**컴퓨터공학 설계 및 실험Ⅱ**

2주차 예비보고서

서강대학교 공학부 컴퓨터공학 전공

20171646 박태윤

**1. HDL이 무엇인지 조사하고 Verilog이외의 HDL에 대하여 조사하시오.**

Hardware Description Language의 약자이기도 한 HDL은 컴퓨터 언어 중 하나이다. 그러나 C나 java 등과 같은 일반적인 프로그래밍 언어와 비교했을 때 컴파일 과정에서 차이점을 나타낸다. 또한 프로그래밍 언어는 순차구문으로 구성되어 시간적인 개념이 없는 반면에, HDL은 시간에 대한 개념을 가지고 있다는 특징이 있다. 이 언어는 전자 회로 설계, 검증 등 주로 하드웨어 분야에서 사용이 된다.

Verilog는 HDL의 한 종류이며, 다른 HDL로는 VHDL이 있다. VHDL은 주로 디지털 회로 설계에 사용이 된다. C언어와 상당히 유사한 Verilog와 달리 VHDL은 Ada언어와 Pascal언어를 기반으로 두고 있다. 상대적으로 VHDL이 Verilog에 비해 엄격한 문법을 가지고 있다.

**2. Verilog의 역사와 발전 과정을 조사하시오.**

Verillog는 1983년 후반에서 1984년 초에 개발되었고 Cadence Design Systems에 인수되었다. 이후 IEEE(전기 전자 기술자 협회)에 제출되어 IEEE표준 1364-1995로 만들어졌고 이 때 Verilog는 통칭 Verilog-95라고 불린다. 그리고 기존 IEEE표준이였던 Verilog-95에서 결함을 발견해 확장을 한 뒤 다시 IEEE에 제출되고, 이는 Verilog-2001이라 불리는 IEEE표준 1364-2001이 된다. 이후 수정 과정을 거치고 IEEE에 제출을 했으며 2005년에 System Verilog가 IEEE표준으로 만들어졌다.

**3. Verilog의 기본적인 구조와 문법에 관하여 조사하시오.**

Verilog의 구조는 크게 머리부, 선언부, 몸체부로 볼 수 있다.

|  |  |
| --- | --- |
| 머리부 | ‘module module\_name (port\_list);’와 같이 module로 시작하여 모듈명과 포트목록들을 지정하는 부분이다. |
| 선언부 | 포트목록에 나열된 포트들의 방향 비트 폭, reg 선언, wire 선언 등 모듈에서 필요한 것들을 선언하는 부분이다. |
| 몸체부 | 논리합성용 구문, 시뮬레이션용 구문, 라이브러리 설계용 구문으로 구분이 되는 회로의 기능, 동작 등을 표현하는 구문들로 구성이 되는 부분이다. |

|  |  |
| --- | --- |
| 머리부 | Module module\_name (port\_list); |
| 선언부 | port 선언  reg 선언  wire 선언  parameter 선언 |
| 몸체부 | 하위모듈 인스턴스  Gate primitive  always 문, initial 문  assign 문  function, task 정의  function, task 호출  end module |

Verilog에서 쓰이는 변수들의 data type들은 다음과 같다.

|  |  |
| --- | --- |
| Register : 추상적인 저장 장치 | reg : always, initial과 같은 절차형 할당문에 의해 값을 받는 객체  integer : 정수형 변수  time, realtime : 시간형 변수  real : 실수형 변수 |
| Net : 디바이스의 물리적인 연결 | wire : 변수들이 모듈내에서 어떻게 연결되어 있는지 나타내 주는 변수  tri : 선을 서로 연결할 때 사용하여 wire와 달리 3상태(tri-state)net에 사용된다. |

Verilog에서 상수를 선언하는 다양한 방식이 있는데 이는 다음과 같다.

|  |  |  |
| --- | --- | --- |
| 구 분 | 표현방식 | 의 미 |
| 일반적인 표현 | 4’b1111  12’habc  16’d255 | 4bit 2진수 1111  12bit 16진수 abc  16bit 10진수 255 |
| 수의 크기가  지정되지 않은 경우 | 214  ‘ha2  ‘o324 | 32bit 10진수 214  32bit 16진수 a2  32bit 8진수 324 |
| 음수 표현 | -6’d3 | 3와 2의 보수로써 음수 |

위의 예시에서 볼 수 있듯이, 수는 기본적으로 (크기)(n진수)(숫자)로 나타내어진다.

Verilog는 또한 프로그래밍 언어처럼 다양한 연산자(Operator)들을 가지고 있다.

|  |  |  |  |
| --- | --- | --- | --- |
| 형 태 | 기 호 | 연산자역할 | 형 식 |
| 산 술  연산자 | +  -  \*  /  % | 덧 셈  뺄 셈  곱 셈  나눗셈  나머지 | A + B  A – B  A \* B  A / B  A % B |
| 관 계  연산자 | >  <  >=  <=  ==  != | 크 다  작 다  크거나 같다  작거나 같다  같 다  다르다 | A > B  A < B  A >= B  A <= B  A == B  A != B |
| 논 리  연산자 | &&  ||  !  ~  &  |  ^  ^~ or ~^ | and  or  not  비트단위 not  비트단위 and  비트단위 or  비트단위 xor  비트단위 xnor | A && B  A || B  !A  ~A  A & B  A | B  A ^ B  A ~^ B |
| Shift  연산자 | >>  << | 오른쪽으로 이동  왼쪽으로 이동 | 0110 >> 1(결과:0011)  1100 << 1(결과:1000) |
| 결합 & 반복  연산자 | {}  {{}} | 결합 연산자  반복 연산자 | { A,B }  { {A} } |

Verilog는 여러 구문 또한 가지고 있다. C언어와 같은 프로그래밍 언어에서도 쓰이는 If, Case, For 등의 구문들을 가지고 있으며 그 외 timescale, assign, always, initial, (Non) Blocking등의 구문 또한 가지고 있다. timescale, assign, always, initial, (Non) Blocking 구문을 살펴보면 다음과 같다.

-timescale

timescale 구문의 기본 형태는 ‘timescale <시간 단위> / <정밀도> 이다.

‘timescale 1ns / 10ps 라는 구문을 살펴보면

‘/’ 앞에 있는 ‘1ns’가 시간 단위이고 이는 파일 내의 모든 시간단위를 ‘1ns’로

지정하겠다는 의미이다.

‘/’ 뒤에 있는 ‘10ps’는 정밀도이며 시간단위와 관련하여 사용할 수 있는 소수

점의 허용범위를 나타낸다.

-assign & deassign

assign 구문은 net 변수에 특정 값을 지정하는데 사용되며 deassign 구문은 이러한 assign구문의 변수에 대한 영향을 제거시킬 때 사용된다.

ex)

assign wire1 = reg1

* 단순 선 연결.

assign wire2 = (pin1)?reg[0] : reg2[1];

* 2 to 1 mux의 역할. 괄호 안에 있는 것이 참이면 “:” 앞쪽에 있는 것이 거짓이면 “:” 오른쪽에 있는 것으로 연결한다.

assign wire2 = {reg1,reg2[0]};

* 서로 다른 두개의 신호를 하나의 버스로 집어 넣어 줌.

-always

always구문의 형태는 다음과 같다.

always @ (sensitivity list)

begin

Blocking or Nonblocking statements

end

시뮬레이션이 실행되는 동안 반복적으로 실행된다. @ (sensitivitiy list)는 always문의 실행을 제어하는 역할을 하며 Sensitivity list의 신호들 중 하나 이상에 변화(event)가 생긴다면, begin ~ end 안의 구문들이 실행이 된다.

-initial

Initial구문의 형태는 다음과 같다.

initial

begin

Blocking or Nonblocking statements

end

시뮬레이션이 진행되는 동안 무한히 반복되는 always구문과는 달리 initial구문은 시뮬레이션이 실행되는 동안 딱 한 번 begin ~ end 안의 문장들을 나열된 순서대로 실행한다.

-(Non) Blocking

Blocking 구문은 always or initial 구문 안의 begin ~ end 까지를 일반 프로그래밍 언어처럼 문장이 나열된 순서대로 계산과 동시에 저장이 이루어지는 것을 의미한다. 반면 Non blocking 구문은 begin ~ end 안에서 Blocking 구문과는 다르게 모든 계산을 수행한 후 한꺼번에 저장을 한다. 간단한 예시를 보면

|  |  |
| --- | --- |
| Blocking | - symbol : ‘=’  ex)  a=1, b=2, c=3이라고 선언되어있고  (begin ~ end)문 안에서  a = b;  b = c;  c = a;  라고 되어있으면  결과적으로 a = 2, b = 3, c = 2가 된다. |
| Non blocking | - symbol : ‘<=’  ex)  a=1, b=2, c=3이라고 선언되어있고  (begin ~ end)문 안에서  a <= b;  b <= c;  c <= a;  라고 되어있으면  결과적으로 a = 2, b = 3, c = 1이 된다. |

이렇게 같은 변수와 수식이라도 Blocking문과 Non blocking문의 결과값에서 차이가 발생하는 것을 볼 수 있다.