verilog 구문들의 경우 논리합성용, 시뮬레이션용, 라이브러리 설계용 구분으로 구분되며 논리합성용 구문은 if-else, for문 등 대부분의 논리합성 툴에서 게이트 수준 합성을 지원하는 구문들과 동일하다. 시뮬레이션용 구문은 테스트벤치의 작성에 사용되는데 이때 논리합성은 지원되지 않는다. 라이브러리 설계용 구문은 논리합성에 적용되는 셀 라이브러리 설계에 사용되며 셀의 기능, 지연, 핀 정보등을 정의하는데 사용된다.

**3-2) Verilog의 문법**

**i) Verilog Data Type**

i-i) Register : 추상적 저장 장치, 즉 값을 저장하는 장치

* reg : 절차형 할당문(always, initial)에 의해 값을 받는 객체
* integer : 정수형 변수
* time, realtime : 시간형 변수
* real : 실수형 변수

i-ii) Net : 디바이스의 물리적인 연결 부분

* wire : 변수들이 모듈 내에서 어떻게 연결됐는 지에 관한 변수
* tri : 선 연결시 사용

**ii) Constant**

비트 수 제한이 있는 reg의 값을 선언할 때, [bit수][입력 형식][입력 값]을 따른다.

ii-i) size를 정하지 않은 경우

214; – 정수 214, h32; - 16진수 32, o324; - 8진수 324

ii-ii) size를 정한 경우

처음에 비트를 선언한다.

4’b1111; -> 이진수 1111 (4bit), 4’hf; -> 16진수 f (4bit)

ii-iii) 부호가 있는 수

가장 앞에 부호를 선언하며 음의 값은 2의 보수 처리된다.

**iii) Operator**

|  |  |
| --- | --- |
| 논리 연산자 | &&(and), ||(or), !(not) |
| 비트 연산자 | ~(not), &(and), |(or), ^(xor), ~^(xnor) |
| 단항 비트 연산자 | &(and), ~&(nand), |(or), ~|(nor), ^(xor), ~^(xnor) |
| 시프트 연산자 | >>(우측 shift), <<(좌측 shift) |
| 기타 | {}(결합), {{}}(중복), ?:(3항 조건 연산자) |

**iv) Statement**

iv-i) Always문

|  |
| --- |
| always @ (이벤트∙신호 or … or 이벤트∙신호)  begin  assign  if, case  for, function, task call  end |

@ 옆에 정의된 값들 중 하나라도 변화가 발생하면 쓰여진 문장의 순서대로 처리되며 end에 도달하면 다음 값의 변경까지 대기한다.

iv-ii) Initial문

|  |
| --- |
| Initial  begin  assign  조건처리  If, case  end |

시뮬레이션 시 사용되며 순차적으로 한번 처리한다.

iv-iii) Timescale

timescale <시간단위> / <정밀도> 형식으로 선언된다. <시간단위>는 파일 내의 시간 단위를 선언하는 것이며 <정밀도>는 주어진 시간 단위로 구성 할 수 있는 가장 작은 지연을 나타낸다.

iv-iv) Assign & Deassign

assign구문은 입력 피연산자의 값에 변화가 발생할 때 마다 우변의 식이 평가되고 그 값이 할당문 주변의 net을 구동하게 한다. 다시 말해, net 변수에 특정 논리 값을 저장하는데 사용한다.

|  |
| --- |
| wire mynet = enable&data; |

iv-v) Block문

|  |
| --- |
| begin : 블록 이름  reg;  integer;  parameter;  …문장…  end |

이름이 없는 블록 내에선 지역 변수 선언이 불가능하다.