KAERI/AR-836/2009

원자력발전소 계측제어계통에 적용을 위한 CPLD/FPGA 기술 현황분석

Survey of the CPLD/FPGA Technology for Application to NPP Digital I&C System

2009.08

한 국 원 자 력 연 구 원

제 출 문

한국원자력연구원장 귀하

본 보고서를 2009년도 “계측제어 안전계통 필수기술 개발” 과제 및 “계측제어 안전계통 독립검증” 과제의 기술현황 분석 보고서로 제출합니다

2009.08.

주 저 자: 최종균(계측제어인간공학연구부)

공 저 자: 권기춘(계측제어인간공학연구부)
이동영(계측제어인간공학연구부)
이장수(계측제어인간공학연구부)
이영준(계측제어인간공학연구부)
손광섭(계측제어인간공학연구부)
박기용(계측제어인간공학연구부)
이현철(계측제어인간공학연구부)
박원만(계측제어인간공학연구부)
장통일(계측제어인간공학연구부)
김동훈(계측제어인간공학연구부)
오인석(계측제어인간공학연구부)
이정운(계측제어인간공학연구부)
이용희(계측제어인간공학연구부)
허 섭(계측제어인간공학연구부)
김정택(계측제어인간공학연구부)
황인구(계측제어인간공학연구부)
박재창(계측제어인간공학연구부)
이철권(계측제어인간공학연구부)
김장열(계측제어인간공학연구부)
김창회(계측제어인간공학연구부)
천세우(계측제어인간공학연구부)

요 약 문

원자력 산업의 경우, 기존의 원자력발전소에 사용되고 있는 아날로그기술 기반의 계측제어 시스템은 노후화로 인해 보수 및 교체가 요구되고 있다. 그러나 기술지원 및 부품조달의 어려움으로 인해 아날로그 기기의 계속 사용에 어려움이 있으며, 기존 원자력발전소의 유지 및 보수에 어려움을 겪고 있다. 이러한 이유로 인해, 국제적으로 기존의 아날로그기술 기반의 계측제어 시스템을 디지털기술 기반의 계측제어 시스템으로 교체하고 있으며, 신규 원자력발전소에도 디지털기술 기반의 계측제어 시스템을 채택하고 있다.

현재 디지털 계측제어 시스템은 마이크로프로세서 기반의 하드웨어에 소프트웨어를 탑재하는 방식과 프로그래머블 논리 소자 기반의 설계 방식으로 개발되고 있다. 상대적으로 프로그래머블 논리 소자 기반의 계측제어 시스템은 개발이 용이하고, 응답시간이 빠르며, 결정론적 특성의 구현이 용이하고, 기기 및 부품 단종에 유연하게 대처할 수 있다. 또한, 사이버 보안에 대한 문제도 거의 존재하지 않는다. 이러한 이유로 인해, 국내외적으로 프로그래머블 논리 소자를 이용한 디지털 계측제어 시스템의 개발에 관심이 높아지고 있다.

본 보고서에서는 프로그래머블 논리 소자를 이용한 계측제어시스템의 개발을 위하여 프로그래머블 논리 소자의 종류, 구조, 제조 방법 및 특성을 정리하였고, 국내외의 프로그래머블 논리소자 기반의 계측제어시스템의 개발동향을 정리하였다. 그리고 시스템 측면에서 프로그래머블 논리 소자의 설계지침 및 코딩 지침을 정리하였다.

목 차

그 림 목 차

표 목 차

제 1 장 서론

제 1 절 FPGA 기술의 도입

집적회로 기술이 발달하면서 더 많은 컴포넌트들이 하나의 칩에 들어갈 수 있게 됨에 따라, 디지털 시스템의 복잡도는 계속 증가하여 왔다. 처음에는 소수의 트랜지스터를 하나의 칩에 구현하는 소규모 집적회로(Small Scale Integration: SSI)가 개발되었다. 이후, 많은 기술적 발전으로 하나의 칩에 수백 개의 게이트들이 구현된 중규모 집적회로(Medium Scale Integration: MSI), 수천 개의 게이트들이 구현된 대규모 집적회로(Large Scale Integration: LSI), 수만 개의 게이트를 갖는 대대규모 집적회로(Very Large Scale Integration: VLSI) 그리고 수십만 개 이상의 게이트를 하나의 칩에 구현하는 초대규모 집적회로(Ultra Large Scale Integration: ULSI)가 개발되어 왔다.

이러한, 집적기술이 발달되어 디지털 시스템이 복잡해짐에 따라, 설계자가 직접 손으로 회로를 설계하던 방식에서 컴퓨터 지원 설계(Computer Aided Design: CAD) 도구를 사용하여 회로를 설계하기 시작하였다. 설계자는 더 이상 디지털 회로를 설계하기 위하여 게이트들을 손으로 직접 배치할 필요 없이, VHDL(VHSIC Hardware Description Language)이나 Verilog HDL 같은 하드웨어 기술 언어(Hardware Description Language: HDL)를 이용하여 설계를 수행하며, 이렇게 설계된 설계 결과물은 CAD 에 의해 자동으로 합성(Synthesize)되어 칩에 구현될 수 있게 되었다.

본 과제는 원전 계측제어 시스템의 개발에 이러한 고 직접도의 프로그래머블 논리 소자의 도입 가능을 파악하기 위하여 프로그래머블 논리 소자의 특성, 취약성을 조사하였고, 현재 원자력 분야에서 프로그래머블 논리 소자의 적용현황(관련 규제기준, 관련 기술기준, 개발 현황 등)을 조사하였다. 또한 프로그래머블 논리 소자를 이용하여 시스템을 개발 시에 적용가능한 설계지침 및 코딩 지침을 개발하였다.

제 2 절 FPGA 개발 공정

원자력발전소에 적용을 목적으로 개발되는 시스템에는 높은 안전성 및 신뢰성이 요구된다. 특히, 안전-필수(Safety-Critical) 등급의 시스템은 가장 높은 안전성을 요구받고 있다. 따라서 FPGA 기반의 디지털 시스템이 안전-필수 등급의 시스템에 적용되기 위해서는 시스템이 안전하고 신뢰성이 있게 개발해야 하고 이를 입증할 수 있어야 한다. 특히, FPGA 기반 시스템의 경우, 시스템의 모든 주요 기능들이 FPGA에 구현되기 때문에, FPGA에 구

현되는 로직의 안전성 및 신뢰성이 시스템의 안전성 및 신뢰성을 결정한다고 볼 수 있다.

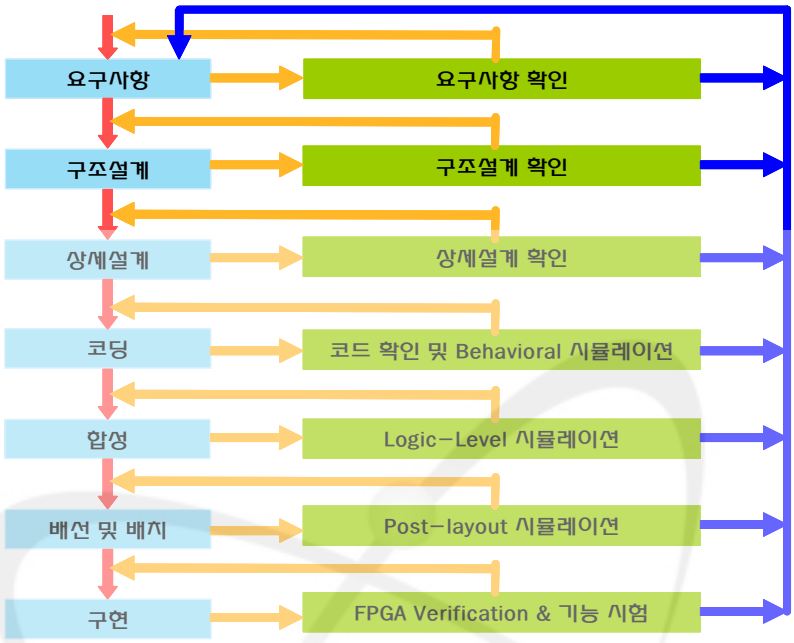


그림 1.1 FPGA 개발 및 검증 절차

그림 1.1은 FPGA의 개발 및 검증 절차를 보여준다. 개발의 첫 번째 단계는 FPGA에 요구되는 기능을 나타내는 요구사항을 체계적인 문서형태로 기술하는 단계이다. 요구사항을 명확히 정의한 후 설계사양을 정의한다. 다음 단계는 블록도 또는 알고리즘 수준의 개념적인 수준의 설계를 체계화하는 구조설계 및 상세설계 단계이다. 구조설계 및 상세설계가 끝난 후, 이를 바탕으로 HDL를 이용하여 RTL(Register Transfer Level) 수준의 코드를 작성하고, 합성한다. 논리 합성도구는 RTL 수준의 코드를 게이트 수준의 넷-리스트(netlist)로 변환한다. 게이트 수준의 넷-리스트는 게이트와 그들 간의 연결이라는 관점에서의 회로를 말한다. 그 게이트 수준의 넷-리스트는 레이아웃을 생성하는 자동 배치 도구와 배선 도구의 입력이 된다. 레이아웃은 구현 도구에 의해 자동으로 칩 위해 회로가 제작된다. 일반적으로 RTL 코드 작성 이전의 단계는 설계자에 의해 수동으로 개발이 이루어지며, RTL 코드 작성 이후의 단계는 소프트웨어 도구에 의해 자동으로 개발이 수행된다.

따라서 설계자에 의해 수동으로 수행되는 최종단계는 RTL 코드 작성 단계이며, RTL 코드가 요구사항을 만족하도록 작성되었는지가 FPGA의 품질(정확성, 신뢰성 및 안전성 등)을 결정한다.

제 3 절 보고서의 구성

제 1장에서는 본 보고서의 서론의 본 보고서의 작성 목적 및 FPGA 개발공정에 대하여 설명하였다.

제 2장에서는 현재 상용으로 제공되고 있는 프로그래머블 논리 소자의 개요 및 구조를 설명하였다. 또한 프로그래머블 논리 소자의 구성기술에 따른 특성을 기술하였고, 상용으로 시장에서 구매할 수 있는 프로그래머블 논리 소자의 종류 및 이를 제공하는 공급자를 설명하였다. 그리고 프로그래머블 논리 소자의 전기적 및 물리적 취약성을 분석하였다.

제 3장에서는 현재 프로그래머블 논리 소자의 원자력분야에 적용에 관련된 규제지침 및 기술기술을 설명하였다. 그리고, 국내외적으로 프로그래머블 논리 소자를 이용하여 원자력발전소의 계측제어 시스템 개발현황을 설명하였다. 또한, 현재 프로그래머블 논리 소자를 이용한 시스템 개발에 사용되는 개발공정을 설명하였다.

제 4장에서는 프로그래머블 논리 소자를 이용한 시스템 개발시에 적용 가능한 설계지침을 설명하였다. 설계 지침에는 프로그래머블 논리 소자의 외부 핀(사용하지 않는 핀, 모드 핀, JTEG 핀)에 대한 처리 방법, 입력 및 출력 신호 설계, 클럭 신호 설계, 리셋 신호 설계, 전원 설계, 타이밍 분석 등에 대한 지침을 포함한다.

제 5장에서는 프로그래머블 논리 소자 설계를 위한 코딩 지침을 설명하였다. 코딩 지침에는 기능 정확성, 타이밍 정확성, 합성성, 유지 보수성 측면에서, 이러한 특성을 만족하기 위한 코딩 방법을 제시하고 있다.

제 2 장 FPGA 소개

제 1 절 FPGA 개요

설계자는 디지털 회로를 구현하기 위해 PAL(Programmable Array Logic), PLA(Programmable Logic Array), CPLD(Complex Programmable Logic Arrary), FPGA(Field Programmable Gate Array)와 같은 프로그래머블 논리 소자를 선호하여 왔다. 그 첫 번째 이유로, 설계자는 이러한 한 개의 물리적 디바이스 내에 복잡하고 다양한 기능을 구현할 수 있기 때문이다. 프로그머블 논리 소자는 여러 규격화된 디바이스의 사용뿐만 아니라 외부 배선과 관련된 불편함과 신뢰성 문제를 없앨수 있다. 두번째 이유로, 설계자는 이러한 디바이스를 사용하여 쉽게 설계변경이 가능하기 때문이다. 즉, 설계에 오류가 있거나, 설계사양이 변경된 경우에, 이러한 프로그래머블 논리 소자를 재프로그램밍 할 수 있다.

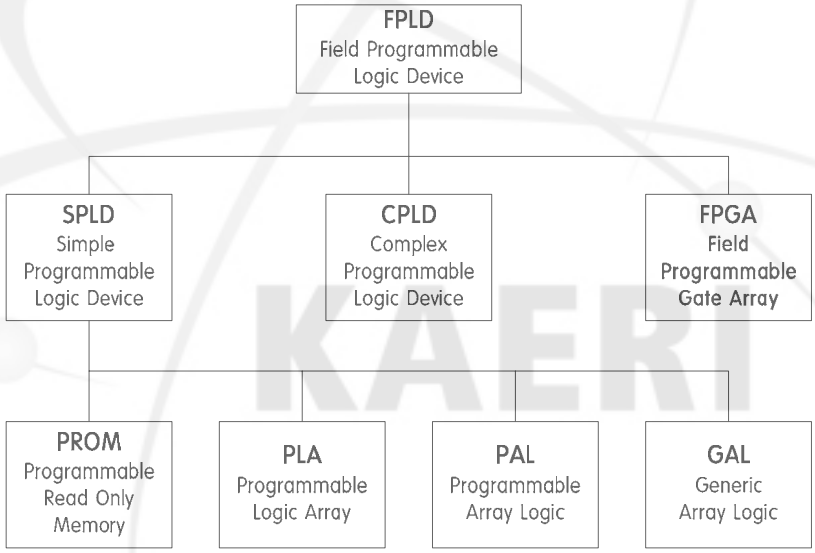


그림 2.1 프로그래머블 논리 소자

그림 2는 많이 사용되는 프로그래머블 논리 소자를 분류한 것이다. 그림2의 최상위에 위한 FPLD의 의미는 디바이스가 반도체 제조공정에서 프로그램되는 것이아나, 사용자의 "사용현장"에서 설계자에 의해 직접 프로그램된다는 것을 의미하며, 일반적으로 프로그래머블 논리 소자라함은 이러한 필드 프로그래머블 논리 소자를 의미한다.

AND-OR 회로의 형태를 갖는 사용자 프로그래머블 로직은 1970년대 초에 개발되었다. 1972~1973년 경에 현장에서 1회만 프로그램할 수 있는 로직 어레이를 사용할 수 있게 되

어, 설계자가 즉시 맞춤형 설계를 할 수 있도록 하였다. 이러한 디바이스를 필드 프로그래머블 로직 어레이(FPLA)라고 한다. 이와 유사한 디바이스는 프로그래머블 어레이 로직(PAL)이다. PAL와 PLA는 게이트 어레이로 이루어진다. PLA에는 프로그래머블 AND 어레이와 프로그래머블 OR 어레이가 있어서, 사용자가 이 두 레벨의 게이트를 이용하여 조합함수를 구현할 수 있게 한다. PAL은 PLA의 특별한 경우로, OR 어레이는 고정되어 있고 AND 어레이에 대해서만 프로그램할 수 있다. PROM, PAL, PLA, GLA를 통칭해서 SPLD라고 한다.

CPLD는 여러 개의 SPLD를 하나의 칩에 내장하고 각각의 SPLD 블록들을 상호연결하여 프로그램이 가능한 디바이스다. SPLD에 비해 집적능력이 크며, 500~16000 게이트의 범위를 갖는다.

1980년 대 말에 Xilinx사는 프로그래머블 논리 소자에 대한 설정 정보를 저장하기 위하여 정적 랜덤 엑세스 메모리(RAM)을 사용하였고, 상당히 많은 양의 로직을 집적할 수 있는 FPGA라는 디바이스를 만들었다. 곧바로 여러 PLD 공급자와 게이트 어레이 공급자들이 시장에 뛰어들어 다양한 구조를 갖는 FPGA를 생산하게 되었다. 일부 구조들은 재프로그래머블 기술을 사용하였고 다른 구조들은 프로그래머블 퓨즈 기술을 사용하였다. FPGA 기술은 지난 15년간 꾸준히 발전하여 왔으며, 오늘날에는 5 백만 게이트 이상을 갖고 있는 FPGA도 있다.

제 2 절 FPGA 구조

1. SPLD 구조

1) PLA 구조

그림 2.2는 3개의 입력과 4개의 출력을 갖는 PLA를 보여준다. 따라서, 그림 2.2의 PLA는 3개의 변수를 갖는 4개의 함수를 구현할 수 있다.

그림 2.2의 회색 영역과 같이 PLA는 프로그래머블 AND 어레이와 프로그래머블 OR 어레이로 구성되며, 각각의 AND 어레이와 OR 어레이는 프로그램 할 수 있도록 되어있다. PLA의 모든 입력은 AND 어레이를 거쳐서 각각의 AND 게이트의 입력과 연결될 수 있으며, AND 게이트의 모든 출력은 OR 어레이를 거쳐서 모든 OR 게이트의 입력을 연결될 수 있다. PLA의 AND 어레이와 OR 어레이의 교차점은 사용자가 프로그램하기 전에는 전기적으로 연결되어 있지 않다. 사용자가 원하는 기능을 구현하기 위해서 이러한 특정 교차점을 전기적으로 연결하면 OR 게이트의 출력에 특정 함수가 출력된다.

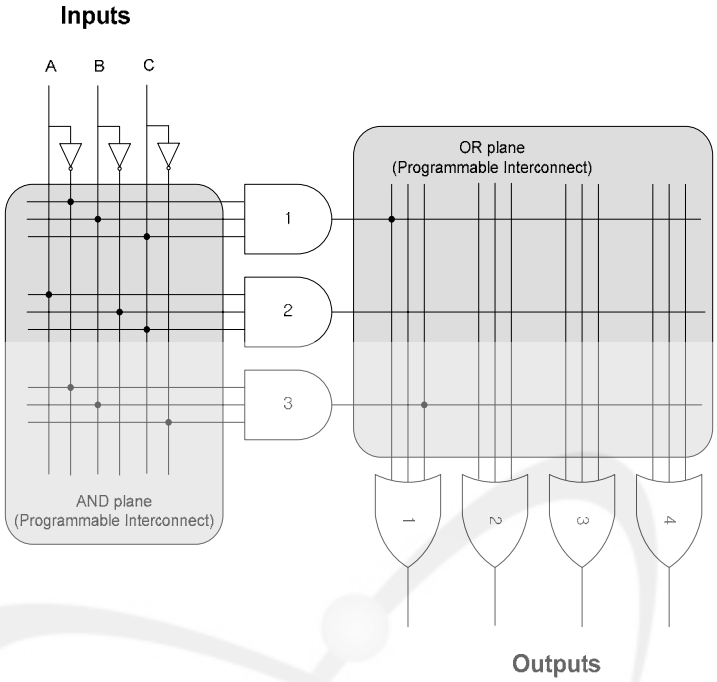


그림 2.2 프로그래머블 로직 어레이 구조

그림 2.2는 특정 교차점이 전기적으로 프로그램된 PLA를 보여주며, 각각의 AND 게이트 및 OR 게이트의 출력은 다음과 같다.

(1) 1번 AND 게이트 출력 = A'BC
(2) 2번 AND 게이트 출력 = AB'C
(3) 3번 AND 게이트 출력 = A'BC'
(4) 1번 OR 게이트 출력 = A'BC + A'BC'

2) PAL 구조

PAL은 PLA의 특수한 경우로, AND 어레이는 프로그램할 수 있지만 OR 어레이는 고정되어 있다. PAL은 그림 2.3과 같이 기본적으로 PLA와 같다. AND 어레이만 프로그램할 수 있기 때문에, PAL은 일반적인 PLA보다 제조비용이 비교적 저렴하고, 프로그램이 용이하다. 이러한 이유로, 논리 설계자는 여러 논리함수를 구현할 때 개별 논리 게이트를 대체하기 위해 흔히 PAL을 사용한다. 그러나, 디지털 논리를 구현함에 있어서 PLA보다 덜 유연성을 갖는다.

그림 2.2 및 2.3에서 보여진 것처럼, PLA와 PAL에는 조합회로가 구현될 수 있다. 또한, PAL으로부터 구동되는 입력을 가지고 있는 래치 또는 플립플롭을 내장하고 있는 PAL 도 있다. 이러한 PAL을 순차 PAL이라고 한다. 이러한 PAL을 이용하면 순차회로를 편리하게

구현할 수 있다.

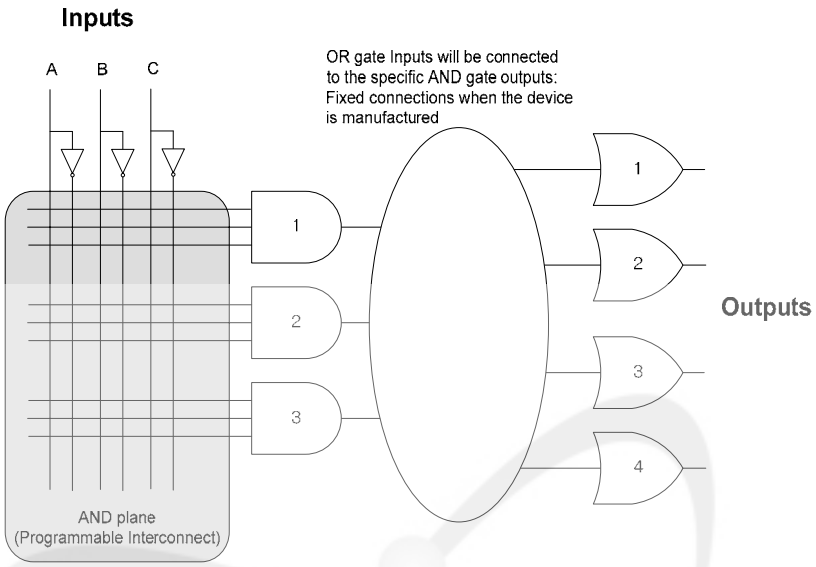


그림 2.3 프로그래머블 어레이 로직의 구조

2. CPLD 구조

CPLD는 PAL 개념의 확장이다. 일반적으로 CPLD는 프로그램 가능한 연결선 행렬구조와 더불어 PAL과 같은 로직 블록(logic block)으로 구성된 IC이다. 보통, CPLD는 500 ~ 10,000 개의 논리 게이트를 가지고 있다.

일반적인 CPLD는 많은 기능 블록들로 구성되며, 각 기능블록은 매크로셀(macrocell)과 PLA(또는 PAL)로 구성되어 진다. 그림 2.4는 8개의 기능 블록으로 구성된 CPLD를 보여준다. 각각의 매크로셀은 플립플롭과 멀티플렉서를 가지고 있으며, 이러한 플립플롭과 멀티플렉서는 기능 블록에서 입출력(I/O) 블록 또는 상호연결 어레이들과의 신호 연결을 위해서 사용된다. 각각의 블록들과의 연결은 상호연결 어레이(interconnection array)에 의해 이루어진다. 각각의 메크로셀은 플립플롭과 입력들이 AND 게이트 어레이와 연결되어 있는 OR 게이트를 포함하고 있다. 몇몇 CPLD들은 PAL 기반이며, 이 경우 각각의 OR 게이트들은 고정 AND 게이트 셀과 조합되어 있다. 그 외의 CPLD들은 PLA 기반이며, 기능 블록안의 AND 게이트의 출력은 어느 것이든 그 블록안의 OR 게이트 입력과 연결될 수 있다.

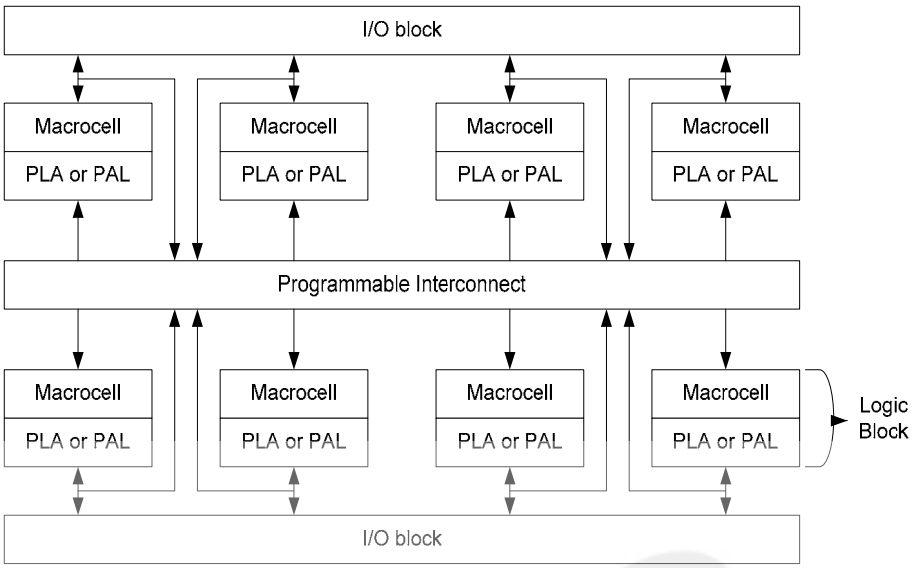


그림 2.4 일반적인 CPLD 구조

3. FPGA 구조

FPGA는 고밀도 프로그래머블 논리 소자로 분류된다. FPGA도 PAL과 마찬가지로 전기적 인퓨즈에 의한 사용자 프로그래밍으로 원하는 회로를 빠른 시간에 구현할 수 있게 하여 준다. 그러나 PAL이 일반적으로 AND -OR 게이트로 된 구조적인 어레이를 취함에 따른 회로 구현의 효율성이 낮은 것에 비하여 FPGA는 다양한 형태의 디지털 회로를 구현할 수 있는 논리 및 연결구조로 인하여 고성능의 회로를 구현할 수 있게 한다.

FPGA는 1985년 Xilinx사에 의하여 처음 소개된 이후로 현재 많은 회사가 다양한 형태의 유사 소자들을 제공하고있다. FPGA는 소자에 포함된 게이트 개수, 입출력블록 및 핀 수, 동작 속도에 따라 가격의 차이가 매우 크고 구현 후 회로 성능에도 차이가 있으므로 응용 목적에 적합한 소자를 사용하는 것이필요하다.

그림 2.5는 전형적인 FPGA의 구조를 보여준다. FPGA의 내부는 일반적으로 프로그래머블 로직 블록(programmable logic block), 프로그래머블 입력/출력 블록(programmable I/O block), 프로그래머블 연결(programmable interconnect)로 구성되어 있다.

프로그래머블 로직 블록에는 디지털 회로를 구현할 수 있는 게이트, 플립플롭, 멀티플렉서, 룩업 테이블 등이 배치되어 있다. 로직 블록을 프로그램한다는 것은 멀티플렉서로 가는 입력 또는 제어 신호를 변경한다든지, 룩업 테이블의 값을 변경한다든지, 또는 AND-OR 게이트 블록들에서 특정 게이트를 선택 또는 비-선택 하는 것을 의미한다.

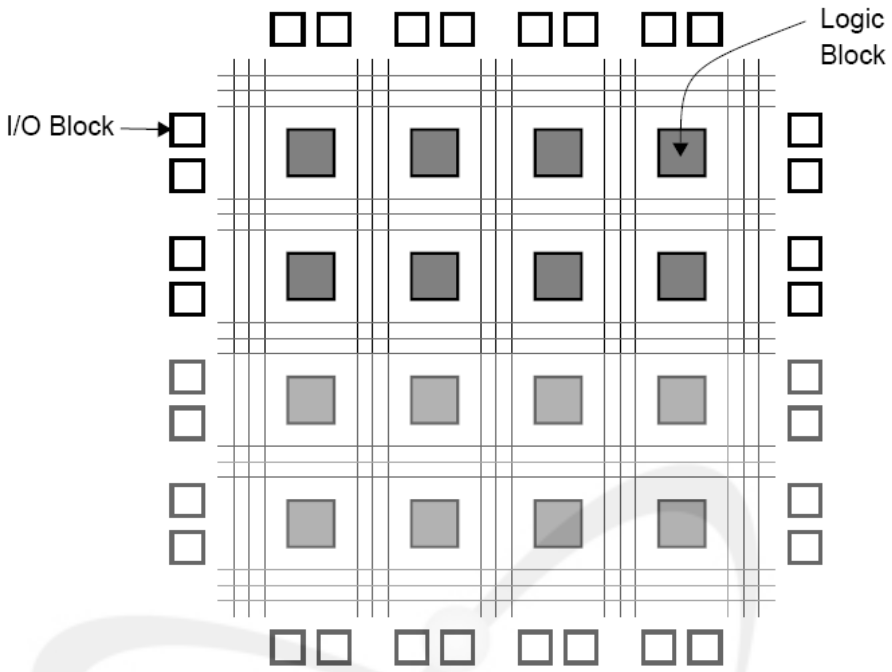


그림 2.5 FPGA 구조

배선 연결에서 프로그램한다는 것은 특정 연결을 만든다든지 또는 끊어버리는 것을 의미한다. 이것은 칩 내부의 다양한 블록을 연결하고, 특정 I/O 핀들을 특정 로직 블록들과 연결하기 위하여 필요하다.

프로그래머블 I/O 블록들은 입력, 출력, 또는 양방향성으로 동작하도록 프로그램이 가능한 블록들을 의미한다. 전형적으로 이러한 것들은 반전/비반전, 삼상태, 수동 풀업과 같은 버퍼의 속성, 또는 핀의 신호 변화 속도(Slew rate)를 조절하기 위한 프로그램을 할 수 있다.

그림 2.5는 일반적인 FPGA 구조를 보여주고 있지만 모든 FPGA가 이러한 구조로 되어 있는 것은 아니다. 상용 FPGA는 공급자에 따라 다양한 구조로 되어 있다.

제 3 절 FPGA 프로그래밍 기술

프로그래머블 논리 소자는 프로그램 가능한 상호연결선 구조로 연결되어 있는 로직 블록으로 구성되어 있다. 설계자는 원하는 디지털 회로를 프로그래머블 논리 소자에 구현하기 위하여 프로그래머블 로직 블록 및 로직 블록 사이의 상호연결을 “프로그램” 또는

"구성" 하여야 한다. 설계자는 프로그래머블 2진 값(0 또는 1)으로 이루어진 일련의 값들을 논리 소자에 다운로드해야 한다. 즉, 설계자는 구현하고자 하는 디지털 회로를 VHDL 또는 Verilog 같은 HDL 언어를 사용하여 설계하고, 이러한 설계 결과를 소프트웨어 도구를 이용하여 합성, 배선 및 배치를 수행한다. 이러한 일련의 작업을 수행하면, 최종적으로 "구성파일(Configuration file)"이 생성된다. 설계자는 구성파일을 프로그래머블 논리 소자에 다운로드하면 프로그래머블 논리 소자에 설계자가 원하는 디지털 회로가 생성된다. 구성파일은 일련의 2진 값들로 이루어져 있으며, 프로그래머블 논리 소자 내의 로직 블록 또는 블록 사이의 상호연결에 정보를 제공한다. 따라서, 이러한 일련의 2진 값들에 따라 로직 블록 내의 AND-OR 게이트들의 선택/비선택이 이루어 지며, 블록 사이의 상호연결을 위한 배선들의 연결/비연결이 이루어 진다.

디바이스 공급자의 제조 기법에 따라, SRAM, anti-fuse, EPROM 또는 EEPROM 등에 의하여 프로그래머블 논리 소자의 구성(또는 프로그램)이 이루어진다. 이러한 프로그래밍 기법들은 칩의 면적, 지연시간, 성능, 집적도, 표준 공정사용, 가격 등에 영향을 미치는 중요한 요소이다. 또한 정보의 불휘발성 (non-volatile), 소자의 재사용성 (re-programmable), 시스템에장착된 채로 재프로그래밍이 가능한가도 이러한 프로그래머블 논리 소자의 프로그래밍 구조에 의하여 결정된다.

1. SRAM 프로그래밍

SRAM (Static RAM) 프로그래밍 구조를 사용하는 회사로는 Xilinx, Altera, Lucent 등이 있다. 이 구조에서는 프로그래밍하기위한 연결 스위치점이 SRAM 셀에 의하여 조정되어지는 패스 트랜지스터(pass-transistors), 트랜스미션 게이트(transmission gates), 혹은 멀티플렉서등을 사용한다. 그림 2.6은 SRAM 프로그래밍 방식에 의한 프로그래머블 논리 소자의 연결 셀 구조를 보여준다. SRAM 방식에서는 연결점을 제어하기 위한 구성(configuration) 정보가 SRAM 셀에 저장된다.

그림 2.6의 패스 트랜지트터의 경우, SRAM의 정보가 0을 가지고 있으면, 패스 트랜스지스터는 OFF가 되어 두 배선의 연결이 끊어진다. SRAM의 정보가 1이면, 패스 트랜지스터는 ON이 되어 두 배선이 연결된다. 멀티프렉서의 경우도, RAM에 저장되어 있는 값에 따라 입력 중에 하나의 입력이 멀티플렉서의 출력과 연결된다.

그림 2.7은 SRAM 프로그램 방식을 사용하는 프로래그래머블 논리 소자 내에서의 논리 회로와 배선 연결에 대한 예를 보여준다. 그림 2.7에서 좌측 상단의 로직 블록의 출력은 두개의 패스 트랜지스터를 거쳐, 멀티플렉서로 입력된다. 해당 멀티플렉서는 SRAM에 의해 제어되며, 멀티프렉서의 출력은 우측 하단의 로직 블록으로 입력된다. SRAM 프로그래

밍 방식을 사용하는 프로그래머블 논리 소자는 이러한 방식으로 내부 논리회로 및 상호 연결선이 구성된다.

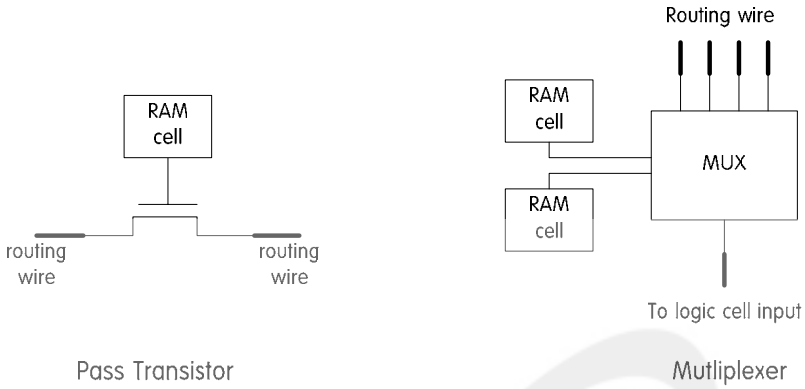


그림 2.6 SRAM 프로그래밍 기술

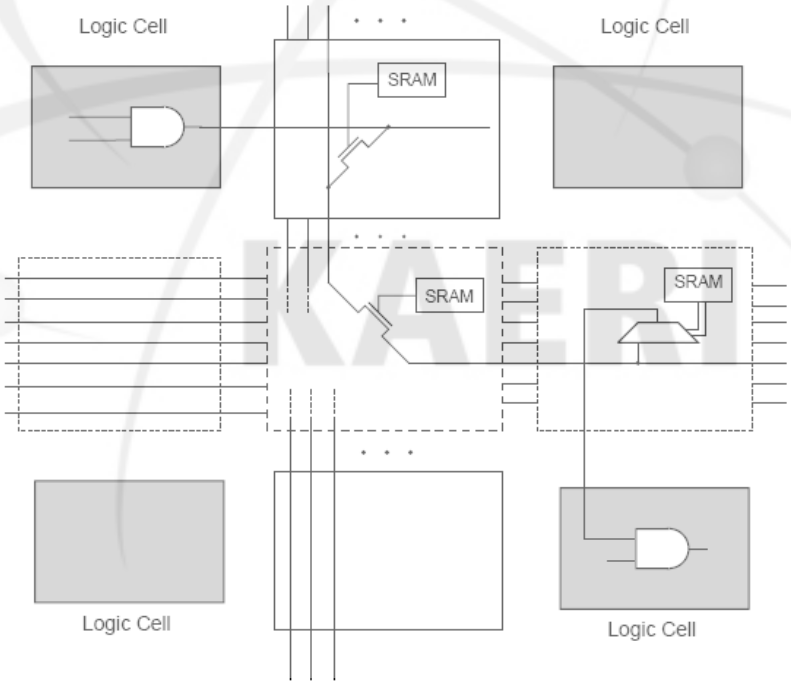


그림 2.7 SRAM에 의해 제어되는 PLD

SRAM은 동작 전원이 OFF되면 그 정보를 유실하는 휘발성(volatile) 기억소자이다. 따라서, 이방식을 사용하는 프로그래머블 논리 소자를 사용하는 시스템에서는 프로그래머블 논리 소자의 구성정보를 영구히 보존하기 위한 별도의 ROM 또는 저장 매체가 필요하다. 여기에 SRAM 셀에 대한 ON/OFF 정보를 저장하여 두고 시스템의 초기화시 이정보는 프로그

래머블 논리 소자의 SRAM을 구성하기 위하여 다운로드 된다. 다른 방식에 비하여SRAM 구조는 비교적 면적을 많이 차지하나 재프로그래밍이 시스템 안에서 장착된 상태에서 이루어질 수 있으며 표준 CMOS 공정을 사용하여 제조할 수 있다는 장점이있다. 일반적으로 다른 프로그래밍 방식에 비하여 SRAM 프로그래밍은 SRAM이 동작하기 위한 정전류 및 동적 전류로 인하여 전력소모가 비교적 크다. 스위칭 소자로 작용하는 SRAM의 저항과 capacitance 성분도 크게 되므로 회로의 성능이 제한을 받을 수 있게 된다. 따라서 SRAM 프로그래밍방식의 프로그래머블 논리 소자는 시스템의 초기개발 용도로 많이 사용되고 있다.

2. Anti-fuse 프로그래밍

Anti-fuse 방식은 Actel, QuickLogic사 등에서 사용한 방법이다. 높은 전류가 두 지점을 통과할 때 퓨즈 배선이 열리는(open) 것과는 대조적으로, Anti-fuse 방식은 높은 전압이 걸릴 때, Anti-fuse 프로그래밍 요소는 높은 저항(open)에서 낮은 저항(closed)으로 바뀐다. Anti-fuse는 n+ 확산 영역과 폴리실리콘 사이의 유전체 층 또는 금속 층 사이의 비결정 실리콘을 사용하여 만든다. Anti-fuse는 일반적으로 OFF 상태이다.

그림 2.8은 Actel에서 사용하는 anti-fuse 방식을 보여준다. 이것은 3개의 층으로 구성되는데 가장 아래층은 heavily-doped된 n+ 반도체층, 유전체로 이루어지는 중간 절연층과 최상층의 polysilicon 전도층으로 이루어진다. 이 구조에 비교적 높은 18V의 전압을 anti-fuse 터미널에 인가하고 5 mA 정도 되는 전류를 흐르게 하면, 중간층의 유전체가 열에 의하여 녹게 되고 Poly-Si 층과 n+ 층 사이에 전기적 연결이 이루어지게 된다. 퓨즈가 파괴된 후 연결되는 두 개의 전도층은 각각 전도성이 좋은 금속층과 연결되어 약 300-500 ohm의 낮은 저항 값을 가지게 된다.

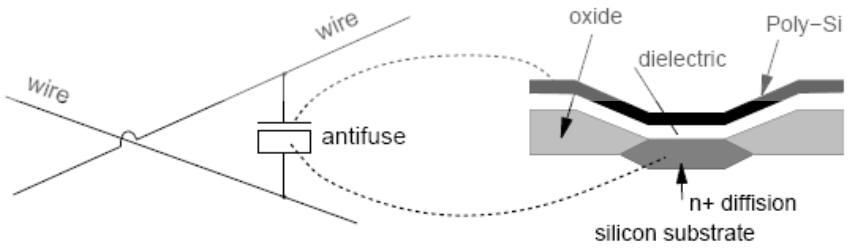


그림 2.8 Actel의 Anti-fuse 프로그래밍 기술

Anti-fuse 방식의 프로그래머블 논리 소자의 프로그래밍을 위해서는 일반 전류를 초과하

는 고전압과 전류가 필요한 별도의 마스크 제조 공정이 필요하나, 프로그래밍을 위한 칩 면적은 매우작다. 한편, 한번 프로그래밍을 하게되면 영구적으로 연결된 링크가 만들어지기 때문에 재프로그래밍이 불가능하며 프로그래밍된 정보는 전원에 상관없이 보존되는 불휘발성이다.

3. EPROM 프로그래밍

EEPROM을 포함한 EPROM 프로그래밍 방식 소자는 Altera, Lattice, Lattice 등에서 제공하고있으며 EPROM 메모리와 동일한 기술에 의하여 제조된다.

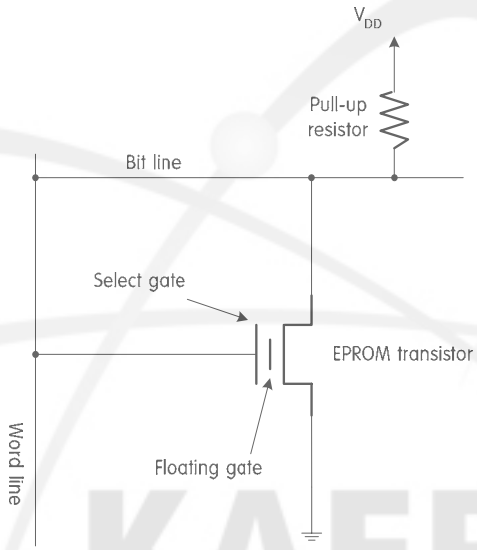


그림 2.9 EEPROM 프로그래밍 기술

그림 2.9와 같이 EPROM 트랜지스터는 select와 floating 두개의 게이트를 가진 구조로, floating 게이트는 select 게이트와 트랜지스터의 채널사이에 전기적으로 연결되지 않은 상태로 들어 있다. 프로그래밍이 되지 않은 상태에서는 floating 게이트에는 전하가 존재하지 않아서 이 트랜지스터는 보통의 경우와 동일하게 select 게이트에 의하여 turn-on 시킬 수 있다. 만약, 이 트랜지스터가 프로그래밍되면 floating 게이트에 전하가 포획되는 상태가되고 이전하는 트랜지스터를 영구적으로 turn-off 하게한다.

이러한 방법으로 EPROM은 프로그래밍 스위치로 사용된다. 프로그래밍된 EPROM 소자는 불휘발성이나, 자외선을 일정시간 조사하게되면 포획된 전하를 잃어버리게 되므로 재프로그래밍이 가능하고 칩면적이 매우 적다는 장점이 있다. 그림 2.9에서 프로그래밍 소자인 EPROM 트랜지스터는 pull-down 소자로 사용되므로 pull-up 저항을 통한 정적전력

소모가 발생하는 단점이 있다.

4. 프로그램밍 기술의 비교

표 2.1은 재프로그래밍 가능성, 저장능력, 방사선 저항능력 등의 다양한 측면에서 프로그래머블 논리 소자의 프로그래밍 방식에 따른 특성을 기술하고 있다.

SRAM 방식은 성능 및 게이트 집적도에서 우수하여 복잡한 회로를 구현 할 수 있는 반면에서, 방사선에 취약하고 SRAM의 휘발성에 의해서 전원이 공급이 중단되면 재프로그래밍 되어야 한다. 본 방식으로 제작된 프로그래머블 논리 소자는 원자력발전소의 안전등급의 계측제어 시스템을 개발 시에는 다중성을 갖도록 설계하는 필요하다고 판단된다. 이 방식의 프로그래머블 논리 소자는 시스템의 프로토타입 개발 또는 비안전 등급의 시스템을 개발하는 데 적합하다고 판단된다.

표 2.1 프로그램밍 기술에 따른 성능 비교

Configuration Technology	SRAM	EPROM	EEPROM Flash	Antifuse
Physical changes during programming	No	No	No	Yes
Reprogrammable	In circuit	Out of circuit	In circuit	No
Reprogramming speed	Fast	3 x slower than SRAM	3 x slower than SRAM	-
Non-Volatile	No	Yes	Yes	Yes
Live at Power-up	No	Yes	Yes	Yes
External boot device required	Yes	No	No	No
Unlimited Endurance	Yes	No	No	NA
Radiation Tolerance	Poor	Good	Good	Best
Area Occupation	Large	Small	2*EPROM	Small
IP Security	Acceptable	Very Good	Very Good	Very Good
Power Consumption	Medium	Medium	Medium	Low
Maximum working Frequency	> 3 Gbps IOs	350 MHz	350 MHz	350-500MHz
Maximum capacity	1-4 Mgates	0.9 Mgates	0.9 Mgates	0.25-0.5 Mgates

EPROM 또는 antifuse 방식의 프로그래머블 논리 소자는 성능 및 게이트 집적도에서 SRAM 방식에 비해서 낮으나, 방사선에 강하고, 비휘발성이라 전원 공급이 중단되어도

재프로그래밍할 필요가 없다. 따라서, 본 방식으로 제작된 프로그래머블 논리 소자는 원자력발전소의 안전등급의 계측제어 시스템을 개발에 적합하다고 판단된다.

제 4 절 FPGA 종류 및 공급자

프로그래머블 논리 소자는 사용자가 요구하는 어떤 기능이라도 구현하기 위해 사용될 수 있는 기본적인 빌딩 블록의 어레이로 이루어져 있다. 프로그래머블 논리 소자들은 빌딩 블록의 종류나 프로그램할 수 있는 정도가 서로 다르다. SPLD, CPLD, 또는 FPGA와 같은 프로그래머블 논리 소자 계열은 현재 여러 공급자에 의해 공급되고 있다. 또한, 이러한 공급자는 자신이 제공하는 프로그래머블 논리 소자의 설계, 구현, 시뮬레이션을 위한 소프트웨어 도구를 함께 제공하고 있다. 표 2.2는 프로그래머블 논리 소자를 제공하는 공급자를 보여준다.

표 2.2 주요 프로그래머블 논리 소자 공급자

공급자	홈페이지 URL
Achronix Semiconductor Coporaton	http://www.achronix.com
Actel Coporation	http://www.actel.com
Altera Corporation	http://www.altera.com
Atmel Corporation	http://www.atmel.com
Cypress Semiconductor	http://www.cypress.com
Lattice Semiconductor Corporation	http://www.latticesemi.com
Quicklogic Corporation	http://www.quicklogic.com
Xilinx	http://www.xilinx.com

표 2.3은 프로그래머블 논리 소자의 주요 공급자와, 각 공급자가 제공하는 프로그래머블 논리소자의 종류를 보여준다.

표 2.3 상용 프로그래머블 논리 소자 종류

Types	공급자	계 열	게이트수 (1,000)	사용자 입출력핀수
EEPROM	Altera	MAX 7000	0.6-5	36-164
		MAX 9000	6-12	159-216
	Lattice	ispLSI 3000	7-20	130-226
		ispLSI 8000	25-45	148-312
		MACH 2 (Vantis)	2.5-5	32-64
		MACH 5	5-20	68-256
	Xilinx	CoolRunner-II	32-512	33-270
EPROM	Altera	MAX 5000	0.6-3.7	16-67
		Classic EPLD	0.3-0.9	22-64
Flash	Xilinx	XC9500/XL/XV	0.8-64	34-192
	Cypress	Delta 39k	30-200	32-264
Anti-fuse	Actel	ACT 2	2.5-8	72-140
		ACT 3	1.5-10	70-228
	QuickLogic	pASIC 1	1-8	40-180
		pASIC 3	8-38	70-316
SRAM	Xilinx	XC 4000E	2-85	64-448
		Virtex-5	30-330	400-1200
		Spartan-3	50-5000	124-784
	Altera	FLEX 10K	10-250	59-470
		APEX 20K	30-1500	128-808
		Cyclone III	50-120	94-535
		Stratix III	50-340	288-1104

주의1) PLD 게이트 수로 환산됨, 다른 소자는 ASIC 게이트수로 환산된 것임.

주의2) PLD 게이트 수가 ASIC 게이트 수에 비해 각 회사에 따라 1.5-3배 정도 크게 나타남.

제 5 절 FPGA 기술의 취약성

1. 신뢰도

전자기술의 발전으로 반도체로 구성된 모든 전자부품의 집적도가 급진적으로 높아지고 있다. 이러한 집적도가 높아질수록 전력의 밀집도(Power Density)가 높아지고, 이로 인하여

반도체의 Junction Temperature 가 높아진다. 또한, 집적도가 높아질수록 소자간의 연결 부위의 wear out 확률이 높아진다.

일반적으로 디지털 부품의 고장률은 그림 2.10과 같은 욕조곡선(Bathtub)의 모양을 갖는다. 그러나, 부품의 집적도가 높아질수록 그림 2.10의 아래의 욕조곡선에서 위의 욕조곡선으로 이동한다. 즉, 집적도가 높아질수록 전체적인 부품의 고장률이 높이지고, 집적도가 낮은 부품보다 Wear out 현상이 일찍 발생한다. 또한 고장률이 상수인 구간이 짧아진다.

이러한 이유로 인해, 시스템 개발을 위해서 집접도가 높은 디지털 부품을 사용하면, 예방정비기간이 짧이지며, 이는 곧 시스템의 유지 및 보수에 드는 비용이 증가한다는 것을 의미한다.

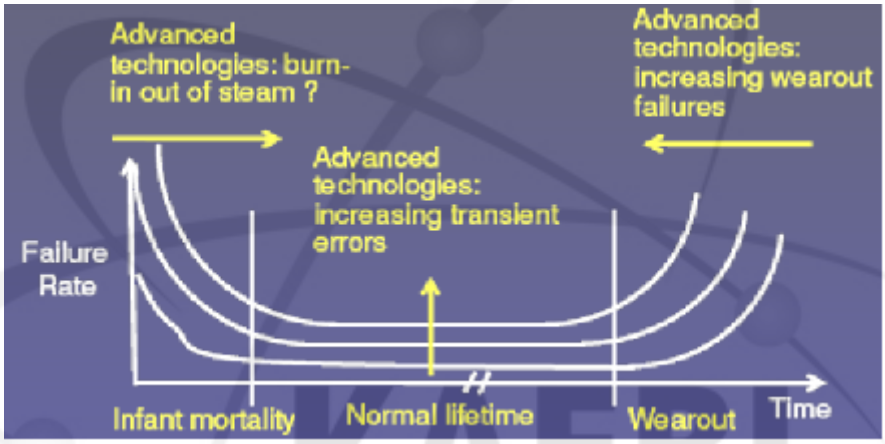


그림 2.10 집적도에 따른 전자 부품의 신뢰도 그래프

2. 방사선의 영향

그림 2.11과 같이 태양 또는, 타 은하에서 지구에 도착하는 우주선(Cosmic Ray)은 지구의 대기권과 충돌하면서 중성자(Neutron), 양성자(Proton), 광자(Photon), 알파입자(alpha particle)같은 다양한 입자들을 생성한다. 이렇게 생성된 입자들이 다시 대기 중에 있는 다른 입자들과 재 출동을 하면서 2차적인 입자들을 생성하게 되고, 이러한 2차 입자들이 다시 3차 충돌을 하게된다. 이러한 충돌이 연쇄적으로 발생하면서, 지구의 상층부에서 부터 지표까지 다양한 입자들이 대기 중에 존재하게 된다.

그림 2.11 우주에서 지구상에 미치는 방사선 효과

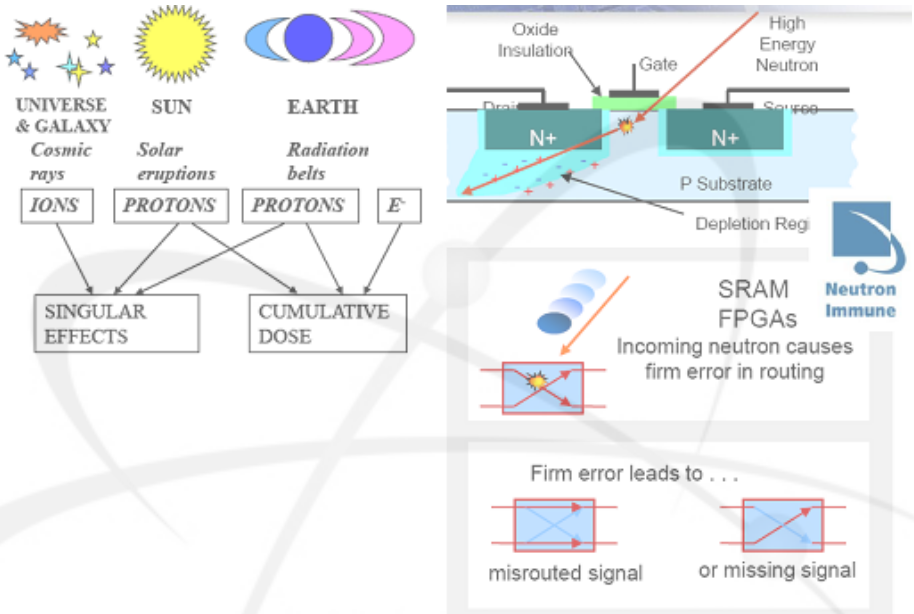


그림 2.12 방사선이 반도체 부품에 미치는 영향

이렇게 발생된 입자들은 그림 2.12와 같이 반도체 전자부품에 다음과 같은 현상을 발생시킨다. 이러한 현상은 전자부품의 일시적인 오작동 또는 영구적으로 전자부품을 파괴시킨다.

ㄱ) Ionizing Effect

- Non Destructive Effect
  - SEU (Single Event Upset)
  - MBU (Multiple Bit Upset)
  - SEFI (Signle Event Functional Interrupt)
  - SET (Single Event Transient)
  - SED (Single Event Disturb)
  - SHE (Single Hard Error)
- Destructive Effect
  - SEL (Single Event Latch-up)

- SESB (Single Event SnapBack)
- SEB (Single Event Burnout)
- SEGR (Single Event Gate Rupture)
- SEDR (Single Event Dielectric Rupture)

ㄴ) Ionizing Effect

- Move of atoms in cystals
- Creation of defects
- Creation of additional levels of trapping

현재 지표(Sea Level)에서 반도체 전자부품에 영향을 가장 많이 주는 입자는 중성자로 알려져 있으며, SRAM 방식의 프로그래머블 논리 소자가 중성자에 가장 취약한 것으로 보고되고 있다. 표 2.4는 지표, 지상 5000ft, 지상 30000ft 및 지상 60000ft에서 특정 공급자 FPGA의 방사선에 의한 SEU 실험결과를 보여준다.

표 2.4 FPGA SEU 실험결과

FPGA	Technology	Equivalent Functional Failure FIT Rates per Device			
		Ground-Level Applications		Commercial Aviation	Military Aviation
		Sea Level	5,000 Ft	30,000 Ft	60,000 Ft
Actel AX1000 1M-Gate	0.15µm Antifuse	No Failures Detected	No Failures Detect	No Failures Detected	No Failures Detected
Actel APA1000 1M-Gate	0.22µm Flash	No Failures Detected	No Failu Detect	Failures Detected	No Failures Detected
Actel A3P1000 1M-Gate	0.13µm Flash	No Failures Detected	No Failu Detect	Failures Detected	No Failures Detected
Xilinx XC2V3000 3M-Gate	0.15µm SRAM	1,150 FITs	3,900 FITs	170,000 FITs	540,000 FITs
Xilinx XC3S1000 1M-Gate	90nm SRAM	320 FITs	1,100 FITs	47,000 FITs	150,000 FITs
Altera EP1C20 1M-Gate	0.13µm SRAM	460 FITs	1,600 FITs	67,000 FITs	220,000 FITs
Altera EP2C20 1M-Gate	90nm SRAM	700 FITs	2,400 FITs	103,000 FITs	330,000 FITs
Altera EP2S30 2M-Gate	90nm SRAM	1,500 FITs	5,200 FITs	225,000 FITs	710,000 FITs

주의) SEU 결과는 FIT(number of errors in 109 hours) 단위로 표시되어 있음.

3. 타이밍 지연

그림 2.13과 같이 이상적인 상황에서의 프로그래머블 논리 소자내의 디지털 회로는 입력에 대하여 시간의 지연이 없이 정확한 출력을 발생한다. 그러나, 현실에서는 회로의 물리적 특성, 사용온도, 사용전압, 회로설계에 적용된 기술에 따라, 각 게이트 및 신호통로에 신호 지연이 발생하며, 이러한 신호 지연에 의하여 그림 2.14와 같이 입력에 대한 출력의 신호지연 현상이 발생한다.

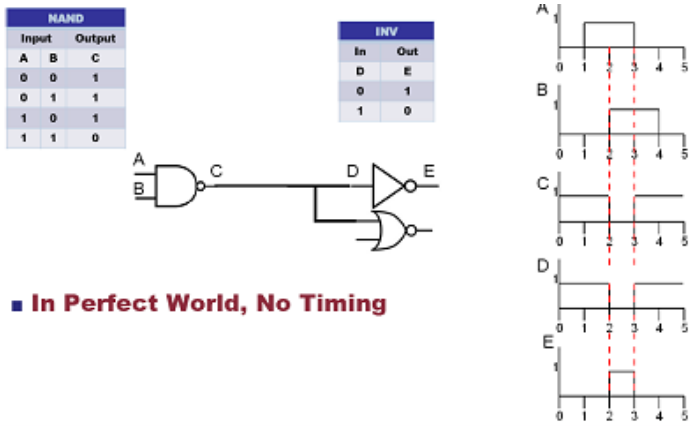


그림 2.13 이상적인 상황에서의 로직 신호

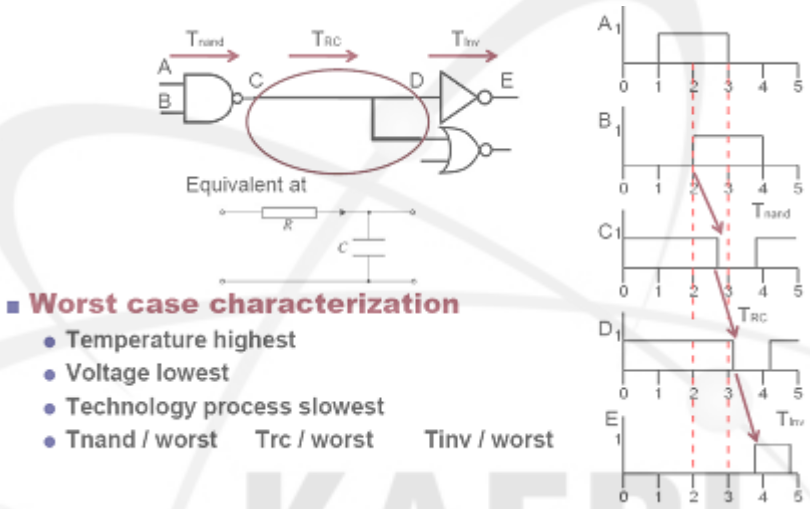


그림 2.14 현실 상황에서의 로직 신호

신호 지연 현상 중에 의해 디지털 회로에는 글리치(glitch), 준안정성(metastability)과 같은 다양한 불안정성이 존재하게 되고, 이로 인해 설계자가 의도했던 회로의 기능이 정상적으로 작동하지 않을 수 있다. 이러한 현상을 고려하지 않고 프로그래머블 논리 소자를 설계한다면, 최악의 경우, 요구된 출력신호를 손실할 수 있다.

1) 글리치 (Glitch)

글리치(Glitch)는 디지털 회로에서 발생할 수 있는 매우 짧은 기간동안 나타나고, 사라지는 전압이나, 전류의 원하지 않은 노이즈 펄스이다. 이러한 노이즈 펄스의 폭은 소자의 물리적 특성과 사용 환경(온도, 습도, 진동 등)에 따라 변할 수 있다. 따라서, 글리치는 사용조건에 따라 어떤 때는 논리회로에 전혀 영향을 주지 않을 수도 있고, 어떤 때는 논

리회로의 오동작을 발생시킬 수 있다. 이러한 글리치의 거동 특성에 때문에 글리치에 의한 회로의 오동작 원인을 찾기가 상당히 어렵다.

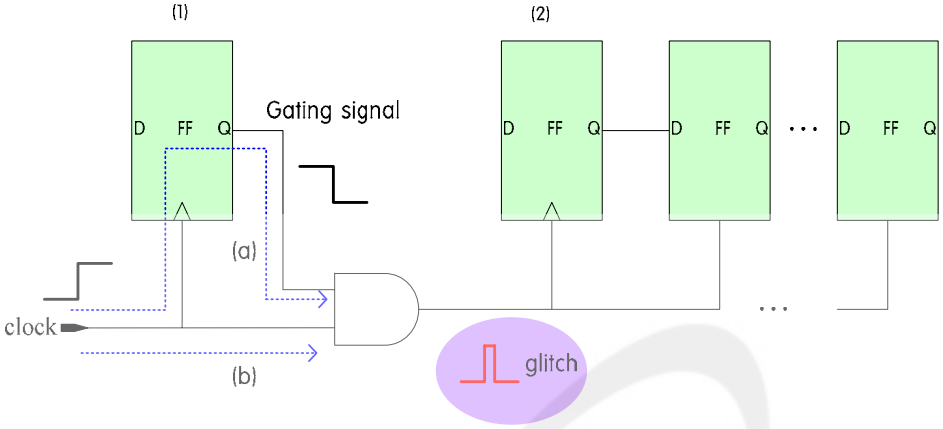


그림 2.15 잘못된 클록 게이팅 예

그림 2.15는 클록 게이팅 회로의 예를 보여준다. 클록 게이팅 회로는 전력소모를 줄이기 위한 회로 설계에 유용한 기술이며, 일반적으로 사용되는 설계방법 중에 하나이다. 즉, 그림 2.15의 (1)번 플립플롭의 출력(게이팅 신호)는 (2)번 플립플롭의 동작을 제어하는 인에이블 신호로 사용된다. 만약 (1)번 플립플롭의 출력이 High에서 Low로 변하면, AND 게이트의 출력은 클록 신호에 상관없이 Low 값을 출력한다. 따라서, (2)번 플립플롭의 클록 입력은 OFF 된다. 클록이 OFF 되면, AND 게이트의 출력을 클록으로 사용하는 도메인의 회로는 동작을 정지하고 이는 곧 회로의 전력소모량을 줄일 수 있다. 따라서, 특정 조건에서만 동작하는 회로는 이러한 클록 게이팅 회로를 사용하여 전력소모량을 줄인다.

그러나 정확하게 설계되지 않으면 래치 및 플립플롭을 잘못 트리거 할 수 있는 글리치를 발생시킬 수 있다. 그림 2.15의 예와 같이 클록의 상승에지에서 1번 플립플롭(Gating Flipflop)의 출력(Q, 게이팅 신호)이 High to Low로 바뀐다면, 클록부터 플립플롭의 출력까지의 지연(a 경로)에 의해서 클록 신호의 상승에지 신호(b 경로)보다 늦은 시간에 게이팅 신호가 AND 게이트에 입력된다. 따라서, AND 게이트의 출력에는 원하지 않은 좁은 폭의 펄스인 글리치가 발생할 수 있으며, 이는 특정 조건에서는 회로의 오동작을 발생 시 킬 수 있다.

2) 준안정성(Metastability)

이상적인 플립플롭에서 입력 D가 클록의 상승에지와 엄밀하게 동일한 시간에 변하여도

플립플롭은 정확히 동작한다. 그러나 그림 2.16과 같이 실제 프립플롭에서 입력 D는 클록의 상승에지 이전에 어느 정도의 시간동안 안정되어야 하는데, 이 구간을 셋업시간(t_{su})이라고 한다. 또한 입력 D는 클록 상승에지 이후에 어느 정도의 시간동안 안정되어야 하는데, 이 구간을 홀드시간(t_h)이라 한다. 즉, 입력 D는 시간영역에서 언제라도 변할 수 있지만, 클록의 상승에지 이전의 셋업시간부터 상승에지 이후의 홀드시간 동안의 시간구간에서는 반드시 안정된(0 또는 1로 유지) 값을 유지해야 한다.

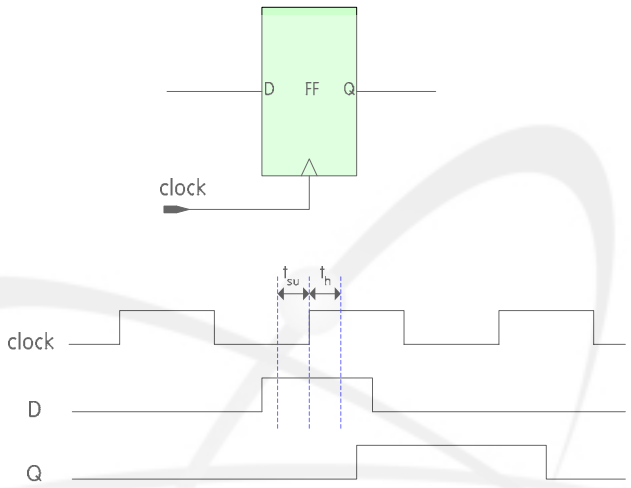


그림 2.16 플립플롭의 셋업시간 및 홀드시간

그림 2.17의 (a)와 같이 플립플롭의 셋업 및 홀드 구간에서 안정된 입력 D가 들어오면, 플립플롭은 이에 따른 출력신호(High)를 내보낸다. 그러나, 그림 2.17의 (b)와 같이 입력신호 D가 셋업 및 홀드 구간 내에서 변화면, 플립플롭은 어느 일정 기간 동안 준안정(Metastable)한 출력을 내보낸다. 여기서 준안정이란 플립플롭의 출력신호가 High로 인식되기 위한 전압(V_{high}) 수준 보다는 낮고, Low로 인식되기 위한 전압(V_{low}) 보다 높은 수준의 전압을 출력하는 상태를 말한다. 플립플롭은 일정시간이 지나면 준안정상태에서 High 또는 Low 출력을 내보낸다. 그러나, 플립플롭의 출력이 High가 될지, ,아니면 Low가 될지는 예측할 수 없다. 따라서, 이러한 준안정성은 설계된 회로의 오동작을 발생시킬 수 있다.

이러한 준안정상태는 여러원인에 의해 발생할 수 있지만, 다음과 같은 주요 원인에 의해 발생할 수 있다.

- 설계자가 소자의 특성을 파악하지 못하여 충분한 주파수의 클록을 사용하지 못한 경우
- 입력신호가 비동기 방식인 경우

- 여러 원인에 의해 클록에 비스듬한 부분(skew)이 심하게 발생한 경우
- 다른 주파수를 갖는 클록 도메인의 회로가 합쳐지는 경우
- 플립플롭의 입력이 되는 조합회로에 시간지연이 발생하는 경우

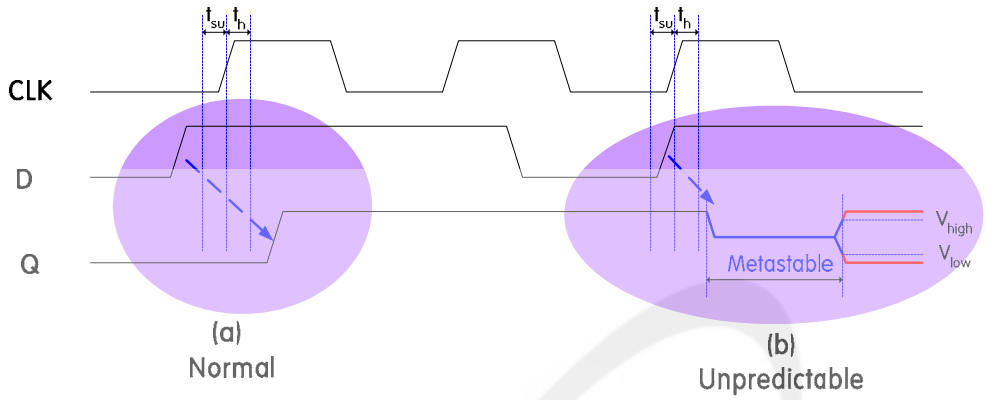


그림 2.17 플립플롭의 준안정상태

디지털 회로에서 이러한 시간 지연이 발생하는 것을 피할 수는 없다. 그러나, 이러한 시간 지연에 대해서 회로가 견딜(tolerable) 수 있는 회로를 설계하는 것은 가능하다. 예를 들어, 회로의 분석을 통해서, 충분한 주기를 갖는 클록을 선택하여 회로에 사용하면 준안정성이 발생하는 것을 최대한 방지할 수 있다. 또한, 비동기 신호를 플립플롭을 사용하여 동기화시키도록 회로를 설계하면 발생되는 글리치나 준안정상태에 대해서도 회로가 정확히 동작하도록 설계할 수도 있다. 전에 언급했듯이, 이러한 방법들은 시간지연을 방지할 수는 없고 시간지연에 대한 영향을 최소로 하게끔 설계하는 것이다.

제 3 장 FPGA 적용 현황

제 1 절 FPGA 규제기술 및 인허가 현황

1. 국내의 FPGA 관련 규제지침

국내의 원자력분야 규제기관인 KINS의 FPGA 관련 규제입장을 요약하면 다음과 같다.

ㄱ) FPGA의 개발공정은 기존의 전통적인 하드웨어 개발공정과는 다르며, 소프트웨어 개발공정과 비슷하다.

ㄴ) FPGA 개발공정은 IEEE 7-4.3.2를 만족하여야 한다.

ㄷ) FPGA 개발을 위해서는 최소한 다음과 같은 문서가 요구된다.

- Hardware 기능 요건 명세서
- 확인 및 검증 계획문서
- 소스코드 및 최종 설계 출력문서들
- Simulation Waveform
- Board Level Testing Waveform
- 설계 도구로부터 생산된 중간단계의 설계 파일들
- Test Vector Files

ㄹ) FPGA 개발에 대한 인허가를 위해서 그림 3.1과 같은 FPGA 개발공정을 따라야 하며, 각 개발공정마다 최소한 그림 3.1에서 언급한 출력물이 요구된다.

2. 미국의 FPGA 관련 규제지침

현재의 미국의 원자력분야 규제기관인 NRC의 규제입장을 요약하면 다음과 같다.

ㄱ) 현재 NRC는 FPGA 기반의 안전필수 시스템에 대한 규제지침이 없으며, 원자력발전소의 FPGA 기반의 안전필수 시스템의 인허가를 위한 지침 및 기준이 필요하다.

ㄴ) 현재 이러한 규제지침 및 기준을 규정하기 위한 구체적인 계획은 현재 없으나, FPGA 설계 시 다음과 같은 사항이 고려될 필요가 있다고 본다.

- FPGA 기반의 안전필수 시스템은 각 부품레벨, 보드레벨 및 시스템레벨에서 특정 설계지침이 필요하다.
- FPGA 기반의 안전필수 시스템의 확인 및 검증은 FPGA의 안전설계관행(practice)에 적합해야 한다.
- FPGA는 인허가뿐만 아니라 설계 생명주기 동안에 hardware/software 시스템으로 취급되어야 한다.
- 계측제어 공급자, 원자력 발전소 운영자 및 규제기관이 동일한 지침을 사

용할 수 있도록 FPGA 안전설계 관행(practice)은 규제되어야 할 필요가 있다.

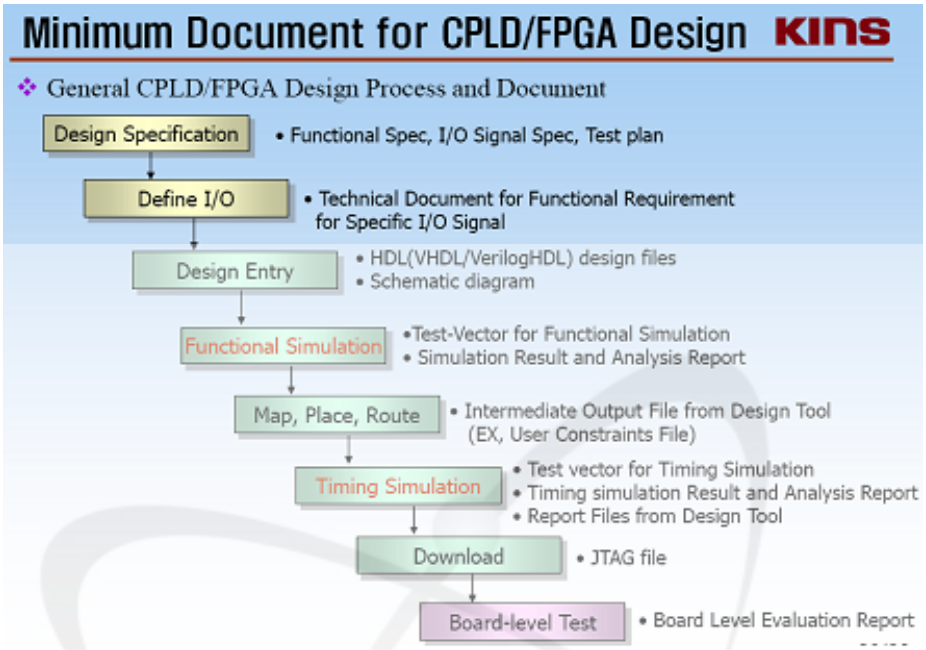


그림 3.1 FPGA 설계에 관련 최소 요구 문서

3. IEC의 FPGA 관련 기술기준

국제 기술기준 협회 중에 하나인 IEC는 원자력산업에 FPGA 기반 시스템을 적용을 위하여 CEC(Complex Electronic Component: FPGA, Multicore microprocessor 등의 고 집적회로를 기반으로 한 시스템)의 개발 및 검증관련 지침을 2012년 제정을 목표로 기술기준 IEC 62566를 개발하고 있다. 현재는 Draft 상태로 세계의 IEC 관련 회원들에게 Draft에 대한 검토를 받고 있으며, 다음과 같은 하드웨어 또는 소프트웨어에 대한 지침을 포함하고 있다.

- Pre-Developed Hardware Selection
- Built-in-Software of three different types
- Multicore microprocessors specificities
- HDL design (FPGA, PLD, ASIC, ...)

본 기술기준에서는 원자력발전소 안전등급의 계측제어 시스템에 적용을 위해서 프로그래머블 논리 소자를 설계하기 위해서는 그림 3.2와 같은 개발공정을 따를 것을 제안하고 있다.

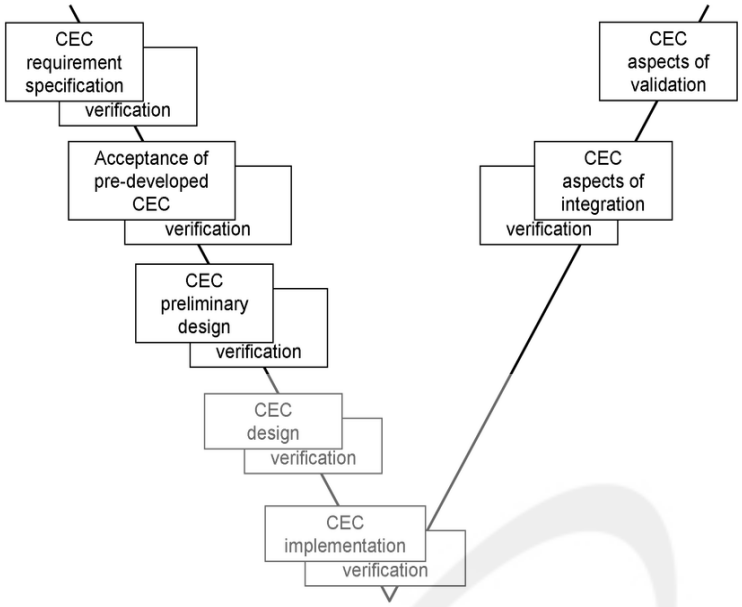


그림 3.2 IEC 62566에서 제안한 FPGA 개발공정

본 기술기준에서는 그림 3.2의 각 단계마다 수행되는 업무에 대한 요건들을 규정하고 있다. 제안된 개발공정의 특징은 소프트웨어의 개발공정과 크게 다르지 않다는 것이다. 즉, 요구사항작성, 기본 설계, 상세설계, 구현, 그리고 각 단계마다 설계에 대한 확인 및 검증 업무는 소프트웨어의 개발공정과 동일하다. 물론, 소프트웨어는 기본적으로 컴파일 된 후에, 메모리에 2진 코드로 저장되어, 마이크로프로세서가 이러한 코드를 순차적으로 실행하고, FPGA 같은 프로그래머블 논리 소자에서는 동시적으로 처리되는 디지털 회로가 된다는 점에서 세부적으로 코딩 방법, 확인 및 검증 방법의 기술적인 측면이 다르다.

4. DO254 FPGA 관련 기술기준

항공분야에서는 항공분야의 Working Group(EUROCAE, RTCA), 인증기관(Federal Aviation Administration, Joint Aviation Authorities, ...), A/C 공급자(AIRBUS, Boeing, Cessna) 및 부품공급자 참여하여 이미 DO254-ED80이라는 FPGA 관련 기술기준을 제정하여 사용하고 있다. DO254-ED80는 다음과 같은 내용으로 구성되어 있다.

- Chapter 1 : introduction
- Chapter 2 : system aspects of HW design assurance
- Chapter 3 : HW life cycle
- Chapter 4 : planning process
- Chapter 5 : HW design process
- Chapter 6 : validation and verification process
- Chapter 7 : configuration management process

- Chapter 8 : process assurance
- Chapter 9 : certification liaison process
- Chapter 10 : HW life cycle data
- Chapter 11 : additional considerations
- Appendix A : modulation of HW life cycle data based on HW design assurance level
- Appendix B : design assurance considerations for level A and B functions
- Appendix C, D : glossary of terms, acronyms

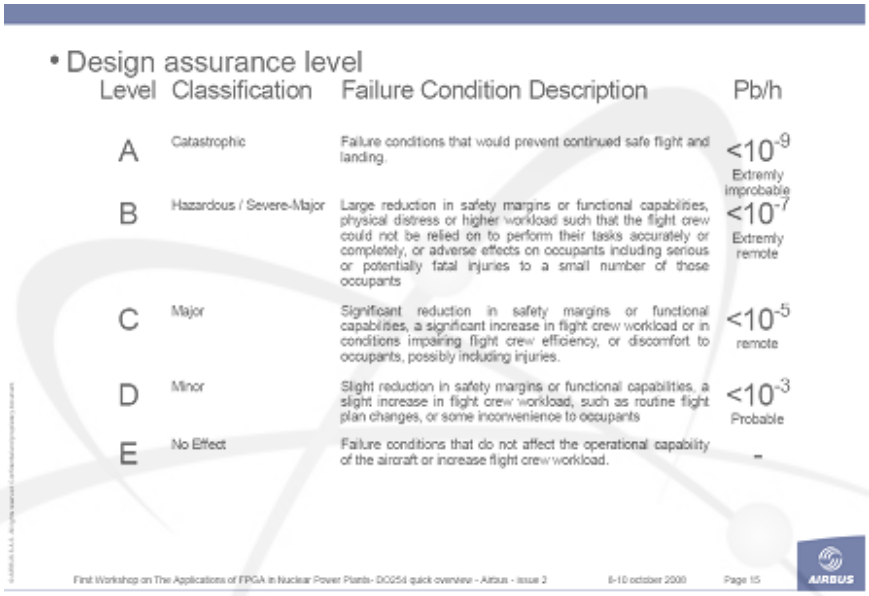


그림 3.3 시스템 Criticality Level

본 기술기준은 그림 3.3과 같이 개발되는 시스템의 Criticality level에 따라 시스템의 개발 및 검증에 필요한 활동 및 활동의 심도에 차등을 두도록 하고 있다. 예를 들어, Criticality level이 A 또는 B인 시스템에 대한 확인 및 검증은 독립적으로 수행되도록 요구하고 있다.

제 2 절 FPGA 원자력발전소 적용현황

1. 원전에 FPGA 기술 도입 방안

현재 FPGA를 이용한 시스템 개발은 3가지로 분류된다. 첫째는 기존의 노후화된 아날로그 보드를 디지털 기기로 교체하기 위해 FPGA 기술을 사용할 수 있으며, 두 번째는 노후화(생산중단)된 마이크로프로세서의 교체를 위해 FPGA 기술을 사용할 수 있다. 즉, 소프트웨어 기반의 디지털 시스템은 마이크로프로세서를 사용한다. 그러나, 이러한 마이크

로프로세서 개발 기술은 급격히 발전하고 있으며, 예전의 모델은 공급자가 생산을 중단하는 경우가 발생한다. 이러한 경우에 해당 마이크프로세서의 기능을 그대로 FPGA로 구현하여 해당 프로세서를 교체할 수 있다. 세 번째는 FPGA 기반의 디지털 시스템으로 안전필수 시스템을 새로이 개발하여 원자력발전소에 적용하는 것이다.

1) 아날로그 보드를 FPGA로 교체

그림 3.4와 같이 많은 아날로그 부품으로 구성된 보드의 기능을 한 개의 FPGA로 구현할 수 있다. 따라서, 기존의 아날로그 보드를 FPGA 기반으로 대체하면, 비용 및 공간이 절감될 수 있다.

그림 3.4 Discrete 부품으로 구성된 보드의 기능을 FPGA로 구현

2) 생산 중단된 마이크로프로세서의 교체

그림 3.5는 프랑스의 EDF 사에서 Motorola의 마이크로프로세서 모델 6800의 기능을 FPGA로 구현한 것을 보여준다. EDF는 FPGA로 MP6800의 기능을 구현하기 위하여 SRAM 타입의 FPGA로 프로토타입을 개발한 후에 Antifuse 타입의 FPGA를 사용하였다. 이와 같이 생산이 중단된 마이크로프로세서를 사용하는 시스템의 경우에, 유지 및 보수를 위해 마이크로프로세서의 기능을 그대로 구현한 FPGA로 교체할 수 있다.

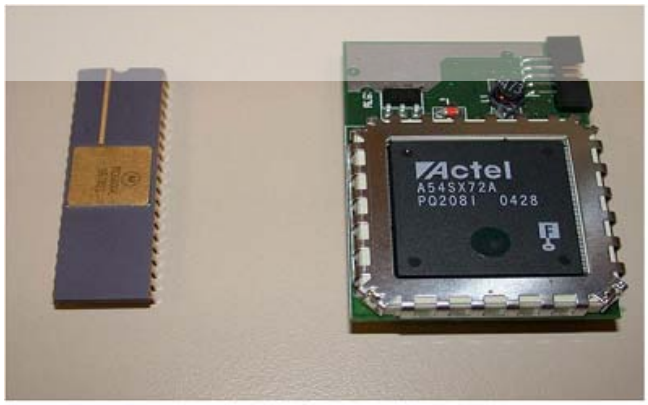


그림 3.5 Motolar MP6800을 FPGA로 교체

3) 새로운 시스템의 개발

그림 3.6은 국내의 한국원자력연구원과 주식회사 포스콘에서 개발한 PLC(Programmable Logic Controller)의 입출력 모듈의 설계를 위해서 사용된 FPGA를 적용한 예를 보여준다. 이와 같이 신규원전에 적용을 목적으로 FPGA 기술을 도입할 수 있다.

그림 3.6 포스콘 안전등급 PLC 개발

2. 국내외 FPGA 원전적용 현황

1) 유럽

프랑스의 EDF에서는 다양한 FPGA 관련 사업을 수행하고 있다. 2절에서 언급한 것과 같이 1E 등급에 적용되는 CPU 보드를 FPGA를 이용하여 재설계를 하고 있다. 또한, 다음과 같은 사업에서 FPGA 적용을 고려하고 있다.

- Upgrading of the control rod positioning system
  - Non safety classified
  - 32 900 MW units affected
- Redesign of pump speed control systems
  - 1E classified
  - 32 900 MW units affected

우크라이나의 “Raidy”에서는 원자력발전소의 안전등급 계측제어계통에 적용을 목적으로 한 I&C 플랫폼을 FPGA 기술을 이용하여 개발하였다. 또한 Raidy는 본 시스템을 이용하여 불가리아의 원자력발전소 Kozloduy 5&6 호기의 ESFAS를 개발하는 사업을 수행하고 있다. 본 사업은 2008년에 시작했으며, 2011년에 완료예정으로 있다. 그림 3.7은 Raidy에서 개발한 FPGA 기반 계측제어 플랫폼을 보여준다.

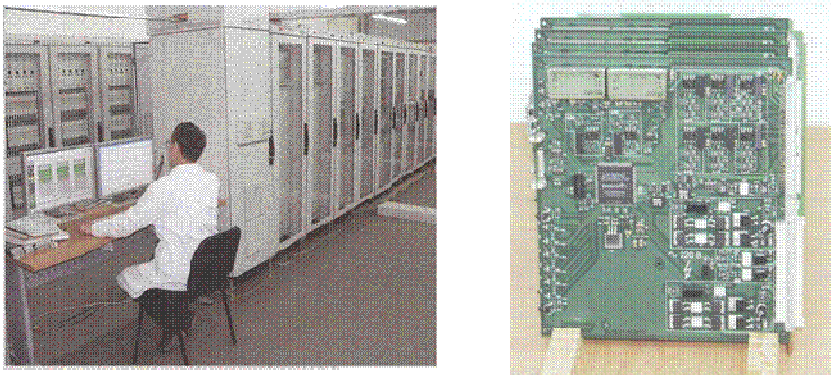


그림 3.7 우크라이나의 FPGA 기반 계측제어 시스템

2) 미국

미국의 Wolf Creek은 2004년에 RPS 및 ESFAS와 같은 안전등급의 계통에 사용하기 위한 FPGA 기반 프랫폼을 개발하는 사업을 시작했다. 현재 NRC에서 인허가 중이며, 인허가가 끝나면 2009년 이후에 발전소에 설치를 계획하고 있다. 그림 3.8은 개발된 ALS(Advanced Logic System)을 보여준다.

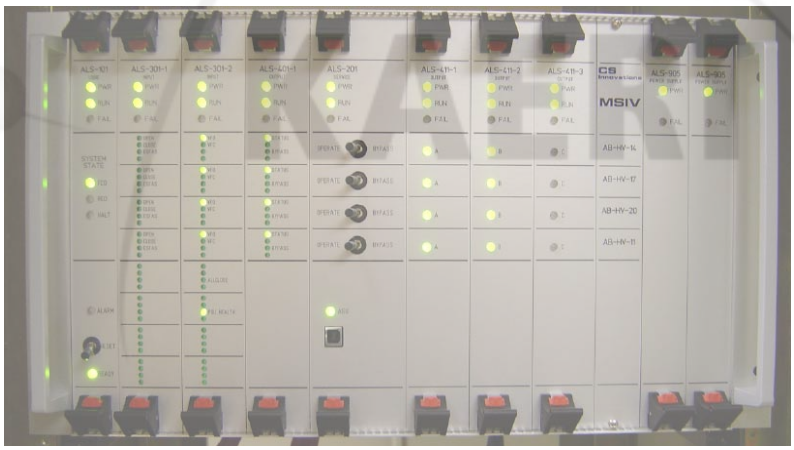


그림 3.8 ALS 플랫폼

3) 캐나다

기존의 캐나다 CANDU 원자력발전소의 대부분은 30년이 넘었으며, 기기의 노후화 문제에 당면하게 됐다. 특히, 계측제어 시스템의 교체는 당면과제로 대두되었다. 그래서 캐나에서는 노후된 I&C 기기를 대체하는 하나의 수단으로 FPGA 기반 하드웨어 플랫폼을 개

발하고 있다. 캐나다는 발전소의 Safety Shutdown System No 1(SDS1)의 트립 논리를 FPGA에 구현하였고, 이에 대한 결과를 검증하기 위한 시뮬레이션 드을 수행하였다. 그림 3.9는 SDS1의 한 채널을 보여준다. 아직 까지는 발전소에 탑재하지는 않은 상태이며, 노후화된 계측제어 시스템에 FPGA의 적용성을 파악하기 위한 업무를 수행한 것으로 판단된다.

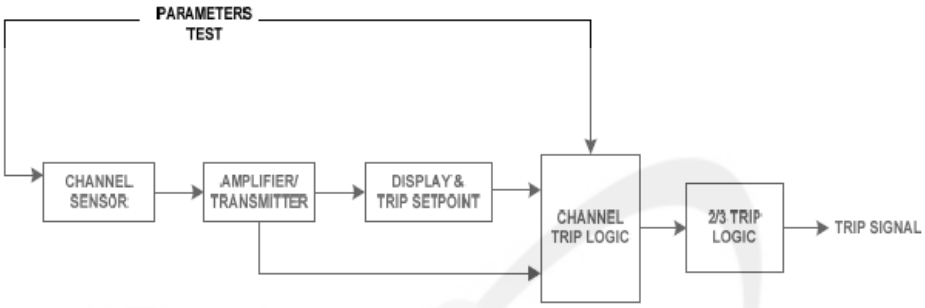


그림 3.9 CANDU의 SDS1 채널

4) 일본

일본의 도시바는 그림 3.10과 같이 BWR의 원자로에 중성자 레벨을 측정하기 위하여 FPGA를 적용한 PRMS(Power Range Monitoring System)를 개발하였다. PRMS는 원자로에 설치된 중성자 감지기와 각 냉각수 순환로에 설치된 미분 압력계(Differential transmitter)로부터 전기적 신호를 받아서 신호처리를 한후에 원자로보호계통에 신호를 출력한다.

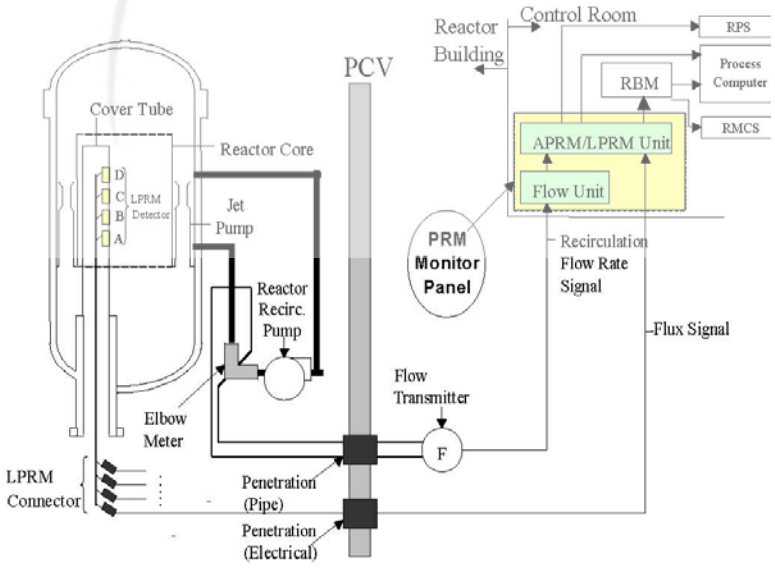


그림 3.10 일본의 BWR를 위한 Power Range Monitoring System

그림 3.11과 같이 PRMS는 LPRM(Local Power Range Monitor) 모듈과 APRM(Average Power Range Monitor) 모듈로 구성되어 있다. LPRM은 감지기로부터 전기적 신호를 받아서, 신호를 증폭한 후, 아날로그 신호를 디지털 신호로 변환한다. 변환된 디지털 신호를 이용하여 LPRM 레벨을 결정하고, 이 값이 설정치를 초과하면 알람을 출력한다. APRM은 LPRM모듈들(최대 22 LPRM)에서 받은 신호를 평균하여 평균 원자로 출력을 결정한다. 만약 평균 원자로 출력이 설정치를 초과하면 알람 또는 트립 신호를 출력한다.

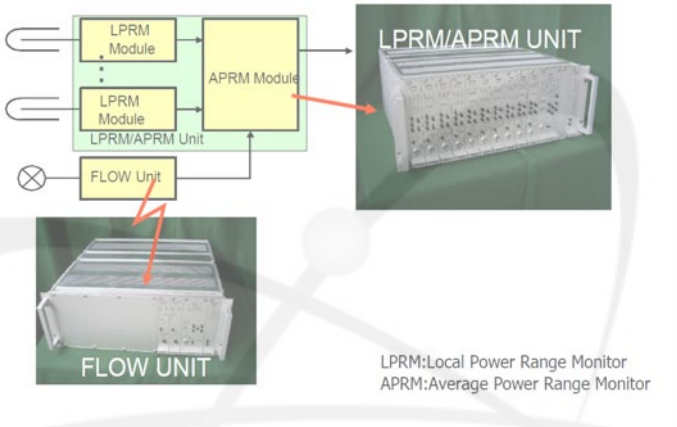


그림 3.11 PRM 구성

그림 3.12와 같이 LPRM의 모든 기능은 FPGA로 구현되었다. 즉, 신호처리 기능, 운전원에서의 설정치 입력기능, 설정치 비교기능, 운전원과의 연계등을 처리하기 위한 모든 기능을 FPGA로 구현하였다.

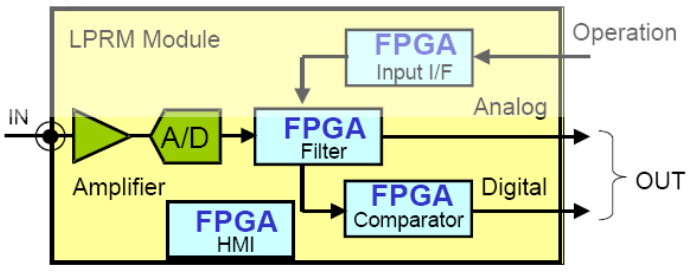


그림 3.12 LPRM 구성

5) 한국

한국에서는 모든 기능을 FPGA로 구현한 안전등급 계측제어 계통이 개발되지는 않았다.

그러나, 한국원자력연구원, 두산 중공업 및 포스콘은 KNICS(Korea Nuclear Instrumentation and Control System) 사업을 통해서 개발된 원자로 보호계통, 공학적 안전설비-기기제어계통 및 이들 계통에 플랫폼으로 사용되는 PLC를 개발하였다. 포스콘에에서 개발한 PLC의 제어기기에는 기능을 구현하기 위하여 다수의 CPLD 및 FPGA가 사용되었다. KNICS 사업을 통해 개발된 안전등급 제어기기는 한국의 인허가 기관에서 인허가를 받았으며, 인허가 검토 항목에는 이러한 CPLD 및 FPGA의 개발 및 검증 결과물도 포함되어 있다. 그림 3.13은 안전등급제어기기에서 사용된 특정 FPGA의 기능시험을 보여준다.

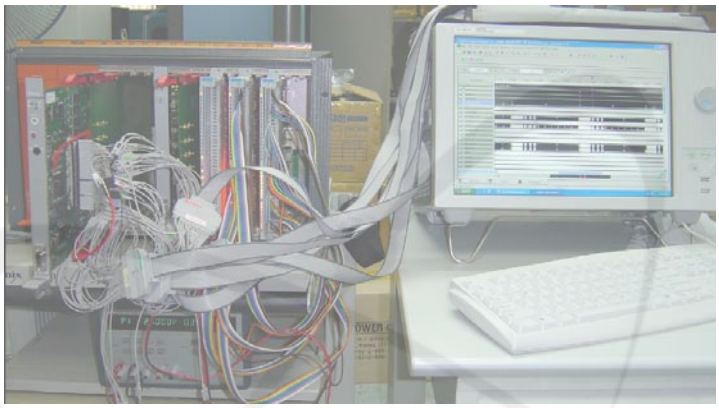


그림 3.13 안전등급제어기기 기능 시험

제 3 절 FPGA 개발공정 적용 현황

1. 개발공정

안전등급의 계측제어계통에 FPGA를 적용하기 위한 FPGA 개발 공정 크게 설계공정과 검증공정으로 구분된다. 설계공정은 시스템 요구사항 중에 FPGA에 해당하는 요구사항을 추출하여 요구사항을 만족하는 FPGA를 설계하고, 이를 구현하는 활동을 포함하며, 검증공정은 설계 및 구현된 FPGA가 요구사항을 만족하는지를 확인하고 시험을 통해서 검증하는 활동을 포함한다. FPGA에 개발에 적용되는 개발공정은 국제적으로 어느 정도 일치하는 것으로 판단된다. IEC 62556에서는 FPGA 개발을 위하여 기존의 소프트웨어 개발을 위하여 채택되었던 V 모델을 제안하고 있으며, 항공 산업 및 원자력 산업에서의 FPGA 기반 시스템의 개발을 위해서 채택된 개발공정도 이와 다르지 않다고 판단된다. 예를 들어, IEC62566은 그림 3.2와 같은 절차로 FPGA 기반 시스템을 설계하도록 제안하고 있으며, 한국원자력연구원에서 FPGA를 설계하기 위하여 채택한 모델은 그림 3.14와 같다.. 그림 3.2와 그림 3.14는 같은 V 모델을 기반으로 FPGA를 설계하고 있다. 또한 그림 3.15는 항공분야에서 채택하고 있는 개발공정이며, 이 개발공정도 V 모델을 기반으로 하고 있

다.

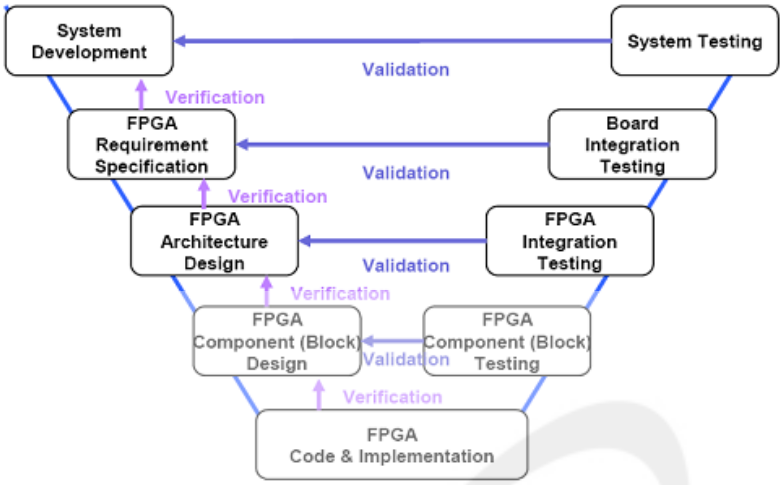


그림 3.14 한국원자력연구원의 FPGA 기반 시스템 개발공정

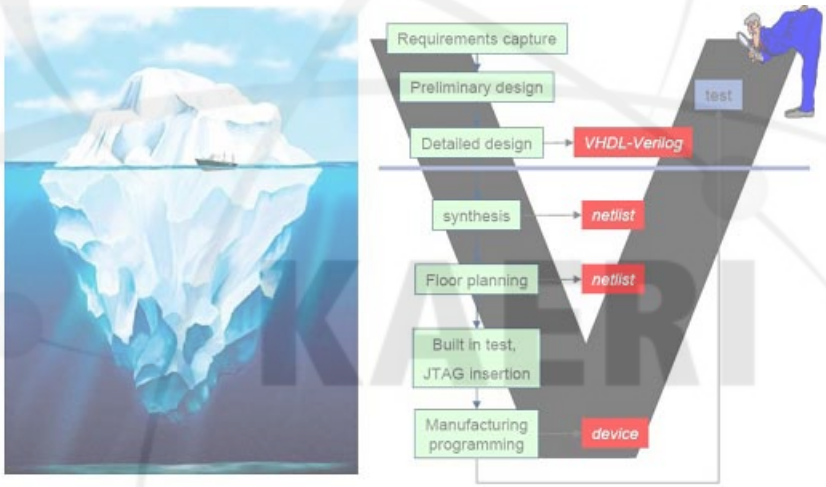


그림 3.15 항공분야의 FPGA 기반 시스템 개발공정

2. FPGA 설계활동

시스템 설계는 일반적으로 Top down 설계방법이 사용되며, 시스템-수준 설계, 보드-수준 설계, 소자-수준(FPGA 등의 단일 칩 수준) 설계로 구분된다. 각 수준마다 수행되는 일반적인 설계 활동은 다음과 같다.

ㄱ) 시스템 설계

- 시스템의 기능을 하위 단위로 분해(FPGA, DSP, GPP, Memory)

- 다중 기능을 하나의 FPGA에 구현
- FPGA 타입 및 크기 선정
- 확인 및 검증 측면에서 시스템을 분해

ㄴ) 보드 설계

- Power fluctuation 및 ground bounce를 공급
- 적절한 전원 및 ground plane 설계
- PCB layer stacking
- 최대 rise time 및 FPGA에 요구되는 power sequencing을 만족하기 위한 power regulator의 선정 및 설계
- power decoupling capacitors를 사용
- 동시의 스위칭 출력영향을 완하
- FPGA I/O의 출력 rise time을 축소
- 같은 보드에 탑재되는 디지털 영역과 아날로그 영역을 분리
- clock 및 다른 고속 라인을 분리
- FPGA power dissipation and cooling

ㄷ) FPGA 설계

- Synchronous 설계 또는 Asynchronous 설계의 선택
  - Asynchronous 설계는 빠른 응답시간, small design이 가능하나, glitches, bus skews, timing issues들에 약함.
  - Synchronous 설계는 느리지만 제어된 응답시간을 가지며, larger design이 될 수 있음. 그러나, glitch, bus skew, timing issues 들이 없음.
  - Asynchronous 신호가 FPGA안에서 Clock 신호에 따라 입력될 때는 Metastability가 중요함.
- 높은 fan-out을 갖는 신호는 timing skew를 줄이기 위하여 global FPGA line 을 사용
- I/O 버퍼를 사용하는 것보다 I/O flip-flop을 사용
- HDL State Machine Design을 사용
- SEU 영향을 완화하기 위하여 다음과 같은 방법을 사용
  - SEU immune 또는 SEU tolerant FPGA 를 사용
  - Error Detection and Correction Code 사용
  - Module Redundancy 사용
  - Scheduled or Error-initiated Reconfiguration을 사용
  - Wachdog timer를 사용
- FPGA의 사용하지 않은 pin은 high impedance 상태로 두지 말고 pull-up 또는 pull-down 저항을 갖는 전압으로 고정시킴.
- 노이즈 신호를 완화하기 위하여 voltage level translator를 사용(Max3372)

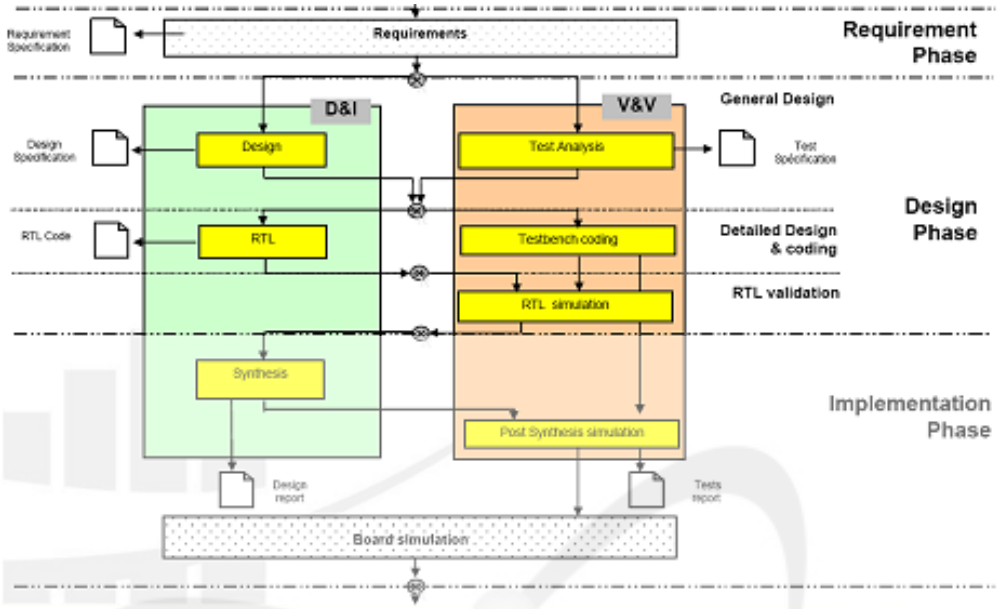


그림 3.16 FPGA 세부 설계활동

그림 3.16과 같이 FPGA 수준에서의 세부 설계활동은 다음과 같다.

(1) 요구사항명세 작성

(2) 설계명세 작성

(3) 코드 생성

(4) 합성

(5) 배선 및 배치

(6) 소자에 탑재

일반적으로 코드는 VHDL 또는 Verilog 같은 하드웨어 명세 언어로 RTL(Register Transter Level)로 작성된다. 이러한 코드는 합성을 거쳐서 게이트 수준의 netlist 가 생성된다. 그 이후에 배선 및 배치를 수행하면 최종적으로 2진 값으로 이루어진 구성파일이 생성되며, 이러한 구성파일을 최종적으로 소자에 다운로드 한다.

3. FPGA 검증활동

설계된 FPGA의 검증활동으로는 확인활동 및 검증활동을 구분된다. 확인활동은 FPGA 개발단계마다 현재 단계의 설계 결과물이 이전 단계의 설계 결과물의 요건에 따라 정확하고, 일관되게 설계되고 있는 지를 추적하는 업무들을 말한다. 검증활동은 개발된 FPGA의 최종 결과물이 시스템 요구사항 및 FPGA 요구사항을 만족하는 지를 시험을 통하여 추적

하는 업무를 말한다. 그림 3.17은 계측제어 시스템에 적용된 FPGA의 개발공정 및 개발공정에 수행되는 세부 설계 및 검증활동의 예를 보여준다.

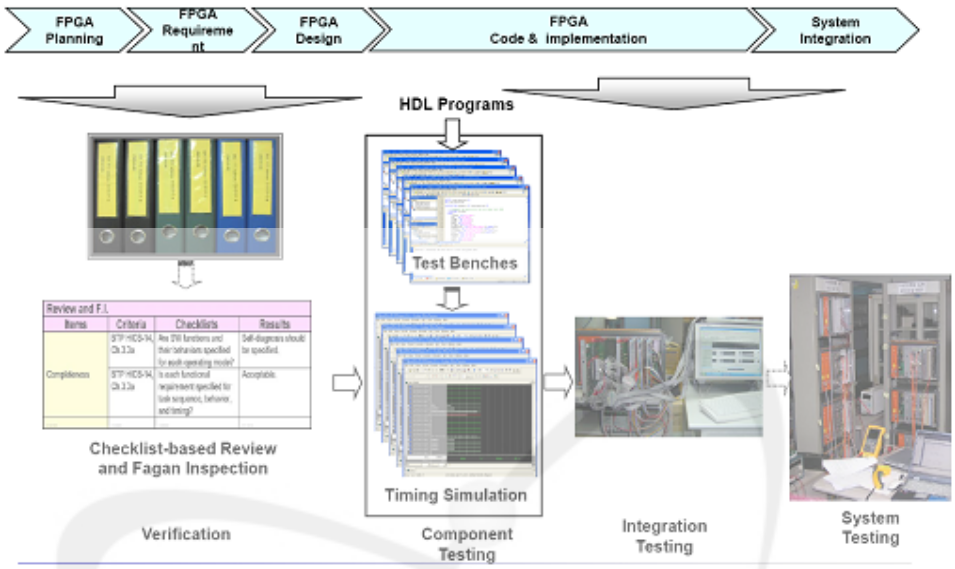


그림 3.17 FPGA 개발공정 및 활동

1) 확인활동

FPGA의 확인 방법으로는 검토, 타이밍 분석, 시뮬레이션 분석이 사용되고 있다.

가. 검토

FPGA의 확인활동의 하나로 수행되는 검토는 전문가 그룹을 구성하여, 이 전문가 그룹이 FPGA의 설계 단계에서 생산되는 모든 설계 결과물(시스템 요구사항명세서, FPGA 요구사항 명세서, FPGA 설계문서, 코드 등)의 검토하고 설계에 오류가 없는지를 확인하는 작업이다. 이 확인 작업에는 Checklist 기반 검토 등 소프트웨어 확인 작업에서 수행되는 기법을 적용할 수 있고, 다양한 상용 소프트웨어 도구를 활용하여 확인 작업을 쉽고 빠르게 수행할 수 있다.

나. 정적 타이밍 분석

정적 타이밍 분석은 구현된 FPGA 로직에 대하여 가장 Worst Case에 대하여 타이밍 분석을 수행하여 Worst Case 경우에도 요구된 기능을 요구된 제한 시간 안에 수행할 수 있는지를 확인한다. 또한, FPGA의 회로설계 시에 발생할 수 있는 글리치, Metastability 같은 타이밍 문제가 발생하지 않는 지를 확인한다. 이러한 작업은 상용 소프트웨어 도구를 활용하면 작업을 쉽고 빠르게 수행할 수 있다.

다. 시뮬레이션 분석

개발된 FPGA를 확인 업무에서 가장 중요한 업무는 시뮬레이션을 통한 FPGA의 기능 및 성능을 확인하는 것이다. 시뮬레이션이란 설계자가 작성한 코드를 FPGA에 다운로드 하기 전에, 코드의 동작이 요구된 기능 및 성능을 만족하는 지를 확인하기 위하여 코드의 입력 데이터(테스트 벡터 또는 시험사례)를 만들어 가상으로 인가하고 그 출력이 예상출력과 동일한지를 확인하는 작업이다.

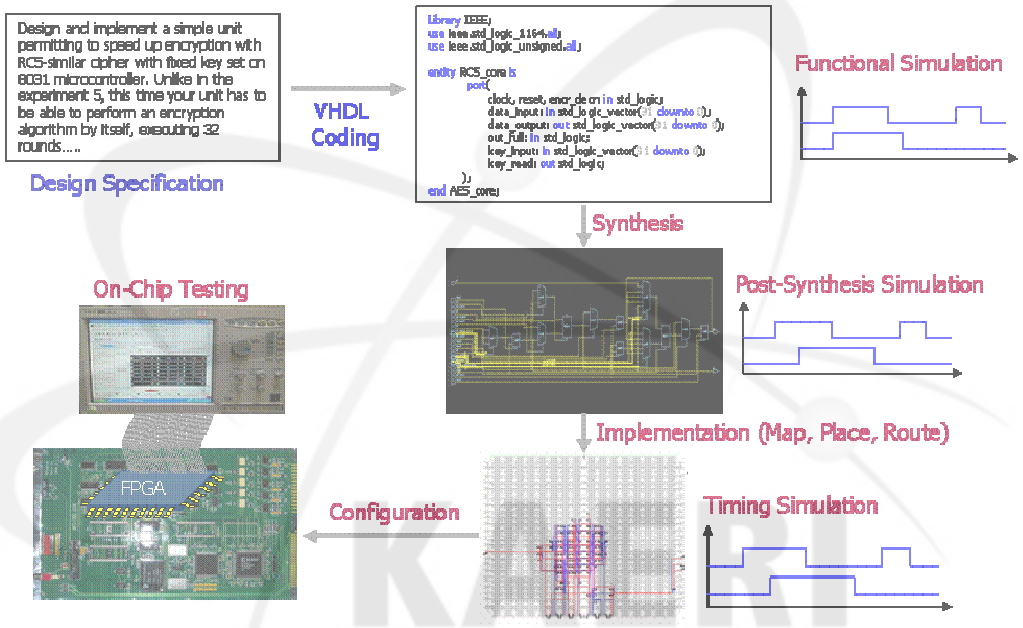


그림 3.18 시뮬레이션 분석 방법

그림 3.18과 같이 시뮬레이션은 코드의 성숙도에 따라 크게 기능적 시뮬레이션, 합성 후 시뮬레이션, 타이밍 시뮬레이션으로 구분된다. 디지털 회로를 설계하기 위해서는 FPGA의 요구사항을 정의하고, 이를 합성가능한 RTL 코드로 코딩한다. 코딩이 완료되면, 이를 시험할 수 있는 시험사례를 가상으로 인가하여 작성된 코드의 동작을 시험한다. 이를 기능 시뮬레이션이라 하며, 이때에는 신호의 지연정보를 포함하지 않은 시뮬레이션이 된다. 따라서 이러한 기능 시뮬레이션에는 내부 회로에 따른 신호의 지연이 고려됮 않기 때문에 짧은 시뮬레이션 시간에 복잡한 회로의 동작을 확인할 수 있다. 만일 이 단계에서 시뮬레이션 결과에 문제가 있는 경우에는 다시 코드를 수정하여 시뮬레이션하고, 문제가 없는 경우에는 RTL 코드를 최소한의 논리 회로들의 연결로 실제 회로로 합성하는 합성단계를 거친다. 합성 후에는 요구된 대로 회로가 합성되었는 지를 확인하는 합성후 시뮬레

이션 시험을 수행한다. 이후에는 FPGA 배치 및 배선 과정을 수행한다. 배치 및 배선이 끝난 다음에는 내부 회로가 FPGA에 물리적으로 구현 되었을 때, 신호가 배선과 회로를 지날때의 지연정보와 회로 전체의 최소한의 회로 구성요소들의 연결정보로 이루어진 네트리스트가 추출된다. 이 네트리스트를 시뮬레이션 함으로써, 신호의 지연정보를 포함한 전체 회로의 타이밍 시뮬레이션을 수행하게 된다. 위의 세 단계의 시뮬레이션에는 동일한 시험사례의 입력으로 그 동작을 확인하여, 시뮬레이션 결과에 문제가 있는 경우에는 다시 코드를 수정하여 위의 과정을 다시 반복하여 오류를 제거한다.

2) 검증활동

FPGA의 시뮬레이션이 성공적으로 이루어지면, 배치 및 배선이 완료된 설계정보를 FPGA에 다운로드한다. 이러한 작업을 “구성(Configuration)”이라 한다. 구성이 끝나면, 그림 3.19와 같이 직접 FPGA의 입력 핀에 신호를 입력하고 Logic Analyzer 등의 신호탐지 장비를 이용하여 출력신호을 분석하는 시험을 한다. 본 시험에서는 최종적으로 구현된 FPGA를 물리적으로 시험하는 시험으로 요구사항에서 기술된 모든 요구사항을 만족하는 지를 시험하기 때문에 검증 시험이라고도 한다.

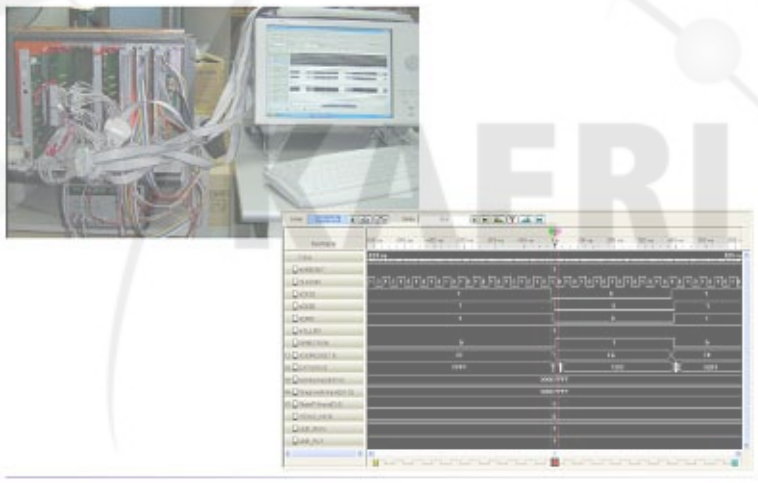


그림 3.19 하드웨어 검증 시험

4. FPGA 설계 및 검증 도구

FPGA로 구현하기 위해서는 소프트웨어 개발도구 및 검증도구를 사용하게 된다. 안전등급의 시스템에 사용되는 소프트웨어 개발도구 및 검증도구는 충분히 안전등급에 사용할 수 있음이 입증되어야 한다. 그러나, 현재 상용으로 나와있는 도구 중에는 이러한 입증과정을 거친 도구는 없는 것으로 판단되며, 현실적으로 다음과 같은 업무를 수행하는 것이 타당할 것으로 판단된다.

ㄱ) 현재 가장 많이 사용되는 도구를 사용

ㄴ) 도구의 형상을 기록

ㄷ) 잠재적인 도구의 고장을 커버할 수 있도록 FGPA 검증을 계획하고 수행

ㄹ) test coverage를 제공하는 도구를 사용

제 4 장 FPGA 설계 지침

본 장에서는 미국의 항공우주국에서 제공하는 디지털 항공전자분야의 설계지침 및 표준을 번역하여 기술하고 있다. 원자력산업뿐만아니라, 항공분야에서도 시스템의 안전을 가장 중요한 목적으로 개발하고 있다. 따라서, 항공분야에서 사용되는 설계지침을 원자력분야에 충분히 적용할 수 있다고 판단된다.

제 1 절 특별한 핀들

설계 검토를 수행하는 동안 발생하는 가장 일반적인 문제점 중의 하나는 특별한 핀들의 적절치 못한 종단처리(termination)이다. 모든 디바이스들에 대하여, 각 특별한 핀들의 종단처리가 적절하게 되어있는지 데이터 시트 및 설계 회로도들을 주의 깊게 검토해야 한다. 이러한 특별한 핀들의 종단처리는 시험에 의해서 증명되지 않을 수도 있다.

1. 모드 핀

.Actel 디바이스의 초기 세대에 존재하는 이 핀들은 반드시 접지되어야 한다. 이 핀들은 하드 접지(hard ground)가 인스톨된 디폴트 셋팅을 하고, 10 kohm의 저항과 하드 점퍼(hard jumper)를 접지와 병렬로 연결하는 것을 권고한다.

2. JTAG 인터페이스

현대의 많은 디지털 마이크로 회로는 JTAG 인터페이스를 갖고 있다. 높은 신뢰성 설계에 요구되는 하나의 옵션 핀은 TRST*이다. 이 핀이 존재한다면, 하드 접지되어야 한다. 왜냐하면 IEEE 1149.1 스펙은 부품의 안에 풀업 저항을 요구하기 때문이다. 모드 핀에서 사용했던 것처럼, 풀다운 저항을 사용하게 되면 TRST* 핀의 입력 전압이 로직 threshold 상태 또는 이를 초과하는 상태에 있을 수 있다. 만약TRST* 핀이 존재하지 않는다면 TMS가 로직 1을 유지하고, TCLK는 자유롭게 동작하는 독립적인 시스템 클록이여야 한다. TCLK의 입력으로 시스템 클럭을 사용하면 안된다. 왜냐하면 오작동을 하는 동안 칩의 동작 클럭 입력이 출력으로 되거나, 클럭을 클램프(clamp) 할 수도 있다.

3. 사용하지 않는 입력

일반적으로 모든 디바이스들은 적절하게 종단된(terminated) 입력을 갖고 있어야만 한다. 일반적인 CMOS 디바이스들에게 이는 필수사항이다. FPGA와 같은 프로그램 가능한 디바

이스들은 마이크로 회로의 프로그램 가능 특성을 활용하여 소프트웨어를 통해 사용하지 않는 핀들에 신경써야 한다. 예를 들면, Actel SX와 SX-S에서의 HCLK나 global routed 클럭과 같은 클럭 입력은 출력단을 갖고 있지 않다(그것들은 특별한 목적이다). 따라서 사용자가 반드시 적절하게 종단시켜야 한다. 만약 적절하게 종단되지 않는다면 과전류가 흐를 것이다. 또 다른 예로는, 사용하지 않는 LVDS 수신 입력들은 UTMC 문서에서 권고했던 바와 같이 연결되지 않은 채로 남겨놔야 한다. 디바이스들에 따라서 N/C라고 라벨링 된 핀들은 내부적인 용도로 사용될 수도 있다. 따라서 보드상에서 그 핀들을 종단시키는 것은 문제를 일으킬 수 있다. 역으로, 어떠한 경우에는 N/C 핀들을 종단시키지 않은 것이 나쁠 수도 있다. 스펙에 따라서 각 핀들을 주의 깊게 체크해라. 그리고 필요하다면 제조업자에게 연락해라.

4. 테스트 인터페이스

많은 디바이스들이 custom 테스트 인터페이스를 갖고 있고, case-by-case 토대로 다뤄져야만 한다. 이 인터페이스들은 테스트 장비와 연결되기 때문에, 제조사의 지시에 주의 깊게 따라야 한다. 예를 들면 Actel SX-S 디바이스의 테스트 핀들은 직렬로 종단되어야만 한다. 입력과 같은 다른 디바이스 핀들은 종단되어야 한다 다른 것들은 내부적인 종단을 갖고 있다. 일반적인 규칙은 없다.

5. 구성(Configuration) 핀들

각 구성 핀은 최신의 데이터 시트에 따라서 주위 깊고 확실하게 체크되어야 한다. 몇몇 핀들은 내부적으로 매우 높은 풀업 저항들을 갖고 있고, 보드 차원에서 높은 속도의 신호에 의해서 교체(is switched)될 수도 있다 방어적으로 설계해라. 그리고 레벨이 변함없음(solid)을 확실히 해라. 또한 몇 몇 구성 핀들은 본래 명시적으로 동작하면서 원하는 상태로 플로팅될 수도 있다. 마지막으로 비행을 위해서 적절하게 종단되어야 하는 프로그램 가능한 핀들과 같은 특별한 핀들에 대해서 주의해라.

6. 다른 사항들

다른 디바이스들은 다른 핀들을 갖고 있을 것이며, 결정적이며 일반적인 규칙은 없기 때문에 각 핀들은 반드시 체크되어야만 한다.

제 2 절 입력 출력

1. 동시적인 스위칭 출력

때때로 한 번에 스위치 할 수 있는 출력 핀의 수가 제한될 때가 있다 이러한 것들은 데이터 시트에 명시되어 있다 가끔은 어플리케이션 노트(application not)에 설명되어 있다 혹은 설계자가 자신의 것에 남겨둘 때도 있다. 빠르게 스위칭하는 디바이스들, 많은 수의 핀들, 낮은 AC, DC 잡은 마진은 접지면/전원 떨림 현상(Ground/Vdd bounce)이 심각한 문제가 될 수도 있다. 이 떨림 현상을 최소화하는 몇 가지 지침과 그 현상을 고려할 수 있는 몇몇 항목들이 있다.

(1) 만약 필요하지 않는다면, 낮은 슬루(slew)를 갖는 출력의 사용할 것

(2) SSO들을 그룹화하지 말 것 그것들을 분해하라(Xilinx: two for each side of a ground pin)

(3) 순서를 매겨 SSO들의 개수를 제어해라. (Example: Do address and data busses need to switch at the same time)

(4) 몇몇 집합체(families)에 대해서, 만약 보드 상에서 레일(rail)에 종단되어 있다면, 프로그램된 출력 단은 접지 혹은 전원공급을 개선시킬 것이다.

(5) 큰 메모리 배열이나, 긴 라인의 메모리 배열에는 특별히 버퍼를 사용해라. 모든 것들이 FPGA나 ASIC 안에 있을 필요는 없다.

(6) 소켓을 피해라.

(7) 여분의 패드 위치에 대해서는, 미리 전원, 접지, 바이패스 커패시터를 배선해라. 뒤엉킨 전원, 접지 연결은 불필요한 인덕턴스 성분을 갖게 된다.

(8) 입력 threshold을 현명하게 선택해라.

- TTL VIL = 0.8V (매우 민감하다. 이 세팅은 접지 떨림 현상(bounce) 및 상승(rising) 에 모두 민감하기 때문에 피해라)
- 몇몇 디바이스들은 프로그램 가능한 5V CMOS나 다른 입력 전압 threshold 옵션을 제공한다. 이는 접지 떨림(bounce) 현상을 감소시키는 것이 아니라 접지 바운스(bounce) 효과를 완화 시킨다.

(9) 클럭과 접지 떨림을 일으키는 핀과는 물리적으로 떨어뜨려라.

(10) 클럭과 접지 핀과는 가깝게 유지해라.

(11) When Using JTAG and driving board with test data over multiple parts you can induce data pattern sensitivities, particularly with large data busses, perhaps switching patterns from FFFFFFFF to 00000000. 이는 테스트를 위한 인위적인 것으로서, 인위적인 실수이다. 그러나 제어가 되지 않을 경우 하드웨어에 손상을 주거나, 잠재적으로는 과도한 스트레스를 줄 수 있다. 엔지니어링 모델과 항공 하드웨어에 대한 접지 떨림 테스트를 위해서 JTAG나 재구성 가능한 컴포넌트들이 사용될 수 있다.

(12) 테스트 케이블링은, 특히 진동, 열/진공, EMI 테스트를 위한, 정상적인 벤치 테스팅이나 혹은 시스템 어플리케이션에 대한 다른 조건들을 제시할 것이다. 전체 프로젝트 흐름에 대하여 최악의 상황에 대해 계획해라.

(13) Hard-wired된 고속 부품들은 흔히 전원과 접지와의 연결이 최적화되지 않게 연

결되어 있거나 불충분한 바이패싱이 되어 있을 것이다. 이에 대하여, 접지 떨림 효과를 관찰하면서 세심하고 적절하게 테스트가 이루어져야 한다.

(14) 많은 디바이스들에 대하여, tPD는 많은 동시 스위칭 출력(Simultaneous Switching Output)들에 의해서 부정적인 영향을 받을 수 있다.

(15) 접지/VDD 떨림은 시스템 잡음 마진을 줄이면서 다이나믹하게 입력 스위칭 문턱값(Threshold)에 영향을 줄 수 있다.

2. 신호 종단

신호들이 적절하게 종단되어 있는지 확실하게 해라.

(1) 클록 신호들은 문턱값(threshold)을 통해 부드럽게 변화하기 위해서는 특별한 주의가 요구된다. 적재된(loaded) 클록의 경우, 아마도 오래 동작되면, 거짓 혹은 두 개의 클록킹을 일으키는 비단조적인(non-monotonic) 변화을 초래할 것이다. 이는 비활동성의(inactive)의 에지에서 일어날 수 있음에 주목해라. 유사하게 오버슈트와 상승(rising) 또한 잘못된 클록킹, 특히 접지에 대한 변화를 일으킨다.

(2) 대부분 제조업자들은 신호가 지나가는 레일의 외곽이 얼마나 많이 떨어져 있는가에 대한 엄격한 제한들을 갖고 있고, 때로는 외곽에 대한 권고된 제한 사항이 최대 시간과 연관되어 있다. I/O 단자에 손상을 줄 수 있기 때문에 품질 좋은 신호임을 확실히 해라.

(3) 미리 저항의 종단처리에 계획을 세워라. 미리하지 않고, 나중에 종단저항을 추가하는 것 자체가 고생스러운 일이고, 이것은 회로 보드상에 위치에 있는 선들과 컴포넌트들보다 품질면에서 좋지 않을 것이다. 그리고 조립하는 사람에게는 매우 큰 어려움을 줄 것이다.

(4) 하나의 예로, RS-422과 같은 인터페이스에 적절한 종단처리가 되었는지 스키메틱을 조사해라. 종단저항 저항이 없다면, 비록 잡음 마진이 감소되거나 시스템의 대한 스트레스가 증가할지라고, 어느정도는 시스템이 동작하겠지만, 테스트를 하면서 발견하기는 어렵다.

3. 삼상태 버스에 대한 고려

동작되고 있는 삼상태 버스들에 대해 어떠한 오버랩도 허용해서는 안된다. 이는 전력을 낭비할 것이고, 불필요하게 잡음을 발생시키고, 컴포넌트들에게 스트레스를 줄 것이다. 최악의 경우에 버스의 드라이버들 사이에 off-time을 보장해라. 버스가 플로트(float) 되거나 늦은 과도 시간들을 갖지 않게 해라. 왜냐하면 전력소모와 잡음을 증가시키고, 신뢰성에 부정적인 영향을 줄 수 있기 때문이다.

4. 입력 변경 시간들

몇몇 고속 혹은 현대의 디바이스들은 입력 변경 시간들에 대한 매우 엄격한 제한을 갖고 있다. 종종 이러한 제한 사항들은 놀라울 정도로 엄격하다. 이러한 요건을 만족시키지 못하면 오실레이션(oscillation), 다중 클록킹(multiple clocking)이나 손상을 줄 수 있다. 현대의 컴포넌트들은 종종 수백의 나노초의 과도 시간 을 갖는 풀업 저항 혹은 풀-다운 저항들을 수용하지 않을 수도 있다. 컴포넌트들이 종종 수용하지 않을 수도 있다. 이러한 경우에, 버스 홀드(bus hold)나 소프트 래치 회로(soft latch circuit)들을 사용해라. 이것들은 또한 전력 소모도 줄일 것이다. 다른 경우, 몇몇 오래된 디지털 논리 군(family)들은 너무 느린 과도 시간을 갖고 있어서 현대의 디바이스들과 호환되지 않는 출력들을 갖고 있을 수 있다.

파형 측정은 보통 10%에서 90%이다. 그러나 항상 이런 것은 아니다; 때때로 파라미터 측정 방법이 명시되지 않는다. 신호가 보드나 박스로 들어가는 경우에 보호 회로를 사용할 때는 세심한 주의가 반드시 필요하다. 실험실에서의 테스트를 통해서 모든 자격을 갖춘 회로들이 데이터 시트를 만족하는 것이 아니라는 것을 알게 된다. 하나의 예로, 부품이 "shrunk" 되고, 관찰된 오실레이션으로 더 빠른 프로세스로 이동하는 것이다. 따라서 마진에 대한 신중한 고려를 권고한다.

5. 단락된 출력들(Shorting Outputs Together)

이는 때때로 보드의 드라이브를 증가시키기 위해서 사용된다. 하지만 이것은 피해야 한다. 왜냐하면, 만약 스위칭 스피드가 매치되지 않았다면, 컴포넌트들에게 해를 끼칠 수 있기 때문이다. 그리고 이러한 중복 토폴로지(redundant topology)를 테스트 하는 것이 어렵거나 비현실적이 될 수도 있다. 만약 이러한 테스트가 필요하다면, 두 개의 출력들은 동일한 IC상에 있어야 한다.

6. 핀 할당

핀 할당에는 세심한 주의과 계획이 중요하다. 첫째, 동시 스위칭 출력들(SSOs)과 잡음 면역성(noise immunity), quiet 설계에 대해 고려해라. 클록과 중요 신호들은 접지 핀 근처에 놓거나, 접지핀으로 둘러쌓이게 하여 최소한의 크로스 톡(cross talk)을 갖는 짧은 길이의 배선이 PCB(Printed Circuit Board)에서 이루어지도록 주의해라. 모든 데이터 비트들이 일렬로 버스위에 잘 정렬되어 있는 핀 할당은 PCB의 접지 떨림 문제와 FPGA의 내부 배선 문제를 일으키는 것으로 유명하다.

7. 혼합 전압 인터페이싱, DC 호환성, 잡음 마진

여러 개의 군(families)에서 디바이스들을 혼용하여 사용할 때, 디바이스들이 같은 제조업자라 할지라고, 디바이스가 신뢰성 있게 동작하고 있고, 충분한 잡음 마진이 있는지 세심한 주의가 요구된다. 이는 회로 성능을 업그레이드 하거나 노후된 부품을 교체할 때 문제가 될 수 있다.

입력의 경우, 많은 CMOS 기술이 적용된 디바이스 업체들은 TTL 호환이 되는 입력들이라고 광고를 한다. 그러나, 이러한 입력들은 사실 그것들의 TTL 상대(counterpart)하고는 많이 다르다. 모든 디바이스들이 그런 것은 아니지만, 가장 큰 차이점은 전력이 끊겼을 때, 인터페이스를 위해서 나타나는 임피던스이다. 예를 들어, SEU 때문에, 방사 경화된 (radiation-hardened) CMOS 래치들로 대치되었다. Galileo 고도 제어 컴퓨터의 메모리 유닛에 soft 54LS373이 있는데, 블록 여분(redundancy) 회로가 고장이 났다. 왜냐하면 엔지니어들이 전력이 제거되었을 때, 입력 ESD 보호 다이오드들을 통한 스니크(sneak) 경로를 고려하지 않았기 때문이다. 다른 연관된 차이점은 적용될 수 있는 최대 전압이다. 높은 전압에서 낮은 전압으로의 신뢰성 있는 레벨 시프팅을 위해서 몇몇 바이폴라(bipolar) 디바이스들이 사용된다. CMOS 대체 다바이스들은 보호 다이오드에 포워드 바이어스를 사용할 것이다. 그러나 이것은 의도되지 않은 전류 흐름과 회로의 손상 혹은 고장을 일으킨다. 마지막으로 많은 CMOS 입력들은 논리 문턱값(threshold)를 갖고 있는데, 이것은 TTL과 호환되지 않는다. 즉, TTL의 VIH 스펙(specification)을 종종 만족하지 않는다. VIH 최대값은 2.2V, 2.4V, 때로는 2.5V가 명시되어 있지만, 진짜 TTL 디바이스들은 두 개의 다이오드의 드롭 전압에 의해서 정의되는 문턱값(threshold)을 갖고 있다. 이들 범위는 1.2V-1.4V이다. TTL 출력들은 단지 VOH = 2.4V를 드라이브하기 위해서 보장되어 있다. 따라서 이러한 상황에서 거의 없거나 혹은 부정적인 잡음 마진이 나타날 수도 있다. 스위칭 포인트 차이는 신호 무결성에 따라서 회로의 고장을 야기할 수 있다. 스위칭될 때, 종종 TTL 출력들은 특히나 무겁거나 긴 로드(load)를 갖는 파형아네 “bump”를 지니고 있다. 이러한 "bump"는 종종 TTL 디바이스들이 올바르게 동작하기 위하여 충분히 높은 전압 상태에 있지만, CMOS 디바이스들의 더 높은 VIH는 다중(multiple)클록킹을 일으킬 수 있다. 이러한 상황에서 만약 수동 회로가 안정화 되기 위한 충분한 시간이 존재한다면, 풀업 저항들은 DC 잡음 마진을 복구할 수 있다. 그러나 이러한 방법대로 설계된 TTL ?/font> CMOS 클록 인터페이스는 종종 실패할 것이다. 왜냐하면 CMOS 입력은 더블 클록킹을 일으키는 다중 변화(Mutiple transition)를 경험할 수도 있다.

CMOS 출력 단 또한 고장이 잘 날 수 있다. 그리고 미묘한 디바이스 특성들이 에러를 일으킬 수 있다. 모든 명세서를 주의 깊게 체크해라! 예를 들어, 로드(load)를 드라이빙할 때, 많은 CMOS 디바이스들은 높거나 혹은 논리 1 신호를 위해 매우 낮은 전류 레벨에 명세되어 있다. 그러나 TTL 입력들은 상당한 양의 전류를 취급한다. 그리고 CMOS FET 입력에서 바라본 높은 임피던스를 드러내지 않는다. 그리고 출력이 끌어 내려질 수 있다. CMOS 와 TTL 입력이 혼합된 출력 로드(load)의 경우, CMOS 입력, 전형적으로 70% of

VDD,에 필요한 높은 전압과 2.0V의 VIH의 낮은 전압을 갖는 TTL 입력에 필요한 높은 전류를 보증하기 위해서 반드시 분리되어야 한다. 고려할 다른 사항은 CMOS 디바이스의 출력 단의 구조이다. 예를 들어, 몇몇 디바이스들은 높은 레일로의 스윙하지 않을 것이다. 그리고 제한된 전압을 갖고 있다. 만약 다음 입력단에 p 채널 FET가 cut off 상태에 있지 않는다면, 이는 약간의 totem-pole 전류를 일으킬 수 있다. RT54SX와 같이 I/O가 5V를 공겁할 수 있는 몇몇 디바이스들은 3.3V의 코어 전압에 대한 출력들을 드라이브 할 수 있을 것이다. 그러나 이로 인해 CMOS 출력은 같은 보드 상에 있는 5V의 CMOS입력과 호환되지 않는다. 5V I/O 바이어스가 공급될 때, 위에서 언급된 디바이스들은 5V 전압의 스윙을 갖는 2.4V 코어 RT54SXS 시리즈상에서 고정될 것이다.

오늘날 컴포넌트들은 전형적으로 1.5V, 1.8V, 2.5V, 3.3V, 5.0V등과 같이 많은 공급 전압을 갖고 있다. 또한 잘 프로그래밍 할 수 있는 최신의 디바이스들과 함께 풍부한 I/O 표준들이 있다. 따라서 이러한 것들의 특성은 분명하고 심지어 회로도까지 잘 알려져 있다. 따라서 I/O 호환성은 반드시 주의 깊게 증명되어야 한다. 특히 신제품이나 개선된 디바이스나 대체 디바이스로 교체될 때 주의해야 한다.

8. 전력 스위칭과 Cold Sparing

여유적인 측면(redundancy) 혹은 전력 절약 모드를 위해 독립적으로 전원공급이 되는 블록들로 구성된 시스템을 설계할 때는 반드시 상당한 주의를 기울여야 한다. 진성(intrinsic) 혹은 ESD 보호 다이오드를 통해 전력을 낮출 때, 대다수의 CMOS 디바이스들은 낮은 임피던스를 보인다; cold sparing을 갖는 다른 디바이스들은 동작하기에 적합한 높은 입력 임피던스을 갖고 있을 수도 있다. 하나의 예로서, 3.3V PCI 호환성(compatibility)을 선택하게 되면, "Cold sparing" 디바이스는 더 이상 높은 임피던스를 지니지 않는다. 왜냐하면 클램핑 다이오드가 가능하기 때문이다. 프로그램 가능한 디바이스의 경우, 많은 바이폴라 디바이스들이 cold sparing 구조에 호환되지만, 몇몇 디바이스들은 출력을 통하여 VCC에 대한 스니크 경로(sneak path)를 갖는다.

제 3 절 클럭

클럭킹, 유한 상태 머신(Finite state machine) 그리고 시간 분석은 서로 밀접하게 연관되어 있어야 한다. 이 단원은 클럭과 설계 기준에 대한 몇몇 체크 사항에 대해서 논의할 것이다.

1. 스큐가 없거나 낮은 스큐를 갖는 클럭의 사용

처음에 프로그램 가능한 로직을 이용하여 설계를 할 때, 대다수 설계자들은 이산(discrete) 디지털 마이크로 회로에서처럼 제로 스큐 패스(zero-skew paths)라 가정하고 보통 라우팅 리소스(routing resource)를 사용한다. 그러나, 스키매틱 상에서의 망이나 HDL 텍스트에서의 클럭 신호가 일정한 신호처럼 보일지라도, 망의 전기 설계는 분석되어야 한다. 일반적으로 설계자들이 스큐가 없거나, 낮은 스큐의 리소스를 사용할 때, 설명할 수 없는 몇몇 글리치들(glitches)이나, 구체적인 라우팅에는 문제가 없지만, 빈약한 "프로그램 생산물(programming yield)"들로 더 잘 동작하거나, 잘못 동작할 수도 있다. 그래서, 일련의 서로 이웃해 있는 플립-플롭들이 서로 공통의 에지 클럭을 사용할 때, 낮은 스큐를 갖는 클럭을 반드시 사용해야 한다. 스큐가 없거나, 낮은 스큐를 갖는 클럭을 이용하여 설계를 하는 것은 수용할 만한 일이다. 그리고 이로 인해서 전력 소모의 감소나 사용 가능한 클럭 개수가 효과적인 증가된다. 그러나, 스큐에 내성이(skew-tolerant) 있는 설계 기술 및 분석이 주의 깊게 사용되어야 한다.

2. 칩과 칩 사이의 시간 전략

많은 분석 툴이 단일 칩에 대한 로직 분석에는 능숙하다. 그러나 대부분 시스템이나, 칩과 칩 사이의 시간 분석에 대해서는 효과적이지 못하다. 다양한 디지털 디바이스들을 연결시키기 위해서(hook up) 보드 상에서 단지 낮은 스큐를 갖는 클럭을 사용하는 것은 매력적일 수 있다. 그러나 이것은 잘 동작하는 것을 항상 보장하지는 않는다. 그리고 적절한 분석이 반드시 이루어져야 한다. 클럭 주기에 기초들 둔 시간을 셋업하는 것 이외에도, 대기 시간(hold time) 분석이 반드시 이루어져야 한다. 이것은 종종 빠뜨리거나, 적절치 못하게 수행한다. 소스 칩의 클럭과 출력간 최악의 상황이 "최소" 혹은 "최상의 경우"의 시간 파라미터를 사용하여 쉽게 분석되는 동안, 싱크(sink) 칩의 대기 시간(hold time)은 동일한 계산을 위해서, 클럭의 slow path와 데이터의 fast path를 가정하여 반드시 분석되어야 한다. 자동화된 툴들은 모든 최소 혹은 최대값에 대하여 분석할 수는 있지만, 동시에 최소-최소 분석은 할 수 없다 종종 사람이 이를 완성해야 한다. 싱크 칩에 대한 좋은 목표는 0ns 혹은 음의 대기 시간을 갖는 것이다. 그러나, 많은 디바이스들 특히, FPGA의 몇몇 모델들은 이 조건을 만족하지 못한다. 그래서, 대안의 기술로 반대의 에지 클록킹이나, 비동기에서처럼(완벽하진 않지만) treating 신호를 반드시 사용해야 한다. 통과(passing) 기준은 모든 최악의 경우의 셋업과 대기 시간들이 항상 만족되거나, 충분한 메타스테이블(metastable) 상태의 보호가 포함되어야 한다는 것이다. 특정한 스피드 등급으로 표시된 몇몇 부품(part)들은 표시된 것보다 더 빠를 수 있음에 주목해라. 그리고 가장 빠른 군(family)에서의 최소값이 최악의 경우의 분석에 이용되어야 한다. 많은 칩들의 binning criteria는 single ended 이다 sample timing path는 몇몇 threshold를 초과해서는 안 된다. 이는 스피드 등급인 디바이스들 혹은 " slower"라 표시된 두 faster에 대해서는 공통사항이 아니다. 왜냐하면 디바이스에 표시된 것들은 계약상 요구사항이고 "faster"라고 표시된 디

바이스들은 이후의(incoming)의 검사 동안 수용되지 않기 때문이다.

3. 클럭 트리

1) DLL과 PLL의 사용

디지털 시스템에서 DLL과 PLL은 많은 유용한 기능들을 갖고 있다. 그러나 그것들은 만족해야 할 몇몇 요구사항들이 있다. 첫째, 최악의 경우의 주파수들은(가장 빠른 것과 가장 느린 것) 회로와 호환성이 있는지 체크해야 한다 종종 수용 가능한 범위들은 매우 제한적이다. 또한, 반드시 만족되어야 하는 신호 quality 조건이 있다. 다음으로, when these circuits clock finite state machines, or other sequential logic, 이러한 회로들을 안정화 시키고, 락(lock)시키는 시간에 주목해라. 그리고 디바이스와 시스템에 전원이 안전하게 공급되는 것을 확실히 해라. 그리고 또 다른 항목은 DLL 혹은 PLL이 SEU에 의해서 타격을 받았을 때, 최상의성능을 체크하는 것이다. 이로 인해 DLL 혹은 PLL의 프로그램상의 변화가 발생할 수 있고, 이는 때때로 약간의 미묘한 모드(mode)상의 변화를 초래할 수 있다. 시스템의 안전 기능은 반드시 이러한 off-nominal 조건에서 수행되어야 한다.

2) 각 오실레이터에 대한 하나의 root

다이어그램은 각 오실레이터의 하나의 root와 함께 회로에 대한 클럭 트리를 보여줘야 한다. 이러한 다이어그램은 클럭 도메인 상에 있는 PLL, DLL과 칩과 칩들 사이에 ""talk""하는 모든 칩들을 포함해야 한다. 후자의 경우는 비동기 신호의 확인을 위해서이다.

3) 클럭 도메인을 교차하는 로직 블록과 신호의 표시

클럭 트리의 분석을 토대로, 클럭 도메인을 교차하는 모든 블록과 신호들을 확인해라. 그리고 메타스테이블(metastable) 상태의 resolution의 필요성을 정해라. 추가적으로, 신호 동기화에 포함되는 latency를 시스템이 허용할 수 있음을 확실히 해라.

4) 반대 에지 클럭킹에 대한 순환 주기(duty cycle) 분석

클럭의 하나의 에지에서 다른 에지로 넘어가는 데이터의 설계에 있어서, 각 단계(phase)의 가장 최악의 경우의 순환 주기(duty cycle)가 적절하게 계산되었는지 확실히 해라. 상황이 그렇지 않음에도, 종종 설계자들은 50%의 순환주기(duty cycle)를 가정할 것이다. 순환주기(duty cycle)의 왜곡에 대한 원인은 보통 50 +/-10%의 순환주기를 갖는 오실레이터의 특성 때문이다 로직 게이트 및 버퍼 등을 통과하면서 생기는 고르지 못한 delay들.

4. 비동기 인터페이스와 메타스테이블(Metastable) 상태에 대한 실패율 계산

각 비동기 신호를 위해서 적절한 동기 장치를 확실하게 사용해라. 종종 설계자들은 단순히 두개의 직렬 D 플립-플롭들을 사용할 것이다. 자주 사용되고 수용할 수 있는 토폴로지(topology)라 할지라도, 매우 고속의 회로(검토 중인 기술을 위해 사용될)에 있어서 이 동기 장치의 실패율은 무시하지 못한다 이러한 상황을 고려하여 반드시 실패율이 계산되어야 한다. 또한 평상시(nominal) 온도와 전압에서 극한의 상황으로 변할 때, resolution time의 가능성 있는 큰 변화에서 메타스테이블(Metastable) 파라미터가 취하는 조건들에 주목해라. 이러한 것들은 테스트를 하거나 증명하기에 비현실적이기 때문에, 회로상에서 충분한 마진이 게 끔 설계해라. 또한 ASICs, 다른 플립-플롭 macros는 다른 메타스테이블(metastable) 파라미터를 갖고 있을 수 있다는 사실에 주목해라. 이는 FPGA에서도 고려되어야 한다. 메타스테이블 상태(Metastable)에 강한("Metastable state hardened") 몇몇 개별(discrete) 다비이스들이 사용된다. 현대의 플립-플롭들은 꽤 성능이 좋지만, 아직 매혹적인 해결책을 갖고 있는 것은 아니다.

제 4 절 유한 상태 머신(Finite State Machines)

유한 상태 머신 설계는 종교적인 이슈가(religious issue) 될 수 있다. 유한 상태 머신을 위한 최고의 스타일은 무엇인가? 인간 혹은 머신이 상태 할당을 해야 하는가? 유한 상태 머신을 안전하게 어떤 방법으로 설계할까? 이러한 모든 상황에 대한 최고의 해답은 없다. 그리고 체크될 수 있는 매력적인 스타일도 없다. 많은 엔지니어들이 상태 머신을 설계하기 위해서 로직을 결코 보지 않고, HDL을 사용한다는 점에 주목하길 바란다. 이는 중대한 어플리케이션(application)에서 반드시 극도의 관심을 갖고 행해져야 한다.

높은 신뢰성을 요구하는 회로에 대한 몇몇 설계 지침들은 one-hot 상태 머신을 사용하지 말 것을 요구한다. 왜냐하면 많은 수의 플립-플롭은 SEU(Single Event Upset)의 가능성을 증가시키기 때문이다. 이것이 진실일지라도, 상태 머신에 대한 신중한 분석을 통 one-hot 토폴로지(topology)는 모든 단일 비트 에러를 검출하면서 2의 Hamming distance를 갖고 있음을 알아냈다. 이와 비교하여, 동일한 binary coded 상태 머신은 어떠한 lockup state를 갖고 있지 않으면서, 1의 Hamming distance를 갖고 있다. 만약 2n의 상태가 모두 사용된다면, 여분의 체킹 머신없이 규칙에서 벗어난 변이(transition)을 찾아 낼 수 없다. 그리고 사용된 여분의 체킹 머신 그 자체도 SEU와 연관되어 있으며, SEU와 함께 lockup state을 갖게 될 것이다.

1. Lockup 상태의 분석 및 전략

1) 스키매틱에 기반을 둔 머신

중대한 일을 처리하는 상태 머신에 있어서, 모든 가능한 로직 상태에 대해서 반드시 분석되어야 한다. 그리고 머신이 결정적이고 원하는 방식대로 동작함을 증명해야 한다. 또한 예상에서 벗어난 사건들에 대해서도 반드시 고려되어야 한다. 이러한 예가 바로 SEU(Single Event Upset), credible failure mode이다. 많은 소비자 등급의 IC에서 유한 상태 머신(Finite State Machine)들은 SDRAM과 같이 lockup 상태를 갖고 있다. 중대한 일을 처리하는 컨트롤러는 이러한 것을 수용할 수 없다. 분석은 가능한 모든 2n 상태를 포함해야 한다. 전원 버스의 장애나 ESD로 인해 lockup 상태로 되는 것이 A credible failure mode이다. 어떤 고 신뢰성의 머신이든, 모든credible failure mode에 대하여 반드시 강인한 특성을 지니고 있어야 한다. 추가적으로, FSM이 원하는 시퀀스를 통해서 과도기(transition)보다는 정상적인 상태(legal state)에 시작하도록 확실히 해야 한다. 이를 위한 한가지 방법은 POR(Power-on reset) 지시기를 사용하는 것이다. 이것은 클럭과 동기화되도록 반드시 체크되어야 한다. 모든 플립-플롭에 POR이 비동기 혹은 리셋 입력으로 반드시 사용할 필요는 없다. 이것은 리셋 신호 장애에 대한 부하를 증가시키고 고속 회로에서 사용되는 모든 플립-플롭들에 대한 제거 시간(removal time)을 만족시키기에는 힘들기 때문에 많은 경우에 있어서 바람직하지 못하다. 만약 FSM이 항상 원하는 상태로 동작한다면 어떠한 리셋(reset)도 필요하지 않을 수도 있다. 예를 들면, n 마스터 카운터에 의한 분할(divide)과 같은 경우에서 흔하게 볼 수 있는데, 여기에서 리셋(reset)은 불필요하고 리셋상의 결함으로 머신은 정지될 수 있다. 리셋 기능에 대한 한가지 고려사항은 테스트를 위한 설계(design-for-test), 시뮬레이션을 위한 설계(design-for-simulation)이다. 왜냐하면 이것들로 인해 종종 추가적인 리셋 연결이 나올 수 있기 때문이다.

2) HDL로 합성된 머신

분명히 스키메틱(schematic)에 기반을 둔 머신에 대한 모든 기준이 여기에도 적용된다. 그러나 HDL을 이용한 설계에 대해서는 특별한 고려사항이 있는데, 이는 CAE(Computer Aided Engineering) writer가 고 신뢰성을 요구하는 회로들에게 바람직하지 못한 회로를 제공할 수 있다는 점이다. 따라서 중대한 회로에 대해서는 합성기(Synthesizer)가 제공하는 출력 리포트를 반드시 매우 신중하게 검토되어야 한다. 체크할 공통 사항은 다음과 같다: lockup 상태 글리치(glitch)를 일으킬 수 있는 Gray로 인코드(encode)된 머신의 출력물: 의도되지 않은 플립-플롭 복제 원하는 대로 스타일이 명세되지 않음(종종 합성기는 사람보다 더 잘 안다고 판단하여 다른 스타일로 대체할 것이다.) 추가적으로, 몇몇 로직 합성기들은 안전한 상태 머신들을 생성할 것이다. 생성된 설계에 대하여 면밀히 검토해라. 예를 들어서, 때때로 로직이 게이트(gate)를 폭발적으로 증가시킬 수 있음이 보고 되었고, 클럭의 반대의 에지에서 리셋 신호가 생성되기도 한다. 이로 인해 설계자에게는 보이지는 않

지만, 클리어(cleaer)의 제거하는 시간을 촉박하게(tight) 만든다. VHDL과 같은 언어는 모든 물리적 상태를 다루지 못하고 단지 논리적인(logical) 상태만을 다룬다는 것을 명심해라. 따라서 "others" 항은 물리적인 구현이 아니라, enumberated type에서의 상태를 언급할 것이다. HDL은 one-hot인지 gray 코드인지, 어떤 플립-폴롭이 복제되어야 하는지 모른다. 이것은 블랙 박스 시뮬레이션 뿐 아니라, 논리적 등가(logical equivalence)에 의한 부울식(Boolean equations)에 의해서도 발견되지 않는다.

2. 플립-플롭 복제

논리 설계자들은 신뢰성 혹은 성능 측면에서 로직을 복사한다. 예를 들어서, 만약 출력단에 부하가 너무 높으면, 부하는 종종 다중(multiple) 드라이버들 사이에서 분리될 것이다. (몇몇 경우에 있어서, 출력은 하나로 합쳐지지만, 이는 선호할 사항은 아니며, 보통 피해야 한다.) 다른 경우에는 부하를 자르고(cutting) 드라이버를 복제하는 것은 커패시티브(capacitive) 부하를 분산시킴으로써 시간을 만들 수 있다. 조합 로직의 복제는 꽤 간단하다.

그러나, 만약 이 개념이 시퀀셜(sequential) 로직과 유한 상태 머신(FSM)으로 확장된다면, 상황이 좀 까다로워진다. 왜냐하면 상태 정보가 포함되어 있기 때문이다. 실제로 로직은 다른 정보를 회로의 다른 부분에게 넘길지도 모른다. 예를 들면, 로직은 SEU, ESD등의 사건과 같은 과도기(transient) 결함의 현재 상태와는 일치하지 않을 수도 있다. 즉 논리적인 플립-플롭은 연결되는 물리적인 플립플롭에 따라서 회로의 다른 부분에 다른 값을 넘겨줄 수 있다. 이것은 고 신뢰성의 어플리케이션(application)에서 유의해야 할 사항이다. 소프트웨어 CAE 툴들이기쁘게도 이러한 종류의 회로들을 생성한다..(Software CAE tools are more than happy to generate circuits of this class) 그리고 자기-일치성(self-consistency)를 보증하는 로직은 생성하지 않는다. 예시들이 어플리케이션 노트(application note)에 있다.

3. 에러 검출과 수정에 대한 요건 및 구현

강한(robust) 상태 머신을 설계하기 위해서 단순히 해밍(Hamming) 코드와 수정 회로를 추가하는 것은 매력적인 일이다. 만약 SEU 혹은 ESD 사건들이 시스템 클럭에 동기화되지 않고, 로직 망이 글리치(glitch)가 없게 끔 보장되지 않는다는 것을 알았다면, 당신은 강건하게(robust) 동작하는 구조물의 능력에 대해서 재고해야 한다. 일반적인 경우, 다음 상태의 로직을 실행하는 조합 논리 회로들과 상태 레지스터를 만드는 플립-플롭으로의 입력들에 대해서 반드시 분석해야 한다. 특히 이러한 종류의 어떤 계획이든, 당신은 회로 구현이 정적 해저드(static hazard)이 있는지 없는지, 만약 없다면, 상태(혹은 set 상태)로 잘못된 변화(transition)가 만들어 질 수 있는지를 반드시 살펴봐야 한다.

제 5 절 리셋

1. 리셋 논리 회로(과도기와 정상상태에서의 동작 고찰)

리셋 회로에 대한 과도 기의 영향이 반드시 분석되어야 한다 실제로, 이것은 반드시 분석의 초점이 되야 한다. 전원의 어플리케이션(application)에 대하여, POR의 출력 혹은 리셋 회로는 이상적으로고체 논리 레벨(solid logic level)이 되어야 하고, 글리치(glitch)가 없어야 한다. 타이밍 커패시터로 쇄도하는 전류의 양은 그 커패시터 타입의 최대값을 초과해서는 절대 안된다. 만약 비교기가 사용된다면, 논리 게이트(gate)로의 상승 시간(rising timing)은 명세서에 표기된 입력 값을 초과해서는 안 된다 종종 히스테리시스(hysteresis) 입력을 갖는 게이트가 사용된다. 그러한 타입의 입력 조차도 출력 글리치(glitch)들이 발생할 수 있다. 그리고 논리 게이트의 여러 단이 반드시 사용되어야 한다. 고려해야 할 다른 과도기(transient) 요소는 비행을 위한 전원공급기(flight power supply)의 최상과 최악의 상승 시간(rising time)이다. 이러한 것들은 laboratory supplies에 따라서 상당한 차이가 나고, 비단조(non-monotonic) 이거나 상당한 오버슈트(overshoot)와 상승(rising)을 갖고 있을 수 있다. Flight power supplies는 전원 버스에서 전도성 방출(conducted emission)을 최소화하기 위한 제한된 슬루레이트(slew-rate)임에 주목해라. 전원 공급기의 시정수(time constant)는 POR 회로의 시정수 값을 초과할 수도 있다! 방전을 위해, 타이밍 커패시터 방전을 위해 낮은 임피던스(impedance)를 갖는 path를 확실히 확보하고, 논리 게이트의 입력이 확실하게 보호되도록 해라. 항상 그런 것은 아니지만, 대부분의 CMOS 입력들은 입력단에서 전원 공급 레일(rail)까지 ESD 다이오드를 갖고 있다. 그러한 입력을 통해 큰 커패시터를 방전시키는 것은 해가 될 수 있다. 또한 전원 버스에서의 순간적인 붕괴에 대한 회로의 응답과 요건에 대해서 고려해라. 대부분의 회로들이 power-on reset에 대해서 회복할 수도 있지만 모든 회로가 다 그런 것은 아니다. 이러한 예 중의 하나가 세심하게 보호가 필요하는 비휘발성, 지울 수 있는 메모리이다.

리셋 회로의 많은 다이어그램(Diagram)들은 리셋 회로의 "비동기 어플리케이션(application), 동기적 제거(synchronous removal)"를 보여준다. 그러나많은 디바이스, 특히 프로그램 가능한 디바이스 경우에 있어서, 전원 공급이 되는 과도기(transient)동안 입력이 막히거나 무시될 수 있다는 사실을 주목해라. 이는 전하 펌프(charge pump)가 시작되거나, 혹은 환경 구성(configuration)이 로드되고 릴리즈(release)되는 것에 대한 필요 때문일 수 있다. 동기화된 입력을 갖는 디바이스에 대하여, 아마도 클럭 오실레이터는 리셋 신호가 가해지기 전에 수십 밀리초의 시간을 반드시 필요로 한다. 이러한 디바이스들의 출력들은 시스템 레벨에서 반드시 다뤄져야 한다. 왜냐하면 리셋 신호의 경우 HDL 코드 혹은 스키매틱(schematic) 상에서는 괜찮아 보일지 몰라도, 실제 회로에서는 무시되기 때문이다.

정상상태 혹은 DC 영향 또한 중요하다. 타이밍 커패시터와 논리 게이트의 누설 전류를 체크해라. 왜냐하면, 타이밍 커패시터의 누설 전류의 총 양과 타이밍 저항의 저항 값과의 곱은 모든 잡음 마진(margin)을 제거할 수 있는 전압 드롭(drop)을 일으킬 수 있기 때문이다.

2. 트리(Tree)

모든 리셋 소스(source)들의 트리(tree)를 그리는 것은 리셋 논리가 잘 정의되도록 도와준다. 시스템 리셋, 소프트웨어 리셋, 와치독 타이머 등으로부터의 다양한 형태의 리셋 신호가 있는데, 좋은 트리 다이어그램(tree diagram)은 그들 사이의 관계를 잘 보여준다. 반드시 필요할 때에는 적절한 동기화를 만들어라. 또한 만약 리셋이 빠르게 동작되어야할 때에는 (예를 들어 잘못된 쓰기 신호로부터비휘발성 메모리를 보호하거나, 신호탄의 점화와 같은 one-time 사건의 개시로부터 다른 회로를 보호하는 것들) 트리(tree)로 인해 논리와 지연들이 반드시 잘 이해될 것이다.

3. 리셋 활동 시간 vs. FPGA와 크리스털 클럭 오실레이터와 같은 컴포넌트의 시동(startup)

보장된 리셋 시간은 시스템의 모든 회로에 대해서 반드시 충분히 길어야 한다. 이는 종종 빠뜨리기가 쉽다. 예를 들어, 많은 FPGA(FPGA에 있는 전하 펌프(charge pump)는 반드시 전압과 전하 내부 커패시턴스를 확립해야 한다.)들은 시작을 위한 시간, 지연에 대해 대기하는 시간들을 필요로 한다. 그리고 나서 출력이 나온다. POR 신호의 조급한 출력으로 인해 확인할 수 없는 상태가 될 수 있다. 내부적으로 많은 POR 회로들을 갖고 있는 FPGA들은 적절한 로딩(loading)과 릴리즈(release)에 대한 리셋 시퀀스(sequence)를 필요로 할 수도 있다. 이러한 모든 리셋에 대한 타이밍은 반드시 최악, 최상의 경우를 고려하여 분석되어야 한다. 또한 크리스털 클럭 오실레이터와 같은 디지털 논리 보드상의 몇몇 표준 컴포넌트들은 상당한 시간의(흔히 수십 밀리 초) startup시간을 갖고 있을 수 있다. FPGA나 크리스털 클럭 오실레이터와 같이 이러한 것들을 더욱 복잡하게 만드는 컴포넌트들은 전원 공급기의 상승시간(rise time)의 함수인 startup 시간을 갖고 있을 수 있다. 심지어는 흔히 이러한 행위들이 잘 명세되지 않는다. 강한(robust) 시작 시간들은 결정적이다.

4. One-time 사건들(즉 화약 기폭제(pyrotechnic initiation)) 혹은 중대 임무에 대한 신호들의 보호

이 주제는 부분적으로 6.2절에 언급되어 있다. 그러나 이러한 신호들의 중요성 때문에, 여기서 다시 언급한다. 많은 논리 요소들은 전원 공급기의 출력이 상승됨에 따라 그들의

진리표(truth table)에 맞게 행동하지 않는다. 따라서 POR 신호는 반드시 전원 공급기 rise time 과도기 동안 거짓(false) 신호를 막는 게이트로서 동작해야 하고, 그리고 나서 모든 회로들이 안정상태가 된 후 출력해야 한다. 이와는 반대로, 전원이 꺼질 때(come down), 논리 요소들이 전압 드롭(drop)으로 인해 제어(control)을 잃어버리기 전에 중요 회로들이 안전한 상태 있다는 것을 확실히 하기 위해서, POR 회로가 초기에 단정될(asserted) 필요가 있다. 흔히 보호를 필요로 하는 디바이들은 화약 기폭제(pyrotechnic initiator)들, EEPROM들, 플래쉬 메모리들, 기타 등이 있다. 마이크로컨트롤러(microcontroller)와 같은 몇몇 디바이스들은 내부적으로 플래쉬 메모리를 갖고 있어서 POR 신호들에 의해서 보호가 필요한 모든 컴포넌트들과 시스템 인터페이스들을 평가할 수 있다는 사실을 주목해라.

제 6 절 위험요소 분석

정적 위험 요소는 조합망(combinational network)으로 향하는 단일 변수로의 변화가 다른 변수들 사이에서 과도기(transient) 혹은 순간적인 변화가 일어날 때 존재한다. 그러나 이것은 일어나서는 안된다 (e.g., 1 -> 0 -> 1). 그러면 정적 위험 요소가 생겨난다. 정상적으로는, 신호가 정착하기에 충분한 시간이 있다면, 동기적 설계에서는 이러한 문제가 발생하지 않는다. 유사한 조건으로 만약 1-> 0 -> 1 -> 0 형태의 과도기가 존재한다면, 동적 위험요소가 존재한다. 즉 이들은 확실히 서로 바뀌지 않는다. 정적 위험요소가 없는 어떤 회로든지 동적 위험요소도 존재하지 않는다. 필수 위험요소는 논의에서 벗어난다. 이 주제는 많은 논리 클래스에서 다뤄지지 않는다. HDL들과 기능적 시뮬레이션을 이용으로 많은 설계자들이 이 컨셉과는 친숙하지가 않다. 단일 클럭과 공통의 에지를 사용하는 100%의 동기적 설계에서는 일반적으로 관심사항은 아니다. 그러나 검토가 진행되는 동안, 설계자들이 모른 채, 위험요소가 종종 존재한다. 이러한 예 중의 하나는 유한 상태 머신(FSM)에 클럭 신호를 생성하기 위하여 TMR 회로를 사용하는 것이다. 만약 voter가 위험요소를 지닌 상태에서 SEU들의 영향을 완화시키기 위해 voter에 대한 입력의 변화는 voter로부터 발생되는 글리치(glitch)로부터 더블 클럭이 생길 수도 있다. 보통 컴포넌트들은 위험요소가 없는 것처럼 보일지도 모른다 당신이 사용하고 있는 로직 군(family)상에서 잘 실행되는지를 반드시 주의깊게 살펴봐야 한다. 예를 들어 몇몇 FPGA 군의 기본인 멀티플렉서는 글리치가 없는가? 멀티플렉서가 글치기가 없다고 보증할 수 없다. 따라서 세심한 주의 없이 안전한 클럭 생성기라고 간주될 수 없다. 다른 예는 투표된(voted) 출력이 칩 밖으로 나와서 외부 다바이스의 클럭 입력으로 사용될 때이다. 툴이 동작하는 동안 엔지니어에는 알려지지 않은, 회로상에서 생성되는 위험요소를 만들어내기 위해서 논리 합성기에 대한 주의가 필요하다.

제 7 절 전원 시스템

1. 공급 시퀀싱(sequencing)

이와 관련되어 2가지 경우가 있다. 이 단원에서는 하나의 디바이스에 대한 여러 개의 전원공급에 관해서 논의할 것이다. 다음 단원에서는 여러 디바이스들에 대한 인터페이스에 대해서 논의할 것이다. 대부분 더 새로워진 디바이스들은 둘 혹은 더 많은 전원 공급기를 요구한다. 보통 이 공급기들은 논리 디바이스의 코어에 전원을 공급하는 것과 입출력 셀에 전원을 공급하는 것으로 나눌 수 있다. 추가적인 전원 공급이 PLL과 DLL, 특별한 I/O 표준 혹은 외부 전하 펌프와 같은 다양한 바이어스(bias) 공급에 필요할 수도 있다. 이러한 전원 공급기들은 모든 DC 기준뿐 아니라 리플(ripple) 특성(특히 PLL과 같은 회로)도 분명히 만족해야 한다. 분명하지 않은 것은 전원이 하나의 다바이스로 공급되는 시퀀스가 회로의 동작 및 성능뿐 아니라 신뢰성에도 영향을 미칠 수 있다는 사실이다. SX-S 시리즈와 같은 다비아스의 경우 논리 코어에 전원이 중단되기도 전에 I/O의 전원이 bring up되면, 큰 쇄도(inrush)전류가 존재할 수도 있다 만약 전원 공급 순서가 역으로 된다면, 해당사항은 아니다. 몇몇 다바이스의 경우, 잘못된 전원 공급 시퀀스로 인해서 디바이스에 overstress혹은 해가 될 수 있다. Multiple vendors가 이에 해당된다. 보통 시퀀스에 대한 요건은 어플리케이션 노트(application note)나 "fine print"에 나와 있다. 다음과 같은 사항이 고려되어야 한다. 오실로스코프 프로브(probe)를 "slip"하고 접지링(ground ring)이 순간적으로 전원선이나 커패시터 터미널에 접촉되거나, 순간적으로 전원에 단락 되었을 때이다. 이로 인해 잘못된 시퀀스가 발생될 수도 있고 이론상 특정 칩에 overstress나 해가 될 수도 있다. 일반적으로 전원 공급 시퀀스가 요구되는 부품은 피해라. 만약 그런 부품이 있을 때에는 플래그(flagged) 되야 하고, 요건되로 회로가 보호되면서, 설계가 매우 조심스럽게 진행되야 한다.

2. 전원이 공급되지 않은 CMOS I/O로의 신호 입력

8.1절과 유사하게, 서로 같은 보드내에 있든, 떨어져 있든 서로 인터페이싱하고 있는 IC들 사이에 전원 공급 시퀀스는 반드시 주의깊게 살펴봐야 한다. 대부분 IC들, 특히 CMOS의 경우, 전원이 차단되었을 때에는 시스템에 대하셔 낮은 임피던스(impedance)를 갖고 있다. 이러한 IC들의 대부분, 입력 혹은 출력(많은 FPGA 출력은 일반적인 목적의 I/O모듈 안에서 active한 입력을 갖고 있다.)상에서 신호가 이용(application)되기 전에 전원 공급이 bring up이 되어야 한다고 요구한다. 몇몇 프로그램 가능한 IC들은 이러한 것들이 분석되지 않았다 특별한 설계 구성을 반드시 알고 있어야 한다. 예를 들어서, 몇몇 I/O 모듈들은 cold sparing를 제공하고 있다 즉 그것들은 파워가 차단되었을 때, 시스템에 대해서 높은 임피던스(impedance)를 갖고 있다. 다르게 구성된 그와 같은 I/O는 PCI 호환성

을 위해서 전원이 차단되면서, 스위치된 클램프 다이오드(clamp diode)를 갖고 있을 수도 있다. 적절한 최악의 경우를 분석하기 위해서 종종 자세한 설계가 필요하다.

3. 시작 전류 과도기(transient)

대부분 현대의 디바이스에 있어서, 시작 전류 과도기(Startup current transients)는 일반적인 현상이다. 전류의 양은 전원 사이클, 온도, 전원의 램프 비율(ramp of rate), 방사노출이력(radiation exposure history), 전원 공급 시퀀싱 등등 사이에서 시간의 함수가 될 수 있다. 이러한 전류는, 몇몇 앰프에서처럼, 훨씬 더 크다. 전원공급 시스템이 마진을 갖고 정상 상태(steady state) 레벨과 같은 경우에 전류를 제한하지 않은 것은 중요한 일이다. 왜냐하면 시작 시퀀스동안 불충분한 전류로 인해 적절한 초기화 실패, 전원 디바이스 셧다운(shutdown), 혹은 무한 루프의 반복, 시스템 락업(lockup), 치명적인 embrace가 발생할 수 있다. 유사하게 몇몇 부품들은 최소, 최대 전원 공급 rise time에 대한 엄격한 제약을 갖고 있다 이러한 레벨을 만족시키지 못하면, 회로상 문제가 발생할 수 있다.

4. 바이패싱(Bypassing)과 분배

현대의 논리 디바이스들은 수십만 개의 게이트로 이루어져 있어서 상당히 크다. 동기적 설계 기술, 높은 동작 주파수, 많은 I/O 개수들로 인해 전원 분배와 조절 시스템(conditioning system)이 어려워졌다. 대부분 공급자는 어플리케이션 노트(application notes)에 자세한 사항을 언급한다. 어플리케이션 노트에 다른 값의 다양한 커패시터를 너무 많이 사용하는 것 같지만, 이는 설계를 더 어렵게 만들려고 한 것이 아니다. 만약 최악의 상황에 대한 시스템의 분석과 테스트에 대한 세심한 주의를 기울이지 않았다면, 이 규칙을 따라야 한다. 높은 충실도를 만족하는 확실한 전원을 보장하기 위하여, 재구성 가능한(reconfigurable) 논리가 활용될 수 있다. 그런 다음 비행 어플리케이션(application)에 대체된다. JTAG 인터페이스가 사용될 수도 있다. 그리고 JTAG 테스트 패턴들이 SSOs와 같은 설계 제한을 위반하지 않도록 세심한 주의를 기울어야 한다. 공통 에러들이 종종 발생할 수 있는데, 이는 제조사의 권고사항을따르지 않았기 때문이다. 예를 들면, QFP(Quad flat pack)의 모든 면에 바이패스(bypass) 커패시터를 갖고 있지 않은 경우이다.

제 8 절 비휘발성 메모리들

1. 전원공급/차단Power-Up/Down 과도기(Transition) 동안의 보호

이는 비휘발성의 지울 수 있는 메모리들의 공통된 문제이다. Power-up, power-down과 전원 전압 저하 (Brown out) 과도기(Transient)동안 적절하고도 안전한 동작을 위한 모든 신호들에 대해서 철저한 분석과 테스트가 반드시 이뤄져야 한다. 현장에서 사용되는 실제 전원

과는 상당히 다른 특성을 지닌 실험실용 전원을 사용하지 말고, 반드시 현장에서 사용되는 실제 전원과 그 전원의 제한된 특성(Bounded characteristic)들을 반드시 사용해야 한다. 몇몇 다바이스들은 의도되지 않은 쓰기를 방지하기 위해서 리셋 핀을 갖고 있다. 모든 조건에 대한 회로의 설계, 분석, 테스트/평가는 비휘발성 메모리의 내용의 보전을 위해서 아주 중요하다. 회로 동작에 대해서 고려해라. 만약 계획되었던 혹은 의도하지 않았던 쓰기주기 동안에 전원이 차단되었다면, 설계는 비휘발성 메모리의 내용을 확실하게 보호하기 위하여 적절한 쓰기 주기의 완성이 반드시 이뤄져야 한다. 흔히 쓰기주기는 버스의 완료 동작시간뿐 아니라, 10ms 정도 차수의 시간이 걸릴 수도 있는 해당영역에 접근하여 쓰는 시간까지도 포함한다. 관련된 다른 고려사항은 시스템 리셋 신호의 예기치 않은 어플리케이션(Application)이다. 쓰기주기들이 중요 신호들이 안전하게 되면서, 완전하게 마무리되고, 적절히 셧다운(Shutdown)이 확실하게 되기 위해서 셧다운(Shutdown)상태로 들어가야 한다.

2. 쓰기 주기 동안의 피해 분석

메모리가 비행 중에 쓰기동작이 이루어진다면, 비휘발성 메모리의 기술이 반드시 고려되어야 한다. EEPROM과 같은 몇몇 디바이스들은 셀에 쓰기 위해서 높은 전압을 사용한다. 만약 높은 전압의 무거운 이온이 강타한다면, 심각한 오류 상태가 될 수도 있다. 따라서, 쓰기동작은 조심하게 다뤄져야 하고, 저장을 위해 사용되는 기술 또한 현명하게 선택되어야 한다.

3. 주기 카운트(Cycle Count) (즉, EEPROM의 쓰기 주기(Write cycle)의 개수, FRAM, 기타등의 모든 쓰기 주기의 개수)

대부분 지울 수 있는 비휘발성 메모리들은 제한된 사이클 개수를 갖고 있다. 주기의 개수가 104~105 혹은 더 많기 때문에, 주기 개수에 대한 확실하고 빠른 규칙은 없다. 반드시 각각의 디바이스들이 시스템 생명시간과 방사능(Radiation)을 고려하여 개별적으로(Case-by-case) 다뤄져야 한다. 일례로, 여기에 주목할만한, 좀 미묘한 명세서들이 있다. 예를 들어, 흔히 사용되는 128k x 8 히다치(Hitachi) 다이(die)는 페이지 모드(Page mode) 로는 104 주기를 갖고, 바이트 모드(Byte mode)에서는 103 주기를 갖는 생명시간(Liftetime) 쓰기 명세서(Specification)을 갖고 있다. 이 다바이스에 사용되는 쓰기 메커니즘(Mechanism)은 가장 작은 유닛이 쓸 수 있는 8 바이트 서브페이지(Subpage)를 이용한다. 따라서, 한번에 1 바이트(Byte)를같은 메모리 공간에 쓰는 것은 페이지 쓰기(Page write)보다 더 스트레스를 받는다. 왜냐하면 전체 서브 페이지(Subpage)들은 반드시 첫번째로 패치(Fetch)되고 나서, 다시 씌여지기 때문이다. 또 다른 미묘한 특성은 몇몇 FERAM(ferroelectric RAMs)의 동작이다. 이 다바이스의 경우, 읽기 주기들은 파과 판독 모드(Destructive readout mode)상에서 동작한다. 그리고 내부 쓰기 주기는 매번 읽고 나서 실행된다. 따라

서 쓰기 주기뿐 아니라 읽기 주기의 숫자도 반드시 관리되어야 한다. 왜냐하면 각각의 읽기 접근은 그에 따르는 쓰기 주기를 동반하기 때문이다.

4. 과도기(Transients)와 잡음

비휘발성 메모리와 인터페이싱되는 신호들은 깨끗해야 하고, 시스템 잡음이 최소상태를 유지하고 항상 모든 스펙(specifications)을 만족해야 하는 것은 중요하다. 이 경우에, 신호는 논리 신호 뿐 아니라, 전원과 접지 연결도 포함된다 강한(robust) 바이패싱(bypassing)이 사용되어야 한다. 예를 들어 EEPROM의 잡음 글리치(glitch)들로 인해서 잘못된 쓰기 신호가 발생될 수 있고 그로 인해, 디바이스의 내용이 의도하지 않게 변경될 수 있다. 쓰기 신호가 단언되지(not asserted) 않은 채, 비휘발성 메모리로의 정상에서 벗어난 타이밍으로 인해 메모리 내용에 변질될 수도 있다.

5. 신뢰도, 리프레싱(Refreshing)과 리로딩(Reloading)

지울 수 있는 비휘발성 메모리의 신뢰도는 어플리케이션(application)에 따라서 요구되는 신뢰도가 다르다. 만약 디바이스가 큰 메모리 배열의 하나의 부품으로 동작한다면, 몇몇 비트 오류들과 심지어 페이지(page) 오류들 조차도 에러 수정 기술이나 혹은 에러 감지에 의해 완화될 수 있다. 그리고 mapping the failed segment out of service.중앙 처리 장치(CPU)나 FPGA의 메모리를 위한 부트(boot) ROM과 같은 어플리케이션(application)은 완벽한 시스템 성능을 요구한다. 단일 비트 오류의 경우, 비록 그것이 직렬(serial) PROMs에 불편할지 몰라도, 해밍(hamming) 코드면 충분하다. 비휘발성 메모리 디바이스의 몇몇 고장 모드로 인해 비트 오실레이팅(oscillationg) 혹은 유효한 논리 레벨을 제공하지 않는다는 점을 주목해라 이 경우에, EDAC(Error Detection and Correction Code) 다비이스는 그 오류를 EDAC 디바이스의 로직 설계에 따라서 그리고 정적 위험 요소가 있는지 없는지에 따라서, 수정될 수도 있고, 없을 수도 있다. 어떠한 사건에서, 특정한 시스템의 아키텍쳐(architecture)와 결합해서 사용되는 디바이스들은 어떤 확실한 고장이든지, 반드시 락업(lockup) 상태가 되어서는 안된다. 확실한(credible) 고장은 어떠한 단일 비트 에러나 비영구적인 메모리 내용의 의도하지 않은 변질을 포함한다.

변경 가능한 여분(spares)을 갖고 있는 TMR(Triple Modular Redundancy)와 같이 잉여(Redundancy)의 다른 형태들이 필요할 수도 있다. 몇몇 옵션들은 대체 디바이스들 간에 변경할 수 있는 능력, PROM과 같은 영구 메모리의 사용, 지울 수 있는 비휘발성 메모리 기능을 대체할 저장 버퍼의 사용, 위험을 관리하기 위한 선택적 오버헤드(overhead)의 이용 등을 포함한다. 예를 들어, 만약 구성 FPGA의 구성 메모리 디바이스에 결함(fault)이 발생하면, 저장 버퍼와 CPU는 다른 로딩 모드를 이용하여 FPGA를 구성할 수도 있다. 물

론 FPGA가 컴퓨터를 동작시킬 필요가 없다는 가정하에서이다. 일반적으로 중요 어플리케이션에서 PROM과 같은 영구 메모리들이 우주선이나 다른 시스템을 영구적으로 잃어버리지 않기 위해서 사용된다. 이는 부트(boot) 형태와 프로세서를 위한 safe-hold code나 FPGA를 위한 기본적인 동작 구성 등이 될 수 있다.

또 다른 고려 사항은 미션의 길이와 보장된 디바이스의 저정 시간의 비교이다. 이와 관련하여 빠르고 확고한 규칙은 없기 때문에, 각 디바이스들은 반드시 개별적으로(case-by-case) 분석되야 한다. 메모리 내용을 유지하는데보통 10년정도이다. 그러나 이렇게 몇십년의 시스템 수명을 갖는 것은 흔한 일이 아니다. 리프레싱(refreshing)이 위험할 수도 있다. 그리고 특히 방사능 환경에서 저장장치의 무결성(integrity)을 확실하게 보증하기 위해서, 제조자의 도움을 받아 리프레싱(refreshing)의 유용성이 증명되야 한다. 확실히, 디바이스가 리프레시(refresh)될 때, 위에서 언급한 무거운 이온들에 의한 우주환경에서는 쉽게 피해를 입을 수 있다. 컴퓨터의 충돌, 전압저하(brown out)나 버스 결함으로 의도되지 않은 전원 제거나 안전 모드로 들어가는 우주선과 같이 내용에 피해를 주는 에러들이 발생할 수 있다. 또한 각각의 쓰기 주기는 컴포넌트의 동작 수명을 단축시킨다. (each write cycles takes away from the operational lifetime of the component.)

6. 권고사항과 팁

(1) 많은 설계자들은 POR (Power On Reset) 신호를 만들기 위해서 단순한 RC 타이밍 회로를 이용한다. "Power On Reset? signal. Looking closely at the acronym, is has the word "on?in it and the "O?does not stand for "Off.?nbsp 보통 그러한 회로들의 사용으로 power-up에 대해서 메모리가 보호될 것이다. 그러나 보호 회로의 단정(assertion)은 절전되는 동안이나 혹은 전원이 제거되었을 때, 늦춰질 것이다.

(2) POR 회로들은 종종 전원공급 모듈에서 최고로 만들어진다.

(3) 전원 과도기(transient) 조건하에서 중요 메모리들이 적절히 동작하도록 확실히 해라. 흔히 그것들은 FPGA에 의해서 잘못 구현된다. FPGA는 전원이 중단되었을 때와 power-on, power-off와 주기(period) 동안에는 통제상태(under control)가 되는 것을 보장하지 않는다. FPGA와 구성 메모리 다바이스의 내부 power on reset 회로들은 초기 시퀀스에 따라 활동(active) 상태가 될 수도 있다. 전하 펌프는 고전압 격리 FET등을 켜기 위한 충분한 전하와 전압을 제공해야만 한다.

(4) 지울 수 있는 메모리 디바이스의 보호는 아날로그 기능이다. 그리고 디지털 컴포넌트들은 사용되기 전에, 아주 세심한 주의를 기울어야 한다. 타이밍에 따라, 많은 메모리 디바이스들은 보호를 위해 비-표준(non-standard) 전압, 전류 레벨을 요구한다.

(5) 소프트웨어의 결합 가능성이 100%임을 고려해라.

(6) 디바이스 보호: 메모리에 의도하지 않은 쓰기 기능을 방지하기 위하여, 대부분 지

우기 가능한 디바이스에는 쓰기 보호 소프트웨어가 구현되어 있다. JEDEC은 이러한 종류의 보호에 대한 표준을 발행했다. 만약 반드시 필요하지 않는다면, 보드상에서 메모리를 풀기 위하여(unlock) the "keys?를 유지하지 마라.

(7) 부시스템(subsystem) 보호: 소프트웨어 결함(fault)에 대해 시스템을 보호하기 위하여, 하드웨어에 시스템 차원의 쓰기 보호 제한이 구현되어야 한다. 몇몇 시스템은 위험한 소프트웨어에 이러한 기능을 구현하였다 위에서 언급한 5번 항목을 봐라. 의도하지 않은 쓰기 방지를 위하여, 추가적인 방어물로서 외부 하드웨어 이산 명령(external hardware discrete command)을 사용해라.

(8) 락업(lockup) 상태를 위한 다바이스들을 테스트하고 분석해라. 이러한 것들은 비정상적인 로드에서 명령 레지스터, 질이 좋지 않은(poor) 신호의 무결성(integrity), 질이 좋지 않은(poor) 전력, 혹은 SEU까지 많은 종류의 메모리 타입에서 일어날 수 있다.

(9) 시스템의 락업(lockup) 상태를 유발시키는 모든 확실한(credible) 결함(fault) 하에서는, 소프트웨어 기능으로 구현된 부팅을 위한 메모리 이미지들 사이의 중요 스위칭은 그 기능을 보장하지 않을 수도 있다. 결함으로부터 시스템 락업(lockup)을 방지하기 위해서 이산(discrete) 하드웨어 신호를 사용해라.

(10) EEPROM 혹은 플래쉬(flash) 다바이스의 결함 가능성이 100%가 될 수 있음을 고려해라. 이러한 접근방법을 정당화할 수 있는 많은 결함들이 산업계에 있다.

(11) 부트(boot)와 안전-유지 코드(Safe-Hold Code): 고신뢰도, 내방사선(radiation-hardened), 고정 메모리들은 보통 부트(boot)와 안전-유지(safe-hold) 기능을 위해 사용되어야 한다. 계기(instrument)와 같은 어플리케이션(application)에서는, 적절하게 구현된 DMA 기능이 부트(boot) 코드와 함께 메모리를 로드(load)할 수 있다. 이러한 경우에, 하드웨어 논리에 의해서, 계기(instrument)는 안전하게 된다. DMA 기능들은 어떠한 운영(operational) 소프트웨어를 요구해서는 안된다. 프로세서를 리셋으로 클램프(clamp)시키기 위한 하드웨어 이산(discrete) 명령(command)도 추천한다.

(12) 지울 수 있는 메모리에 저장되어 있는 부트(boot)코드와 같은 중요 코드에서 디바이스의 결함(fault)을 완화시키기 위해, 리프레싱(refreshing)을 해서는 안된다. 대신에, 신뢰성 있는 고정 메모리 기술을 사용해라.

(13) 모든 보호 신호들의 마진(margin)들을 증명해라 AC 전압 마진들(예를 들어 크로스톡(crosstalk); 타이밍(power up과power down, 글리치(glitch)들에 대한 보호 신호들). 전압 버스의 power-down 비율은 보통 무시되거나, 이상화(idealized)된다.

(14) 제 3자 다바이스 패킹(packing) 하우스(house): 그들이 원래 제조업자의 테스트 과정들, 여과 기준(screening criteria), 기술에 대해서 완벽히 이해하고 있는지 검증해라. 제 3자 집단에서 수행한 실패율과 원래 다이(die) 제조업자가 작성된 것과 비교해라. 우주 임무를 위한 적절하고 완벽한 테스트가 수행되었는지 확실히 해라.

(15) 만약 기초 기술(fundamental technology)을 신뢰할 수 없는 상태에서, 같은 기술에서

같은 코드를 여러 번 복사하는 것은 위험한 일이다. 예를 들어, 현재 EEPROM의 제조업 실패가 빈번하게 발생하고 있는 상황에서, 같은 디바이스 타입을 많이 복제하는 것은, 심지어 하드웨어 선택까지도, 러시아 룰렛(Russian Roultte)과 다를 바 없다. 하나의 디바이스의 다른 블록에 잉여 복제 코드를 저장하는 것은 공통원인고장(common mode failure)과 관련된 주제일 수 있다.

(16) 레코더(recorder)와 같이 많은 사례를 통해서, 소프트웨어의 디바이스 결함, 블록, 비트가 다뤄져야 한다. 예를 들어, 중요 부트(boot) 코드의 경우, 결함(fault)을 리렛이 되기 전에 반드시 행해져야 하는 소프트웨어 보수관리(maintenance) 이슈(issue)로 다루는 것이 소프트 웨어로 분류된 하나의 기능이 되어서는 안된다.

제 9 절 타이밍 분석 및 여유도

1. 개요

디지털 시스템의 타이밍 분석은 아주 단순하게 요약될 수 있다 데이터 시트(sheet)에 있는 모든 파라미터가 설계의 모든 요소를 만족하는지 확실히 해라. 실제의 경우, 이는 중대한 노력이 될 수 있고, 계산이 정확하게 수행하는지 확실히 하기 위해서, 세심한 주의를 반드시 기울어야 한다. 적절하게 설계되고 분석된 회로는 명시된 전체 동작 환경하에서 모든 컴포넌트의 조합에 대해 잘 동작할 것이다.

단순히 CAE 소프트웨어 툴을 사용해서 버튼을 누르는 것은 매력적인 일이다. 하지만 일반적인 경우에 이는 잘 되지 않는다. 전형적으로, 만약 설게 스타일과 회로들이 CAE 툴 벤더(vendor)가 원하는 모델과 잘 맞는다면, 많은 분석들이 빠르고 정확하게 행해질 수 있다. 그러나 이는 모든 분석에 적용되는 것은 아니다. 그리고 모든 정당한(legitimate) 회로들이 툴 벤더(vendor)의 모델과 잘 맞지 않는다. 왜냐하면 전력, 면적, 기능성에 있어서 엄격한 요구사항이 있기 때문에 CAE 툴이 이를 해결할 수 없기 때문이다.

2. 타이밍 분석의 기초

모든 타이밍 분석의 기본은 클럭과 플립플롭이다. 이전 장에서 상세히 다뤘던 클럭의 경우, 반드시 파라메트릭하게(parametrically), 글리치(glitch)가 없도록 잘 이해되어야 한다. 따라서 로직에 위해서 생성된 어떠한 클럭이든지 깨끗해야 하고, 주기(period)와 순환주기(duty cycle)내에 있어야 하고, 관심있는 다른 클럭과의 위상관계도 잘 알려져 있어야 한다. 예를 들어, 클럭을 만들기 위해 보팅(voting) 회로를 사용함에 있어서 보터(voter)에 위험요소가 있다면, 문제가 발생할 수도 있다. 논리 합성기들은 클럭 생성 회로에서 논리 해저드(hazard)들을 이미 만들어냈거나, 만들 수 있는 능력이 있다.

플립-플롭(혹은 래치)은 이 단원의 기본이다. 아주 단순하게, 플립-플롭의 모든 파라미터들은이 항상 조건을 만족하는지를 반드시 증명해야 한다. 이와 관련하여 한가지 예외는 동기화기(synchronizer)들이 비동기 신호를 동기화하기 위해서 사용했을 때이다. 이것에 대한 지침은 다른 단원의 주제이다.

High, low 모든 상태에 대해서 클럭은 최소 펄스 너비 요구사항을 반드시 만족해야 한다. PLL과 같은 어떤 회로들에게는 최대 지터(jitter)와 같은 다른 요건들이 있을 수 있다. 클럭 속도가 증가함에 따라서 지터(jitter)는 점점 중요한 파라미터가 된다. 다바이스의 명세(specification)와 근접한 클럭에 대해서, high와 low의 시간 측정 방법과 클럭의 특성에 대해서 주목해라. 왜냐하면threshold 전압이 클럭 명세서(specification)와 입력 디바이스의 명세(specification)사이에서 다를 수 있기 때문이다. 또한 로딩(loading)과 환경적 요소에 의해서 영향을 받는 클럭 신호의 변화 시간(transition time)은 이용 가능한 펄스 너비를 열화(degradation)시킬 수 있다. 적절하게 펄스 너비를 유지시키지 못하면, 플립-플롭이 "메타스테이블(metastable)" 상태가 된다.

비동기 프리셋(preset)과 클리어(clear)에 대하여, 반드시 만족해야 하는 두 가지 기본 파라미터들이 있다. 분명히 반드시 보장되어야 하는 펄스 너비 요건이 있다. 그러나 클럭에 비동기적으로 디바이스로부터 프리셋(preset)혹은 클리어(clear)를 제거하는 것은 시퀀셜 논리에서는메타스테이블(metastable) 상태를 유발시킬 수 있다. 이 파라미터는 흔히 제거 시간(removal time)이라고 불리고, tREM으로표시된다. 불행하게도, 많은 데이터 시트들은 제거 시간에 대해서 명시하지 않는다. 그렇다고 해서 요구사항이 아니라고 의미하는 것은 아니다.

데이터(or J, K, T, EN, synchronous clear, 등등)입력에 대해서, 클럭의 가장 빠른/가장 느린 도착 시간에 대해서 모든 setup과 hold time들이 만족함을 보여라. Setup time들은 일반적으로 설계자에 의해서 계산된다. 그리고 테스트에 의해서 적절한 마진이 증명될 수도 있다. 그러나 Hold time들은 흔히 설계자에 의해서 계산되지 않는다. 그리고 CAE 툴들은 종종 hold time들을 부정확하게 계산하거나 정확하지 않은 데이터베이스를 이용하거나 이 둘의 어떤 조합들을 이용한다. 디지털 논리의 오작동(malfunction)에 대한 가장 큰 원인 중의 하나는 hold time 위반이다. FPGA의 경우, 글로벌 클럭을 사용할 때 제조업자가 hold time이 만족되도록 보장하는지를 알기 위해서는 다비이스의 명세서를 주의깊게 체크해라. 항상 이런 경우는 아니다.

하나의 클럭 에지에서 다른 에지로 데이터가 "패싱(passing)"할 때, 최악의 상황에서의 순환 주기(duty cycle)이 계산에 이용되도록 확실하게 해라. 흔히 발생하는 에러의 원인은 모

든 클럭이 50%의 순환주기(duty cycle)을 갖는다고 가정했기 때문이다.

하나의 클럭 도메인(domain)에서 다른 도메인(domain)으로 데이터가 패싱(passing) 할 때, setup과 hold time들이 만족되도록 보증할 수 있는 잘 알려진 위상 관계식이 있도록, 혹은 회로가 적절하게 동기화되도록 확실히 해라.

만약, worst-case 분석을 만족시키기는 부품들을 선별하기 위하여 측정된 값을 사용한다면, 부품의 시험은 best-case 및 worst-case 접근 시간에 대하여 모두 시험하여야 한다. 예를 들어,

(1) 배열의 다른 부분에 어드레싱(addressing)할 때 ROM/PROM의 접근 시간을 위해 4:1의 차이를 둔다.

(2) 높은 공급 전압에서 몇몇 메모리들 속도가 느려질 수도 있다. 왜냐하면 더 높은 공급 전압으로부터 증가되는 속도를 상쇄시키는 것 이상으로 센스 증폭기에 대한 threshold 값이 올라가기 때문이다.

(3) 몇몇 메모리 디바이스들에 대한 접근 시간은 데이터가 읽혀지는 시퀀스의 함수가 될 수 있다. 예를 들어서, 위치에 따라 좌우되는 전의 읽기는 특별한 상태에서 라인들과 센스 증폭기를 떠날 수도 있다. 따라서 0을 읽은 후 0을 읽는 것은 1을 읽은 후 0을 읽는 것과는 다른 결과가 나타날 수 있다.

3. 환경적 영향

강건한(robust)한 회로에 대해서, 설계는 다양한 환경적 영향에 대해서 반드시 내성이 있어야만 한다. 이는 다음의 것들을 포함한다.

- 온도
- 전압
- 수명
- 방사선(Radiation)
- 프로세스, 속도 등급(speed grade), 프로그래밍

일반적으로 분석자들은 가장 광범위한 각 환경적 요소의 가능성 있는 코너(corner)들에 기반을 분 한계 값 분석(Extreme Value Analysis)을 할 것이다. 이러한 EVA를 통해서 넓은 범위의 마진, 불확실성(unforeseen)의 허용오차, 비공칭 조건(off-nominal condition)을 갖는 시스템이 나올 것이다. 그러나 이러한 과정은 많은 경우에 있어서 불필요하게 성능을 제한하고, 자원 소비를 증가시키거나 더 복잡한 아키텍쳐(architecture)와 분석을 요구할 것이다. 예들 들어, 같은 다이(die)에 단지 몇 마이크론 떨어져 위치해 있는 두 개의 플립-플롭에 대해서, 그 중 하나는 -55 ℃에 있지 않을 것이다. 반면에 다른 하나는 +125 ℃

에 있을 것이다. 이러한 경우에 "연필을 날카롭게 하는 것이" 현명할 것이다. 100%의 추적을 가정하는 것은 이 파라미터에게는 유효하지 않다 다른 파라미터들에 대해서는 추적 불가능이 가정될 수도 있다. 흔히 설계자/분석가들은 사용 가능한 데이터 혹은 모델에 의해서 제한되어 있고, 얼마나 많은 추적이 일어나는지에 대해서 결정할 수 없을 것이다. 이러한 경우에는 보수적인 접근방법으로서, 추적할 수 있는 최소량이 가정되어야만 한다.

사용된 온도와 전압은 각각의 특정 임무와 전자공학의 위치의 함수가 될 것이다. 프로젝트의 신뢰도 계획상에서 명시된 플러스 마진이 최악의 경우의 값으로 반드시 이용되어야 한다. 더 낙관적으로 기대한 값이 되어서는 안된다. 어려 임무가 있어왔고, 그 임무에서 실제 값들은 예상된 값들의 범위 밖에 있었다.

컴포넌트들은 노화되고, 특성이 변한다. 그러나 하나의 예로 모든 전파 지연들이 트랙하고(will track) 상대적인 지연은 변화하지 않을 것이라는 가정은 할 수 없다. 예를 들어 어떤 FPGA의 경우에, 생명 테스트 데이터에 대한 여러 연구들은 제조된 하나의 Lot에서 표본으로 추출된 다바이스들에 대해서 지연은 track되지 않을 뿐 아니라, 심지어 같은 부호를 갖지 않을 수도 있다고 보고되었다. 따라서 테스트에 의해서 hold time 마진을 증명할 수 없다. 일반적으로 대부분의 프로그램들은 미션 수명에 걸쳐 전파 지연 변화에 대한 ?0% 를명시할 것이다.

방사능(radiation)에 대한 접근방법도위에서 언급한 수명과 유사하다. 완벽한 추적(tracking)을 가정할 수 없다. 다시 한번, 미션이 수행되는 동안 전파지연 변화에 대한 ?0% 가 일반적으로 사용된다.

전형적인 타이밍 분석 프로그램으로 인해 typically best, typical and worst처럼 프로세스에 대한 셋팅을 선택하는 것이 가능해졌다. 이러한 셋팅을 효과적으로 이용하기 위해서는 이러한 것들이 회로 속도(circuit speed)의 예언자가 아니라 회로 속도에 대한 하나의 범위(bound)라는 점을 주목하자. 많은 기술자들과 분석자들은 이 프로그램이 속도를 예측하거나 혹은 두 개의 회로 중 하나가 더 느리다는 것을 증명해 줄 것이라고 가정한다. 그러나 현실을 그렇지 않다. 예를 들어, 두 트랜지스터가 매우 비슷하게 프로세싱 되었을지라도, 동일하게 프로세싱되지는 않을 것이다. 로트(lot) 변화, 같은 로트(lot)안에서도 웨이퍼간의 변화, 같은 웨어퍼 안에서도 다이(die)간의 변화, 같은 다이 안에서도 트랜지스터간의 변화들이 있다. 따라서 이러한 값들은 반드시 실제 값이 아니라 범위(bound)로서 다뤄져야 한다. 트랙킹(tracking)에 대한 어떤 단계만 있을 것이다. 분석에 사용될 수 있는 양의 크기는 CAE 툴에 있는 이용 가능한 데이터와 알고리즘에 달려 있다.

스피드 등급 셋팅은 종종 타미밍 분석을 잘못되게 만들 수 있다. 부품에 찍힌 최악 혹은

가장 느린 경우의 속도 등급은 사용하기에 정확한 셋팅이다. 가장 최고의 혹은 가장 빠른 경우의 등급을 사용하면, 당신에게 잘못된 해답을 줄 것이다. 예를 들어, 몇몇 FPGA들은 특별한 테스트 회로의 속도를 측정하고 그 속도가 threshold 값보다 작다는 것이 확실시 되면 폐기된다(is binned). 이는 일방적인 관계이다 그리고 더 빠른 속도등급을 통과했을지도 모르는 부품들이 폐기되고, 낮은 속도등급으로 부품에 찍힐 것이다 만약 그렇지 않으면 그 부품들은 선적될 리 없다. 왜냐하면 부품 번호가 잘못되었기 때문이다. 그래서 만약 상세한 지식이 없다면 가장 좋은 혹은 가장 빠른 경우에 대해서 툴이 제공하는 가장 빠른 속도 등급을 사용해라. 이것은 명세서나 벤더(vendor)가 제공하는 read나 레코드 데이터에 나와 있다.

Antifuse에 기반을 둔 FPGA에 대하여, 분석상에서 가정된 트랙킹(tracking)의 양은 다른 디바이스 타입에서 있는 것보다 더 적을 것이다. 다이(die)위에 있는 트랜지스터들은 어느 정도 트랙(track)하겠지만, 두 트랜지스터들이 서로 같이 조립되었기 때문에, 프로그램된 antifuse 저항의 분포는 확률변수(random variable)와 비슷하다.

종합적으로 판단하면, 만약 신호 A는 항상 T나노초 만큼 B신호보다 먼저 도착하기를 원한다면, 최악의 상황에 대한 모든 값을 셋팅하여 동적 시뮬레이션을 하는 것은 잘못된 답을 줄 것이다. 왜냐하면, 모든 경로가 최악이라는 것을 보장하지 않기 때문이다. 실제로도, 보장하지 않을 것이다. 이것은 최소-최대 혹은 극한 값 분석이 정확한 타이밍 분석에 필요한 이유이다.

제 10 절 기타 설계 지침과 기준

1. 노이즈 내성과 노이즈 없는 설계 (Quiet Designs)

잡은 내성이 충분하고 강한(robust) 설계를 위해서 다음과 같은 단계를 밟아라.

(1) 동시적(simultaneous) 스위칭 출력과 신호 종단에 관하여 중요 단계가 리스트되어 있다.

(2) 카드간의 연결을 위해서는 특히, 차동 신호(differential signal)를 선택해라. 새로워진 논리 디바이스들은 직접적으로 차동의 대한 표준(differential standards)을 지원한다. 추가적으로, LVDS(Low Voltage Differential Signaling)와 같은 표준을 지원하는 고속, 저전력 차동 디바이스들이 현재 자격이 갖춰져 있다.

(3) SERDES(SErializer and DEserializer) 컴포넌트들/코어들은 라인의 개수나 잡음을 줄일 수 있다. 따라서 시스템의 잡음 내성을 증가시킨다.

(4) 잡음을 제거하려면, 히스테리시스(Hyseresis) 입력을 사용해라.

(5) 내부 메모리 드라이빙 케이블을 갖거나, 혹은 많은 커패시티브한(capacitive)한 로드를 갖는 논리 디바이스나 플립-플롭을 피해라

(6) TTL이 호환가능한 입력은 흔히 TTL이 호환되지 않는(특히 VIH) threshold나 명세서를 갖고 있다.

(7) 출력(특히 몇몇 CMOS군(family)의출력인 경우)은 TTL 로드들이 충분한 잡음내성을 갖는 유효한 값의 논리 1을 드라이브 못할 수도 있다. 최악의 경우의 전류와 전압과 최악의 경우의 threshold 값을 비교해서 계산해라.

(8) TTL 인터페이스에 대한 DC 마진은 적어도 500mV를 추천하고, 400mV보다는 커야 한다.

(9) CMOS 논리 threshold를 드라이빙하는 TTL 출력에 대해, 풀업(pull-up) 저항은 충분한 DC 마진을 제공해 줄 수 있다. 이후, 이는 전압이 상승하는데 충분한 시간을 제공한다. 그러나, 클럭 입력으로 사용되었을 때에는 신호 모양이 hump처럼 되어, 다중 트리거(trigger)들이 발생하기 쉽다. 이는 피해야 하고, 부실한 인터페이스이다.

2. 방어적 설계와 비공칭(off-nominal)사건에 대한 설계

신뢰할 수 있고 확실하지만, 예상치 못한 사건에 대해 고려해라. 이러한 상황들은 종종 약간의 생각으로 쉽고 경제적으로 다뤄질 수 있다. 여기에 고려할 만한 몇몇 샘플들이 있다.

(1) 제한과 유효성 체크 실행해라: 시스템은 불합리적인 입력에 대하여 합리적인 방법으로 대응한다. 하나의 소스에서 다른 소스로 넘어가는 데이터에 대해서, 단순한 경계(bound) 체크는 연결되지 않은 소스와 같이(이로 인해 'F'가 데이터 버스상에서 되돌아 올 것이다.) 많은 비공칭(off-nominal) 조건에 대해 적절한 조치를 탐지하거나, 일으킬 수 있다. 플로팅(floating) 포인터(pointer) 숫자들의 경우, 입력이 유효한 포맷으로 되어 있는가? 최소 기준은 어떤 신뢰할 수 있는 입력도 하드웨어를 손상시키지 않아야 하며, 복구를 방지해야 한다. 소프트웨어의 실패 확률이 100%임을 가정해라.

(2) Fail-safe 인터페이스를 제공해라. 각 전선에 대해커넥터로 연결이 끊겼다면 회로의 성능과 안전성을 분석해라. 전력의 경우, 어떤 전선이 단선되었을 때, 여분의 세트가 로드를 운반할 수 있도록 다중선(multiple wires)을 이용해라(그리고 이 잉여분에 대해서 확실히 테스트 해라). 신호에 대해서, 플로팅(floating)된 신호들이 안전상태와 동작 상태로 갈 수 있도록 풀(pull)시킬 수 있는 보드상의 종단(on-board termination)들에 대해서 고려해라. 만약 보드나 부시스템(subsystem)이 테스트 에러로 인해 전원과 연결되지 않았다면. 앞에서 언급한 보드상의 종단(on-board termination)이 이를 보호할 것이다. 신호들을 전원 혹은 접지핀 근처에 놓는 것을

피해라. 왜냐하면 단락(short)이 take out the system(???)(SEASAT를 기억해라)

(3) 락업(lockup) 상태: 당신이 설계하는 모든 디바이스들은 FSM(Finite State Machine)에서는 락업(lockup)상태를 갖고 있어서는 안된다. 흔히 사용되는 상업 디바이스를 선택해라. 그리고 방어적으로 작동시켜라. 예를 들어, SDRAM들은 락업(lockup) 상태가 있는데, 이 락업(lockup) 상태는 전원 주기(cycling)이 클리어되기를 요구할 수도 있고, 피해를 줄 수도 있다 잡음 SEU, 혹은 심지어 유효하지 않은 명령들도 이러한 조건을 일으킬 수 있다. 리프레쉬(refresh) 명령이 종종 그러하다. 많은 마이크로프로세서들, EEPROM과 같은 비휘발성 메모리들은 다양한 락업(lockup)상태를 갖고 있다. 몇몇 디바이스들은 리셋에 의해서 클리어 될 수도 있다 다른 것들은 흔히 전원 주기(cycling)이 클리어 되는 것을 요구한다.

(4) 전원 글리치들(glitches): 여러가지 이유와 전기적인 방전, 스위칭 로드의 on-off, 로드상의 결함, 점화장치의 시동자(firing pyrotechnic initiators), 낙뢰(lightning strike), 릴레이 스위칭(relay switching)때문에 전원 글리치(glitch)들이 일어날 수 있다. 이러한 과도현상(transient)으로 인해 회로의 오작동(maloperation), FMS에서의 multiple bit upsets, 상업적으로(commercially) 파생된 컴포넌트의 락업(lockup)이 발생할 수도 있다. 비휘발성 메모리들과 같이 영구적 상태를 갖는 회로에 대해서는 세심한 주위를 기울일 필요가 있다. 시스템이 메모리 업로드 동안, 모순 상태(inconsistent state)를 초래하는 쓰기 주기가 방해 받을 수 있는가? 시스템의 상태에 오류를 발생시키는 거짓(false) 쓰기가 생성될 수 있는가? One-time 사건들이 거짓으로 시작될 수 있는가? 시스템 상태 정보를 유지하는 래칭(latching) 릴레이(relay)들을 위한 드라이빙 회로들이 거짓 트리거(trigger)들로부터 보호될까? 하나의 전원 공급의 글리치(glitch)가 다중 전원공급을 갖고 있는 컴포넌트나, 다른 전원 공급기로부터는 전원을 공급받지 않는 컴포넌트들의 세트에게 전원 시퀀싱(sequencing) 위반을 초래할 있을 것인가?

(5) 테스트 포인트의 단락(shorting): 테스트 포인트들은 사람에 의해서 다뤄지기 때문에, 테스트 포인트들이 오용(abuse)될 수 있음을 가정해라. 즉 확실하고 예상되는 결함(fault)의 예 중 하나가 바로 오실로스코프 프로브의 그라운드 링으로부터의 그라운드 단락이다. 테스트 인터페이스상의 결함(fault)이 비행 하드웨어를 확실히 손상시키지 않기 위해서는 격리 저항(isolation resistor)들을 사용해라.

3. 다양한 팁들과 고려사항, 기준

(1) ESD 등급들(ratings): 현대의 고속 컴포넌트의 대부분은 낮은 ESD threshold를 갖고 있다. 우주선 인터페이스에 적합하게 설계된 몇몇 컴포넌트들은 300V의 제한을 갖고 있었다. 종종 이러한 값들은 데이터 시트에 언급되지 않는다. 따라서 반드시 qualification 테스트 리포트가 입수되어야만 한다.

(2) 여분들(spares), 핀들, 게이트들: 여분의 공간과 여분의 PCB(Printed Circuit Board)의 패드(pad)들을 남겨둬라. ““oop””를 고정시키거나, 변화된 요건을 만족시키기 위해서 필요할지도 모르는 SSI(Small Scale Integration)/MSI(Medium Scale Integration) 컴포넌트들에게 적합한 바이패스(bypass) 커패시터, 접지, 전원을 위한 footprint를 미리 배선해라. 프로그램 가능한 디바이스의 경우 게이트, 플립-플롭, 핀들이 충분히 이용가능 하게끔 확실히 해라. 이러한 것들이 명백한 소리처럼 들리겠지만, PDRs의 초반기처럼 여유분 없이 설계가 이루어지곤 했었다. leaving zero room for change. 여유 핀들이 있는 프로그램 가능한 다바이스의 경우, 입력 핀이 저항을 통해 접지되어 있는 인버터, AND 게이트들, 플립-폴롭들과 같은 다양하고 단순한 기능들을 프로그램해라. 작은 fixes의 경우, 이것은 보드 상에 이용 가능한 논리를 제공해줄 것이고 잠재적으로 ASIC 혹은 FPGA의 re-spin과 보드의 rework 주기를 제거해 줄 것이다.

(3) 보드내의 홀: PCB 주위에 홀을 전략적으로 위치해둬라. 양면 보드상의 늦은 fixes에 대하여, 이는 수정을 단순화할 수 있다.

(4) 테스트 포인트들: 보드에 테스트 포인트를 만들어 놓는 것은 일반적이고도 유용한 일이지만, 이 또한 전략적으로 오실로스코프의 접지 리드(lead)를 혹으로 고정될 수 있게끔 포인트를 둬라. Surface mount 커패시터들의 움직임으로, 종단 위치를 찾는 것이 힘들어지고 이로 인해 긴 접지 리드(lead), 빈약한 연결, 빠르게 움직이는 신호들에 대한 부정확한 신호모양들을 초래한다.

(5) Lid들의 접지: 전하로충전된 환경(charged environment)에 동작을 위해서, lid들이 접지되어 있음을 증명해라. 실제로, 충전된 환경에서는(charged environment), 움직이는 공기가 데워지고, 다시 식혀서 환경 챔버(environmental chamber)로 들어가는 테스트가 필요할 수도 있다. 몇몇 디바이스들의 lid들은 접지되어 있지 않고, 심지어 그런 상태에서 우주 시장(space market)에 팔려지기도 한다. 몇몇의 경우, 접지된 lid들은 선적하기 전 제조업자에 의해서 플로팅(floating)이 만들어진다. 어떠한 전하의 형성도 불가능하게 만들고, ESD 피해에 대한 예방을 확실히 하기 위해서 드레인(drain) 전선(wire)이 사용되어야만 한다.

제 11 절 설계 및 분석 문서화

1. 라벨 스키매틱

모든 스키매틱 안에 인스턴스(instance), 심볼(symbol), 네트(net)를 라벨링을 하는 것은 좋은 습관이다. 이는 다음과 같이 2가지 이유로 인해 유용하다.

(1) The backend 엔지니어링과 분석 툴들은 라벨을 참조하거나, 당신이 제공하는 라벨 혹은 툴에 의해서 제공되는 디폴트(default) 라벨을 참조하는 결과물을 제공할 것이

다.

(2) 내부 프로빙(probing)이나 문제를 해결할 때, 라벨을 이용해, 엔지니어는 자신이 하고 있는 일에 집중할 수 있다.

아래에 라벨링에 대한 개요가 잘 나타나 있다.

(1) 모든 네트(net)에 이름을 부여해라. 눈에 잘 보이지 않게끔 붙어있는 이름보다 눈에 잘 띄는 이름이 훨씬 낫다. 또한 네트(net) 이름은 신호의 의 의미를 반영해야 한다. 이와 관련하여 다양한 규칙이 있다. Viewlogicf은 데이터베이스에 접두사인 '~' 를 갖고 있다. 그리고 스키매틱의 score에 대하여 사용한다. 대부분 엔지니어들은 접미사인 '_N'를 사용한다. 몇몇 프로그램이나 시스템에서 와일드카드로 사용되는 '*' 는 사용하지 않기를 권고한다. 툴(tool)간에 이식성(portability) 휴대성(protability)을 증진시키기 위해서 신호나 네트(net) 이름은 모두 대문자로 표현되어야 한다.

(2) 모든 인스턴스(instance)에 이름을 부여해라. 이는 I/O 심볼들, 계층적 설계를 위한 모듈들, 플립-플롭들, 게이트들, 기타 등을 포함한다. 이를 통해 이후의 작업이 잘 진행될 것이며, 컴포넌트들을 빨리 찾을 수 있게 도와준다. 또한 이는 drawing 들을 유지하는 데 드는 노력을 최소화하고 한 페이지에 인스턴스 번호를 유지하는 데 필용한 노력을 최소화 한다. 일반적인 참고형식은 XYn이다.

- X는 아래의 것들 중 하나:
  M – 계층의 다른 레밸을 나타내는 모듈이거나 심볼에 첨부된 모듈
  F- 플립-플롭
  G- 게이트
- Y는 순서대로 되어 있는(ordered) 세트인 {A, B, C……} 로 각 문자는 그 모듈안 시트 숫자에 따른다.
- n은 신호 시트에 대한 참조 숫자이다. 각 시트는 1에서 시작하고 다른 모든 시트와는 구별된다. 새로운 컴포넌트를 추가할 때, 시간을 절약할 수 있다.

2. 페이지 간의 신호 연결

Off-page로 가는 신호들은 그 신호들이 도착할 시트나 출발된 시트를 참조하여 표시되어야 한다. 20페이지나 혹은 20 페이지가 넘는 스키메틱을 훑어보고 off-page 신호들이 향하는 각각의 페이를 찾는 것이 어렵고 에러 발생이 쉽다.

3. 보조 합성 파일

보조 합성 파일들은 제1의 설계 문서화로 여겨진다. 특히 합성 프로세스를 조종하거나

안내하는 파일이라든지, 합성기로부터 출력된 결과물에 대해서 상세히 기록된 어떠한 파일이든지 문서화 패키지로 조종되고 포함되어야 한다. 예를 들어, 보조 파일은 "안전" 인코딩(encoding)이 사용되었든 안되었든 간에 FSM의 스타일을 결정할 수도 있다. HDL description은 이러한 경우에 설계를 명세하지 않는다. 대다수 설계자들은 설계를 더 portable하게 만들 수 있는 synthesizer-specific directives가 없게 끔 하기 위해서, HDL source 파일이 아닌 directives를 유지하기를 좋아한다. 설계, 합성, 분석 혹은 수정 과정안에서 에러를 범할 기회를 줄이기 위해서 HDL 코드안에directives를 놓은 것을 더 선호한다. 유사하게, 합성기가 실제로 수행했던 것을 문서화한 합성 결과 파일들 또한 주요한 설계 문서화이다. 플립-플롭과 같이 사소한 것들조차도, ASCll 텍스트 입력에게는 회로 설계를 인식할 수 없기 때문에, HDL source는 불충분하다. 합성기 출력 파일은 플립-플롭이 복제되었는지 아닌지를 결정하기 위해서 필요하다. 이는 대부분의 고신뢰성을 요구하는 어플리케이션에서는 아주 중요하다. 비슷하게, 두 번째 예를 들어, 상태 인코딩들은 출력 문서화에 나열되어 있을 것이다 어떤 경우에는, 합성기가 설계자보다 더 많이 알고 있음을 결정하고, 항상 주어진 지침을 따르지 않는다는 것에 주목해라.

제 12 절 디지털 전자 회로의 검토

1. 개요

FPAG의 검토는 설계/분석 과정, minus the synthesis process와 꽤 유사한다. 왜냐하면 이들 과정이FPGA와 동일한 스킬(skill)과 기술들을 사용하기 때문이다. 이와 같은 사용에 있어서, 합성기는 단순히 컴퓨터 스크린상의 버튼을 누르고, 단순히 인간이 할 일을 소프트웨어가 하는 과정이 아니라 요구하는 기능의 신뢰도를 수행하는 논리 설계를 합성시키는 과정이다. 디지털 전자 회로의 검토는 단순하게 단지 설계가 충실하게 모든 요건과 명세(specification)를 만족시키는 것을 증명하는 것뿐이다. 이것은 설계자/분석자들의 몫이고 검토자의 기능이 여유분이다.

이 섹션은 FPGA-타입의 디지털 설계를 검토 안에 취하는 각 단계들을 설명함으로써 그 과정에 약간의 통찰력을 준다. 그것이 현재 인기 있는 FPGA(RT54SX-S series)와 합성 툴(Synplicity)로 쓰여질지라도 전체적인 방법론과 대부분의 각 단계는 어떠한 FPGA에도 적용될 수 있다. 보통 그 기능에 대한 지식을 알기 전에 설계에 대해 검토에 대한 임무가 주어지는 것으로 가정한다.

2. FPGA에 대한 정확한 명세(specification) 확보

다바이스 명세(specification)들은 언제든지 업데이트 될 수 있다. 그리고 제조자의 부품 타

입들 사이에도 미묘한 차이점이 있을 수 있다. 따라서 검토될 부품의 정확한 명세(specification)를 확실히 갖고 있어라. 최신의 명세(specification)를 위해서 제조사의 웹사이트를 체크해라. 왜냐하면 웹에서의 명세서 (specification)배포, 업데이트들은 통지없이 언제든지 이루어질 수 있기 때문이다. 군 명세서(specification)도 지금 온라인에서 이용가능하다. 물론, 시스템을 검토함에 있어서 모든 디바이들이 이와 같은 세심한 주의가 필요하다.

3. 필요한 검토 파일 수집

대부분의 FPGA 설계들은 VHDL, 간혹 Verilog와 같은 HDL로 이루어진다. 이 문서에만 이렇게 가정한 것이다. 잘 알려진 제조업자가제공해주는 툴 혹은 더 좋게는 FPGA 벤더가 제공하는 표준 툴의 사용을 장려한다. 이는 검토자로 하여금 새로운 툴의 대한 학습(이는 시간과 비용을 증가시킬 뿐 아니라 덜 효율적이다.)과 구매를 강요하지 않음으로써, 검토과정을 가능하게 해준다. 이 논의를 위해서 Synplicity라 불리우는 합성 툴이 사용된다고 가정할 것이다 원리는 다른 제조업자들과 유사하다.
검토에 요구되는 파일들은 다음과 같이 시스템, FPGA, FPGA를 둘러싸고 있는 전자공학을 설명하는 것들을 포함한다.

(1) 시스템 설명(description): PDR/CDR 패키지, 시스템 명세서(specification)등.
(2) 보드 스키메틱 세트
(3) 합성 과정을 지도하는 다른 파일과 그에 따른 FPGA HDL 파일들
(4) 존재하는 테스트 브랜치들(branches)
(5) 합성 결과와 타이밍 분석
(6) .srr 파일: Synplicity 합성 로그 파일
(7) .abd 파일: place and route이후의 데이터베이스 파일—이것은 설계이다.

검토를 시작하기 전에, 시스템 동작과 요건에 친숙해져라. 그리고 설계에 대한 전체적인 느낌을 얻기 위해서 보드 스키메틱을 훑어봐라. 전체 시스템과 그의 중요성을 고려하여 FPGA의 위치에 대한 노트를 만들어라.

(1) 디바이스의 정확한 동작이 안전에 결정적인가?
(2) 다바이스가 점화기의 개시 회로나 자세 제어 로켓, 고전압 전력 공급기등을 제어하는가?
(3) 디바이스가 우주선에 결정적이고, 임무에 결정적인 명령어들을 발행하거나, 래칭(latching) 릴레이들을 세트시키거나, 혹은 형상(configuration) 기능들을 수행하는가?
(4) 얼마나 많은 전원 공급기로부터 전원이 공급되고, 그것들 사이의 순서는 어떻게 되어 있는가? Power-up과 powr-down에 대한 시퀀스에 대해서 모두 고려해라.

(5) 회로 리셋은 어떠한가?

(6) FPGA의 기능이 처음으로 시도할 때, 정확하게 수행하는 것이 시스템상에서 결정적인 기능인가? 혹은 재시도에 대한 기회가 있는가?

(7) FPGA가 비동기 데이터나 명령어들을 수신하는가? 혹은 비동기 사건에 대한 프로세싱을 수행하는가?

(8) 얼마나 많은 클럭 소스들이 있는가? 또한 그 클럭들의 주파수, 순환주기(duty cycle), 위상 관계는 어떠한가? 설계자는 클럭 트리를 제공하거나, 검토과정의 일부분으로 생성되어야만 한다.

(9) PCB artwork은 손쉽게 이용가능해야만 한다. 왜냐하면 신호와 전원의 통합 분석에 이용되기 때문에 필요할 때 쉽게 이용가능해야 한다.

이 시점에서 HDL을 읽은 것은 유익하지 못하다. 왜냐하면 대부분 HDL이 빈약하게 쓰여져 있고 문서화되어 있다 이 시점에서 많은 것을 얻을 수 없다.

4. 검토의 수행

검토가 수행되는 세부적임 몇몇 단계들이 있다. 이상적으로는 모든 설계는 가장 상세한 검토를 받는다. 그리고 이러한 검토안에는 FPGA의 사용과 전체적인 전자설계, 논리설계에 대해서 고려되고 정확성이 증명되었다. 그러나 시간과 예산의 제한 때문에 이러한 것들이 항상 가능한 것은 아니다. 하지만 설계를 검토하는 각 과정들은 궁극적인 검토 레벨에 상관없이 모두 동일이다. 단지 차이가 있다면 얼마나 많은 단계들을 수행할 것인가이다. 앞서 이야기 한 제한사항 때문에 종종 “무작위 검사(spot check)” 혹은 설계의 “스캔(scan)”이 행해질 수 있는 전부이다.

한 가지 중대한 사항은 HDL은 설계가 아니라 원하는 로직에 대한 설계자의 단순한 설명이라는 점이다. HDL 시뮬레이션을 하는 행위나 테스트 브렌치(brench)로는 설계의 정확성을 증명하는데 부족하다.

설계는 back-end place and route 기능의 결과물이고, 하드웨어는 구성(configuration) 이후의 물리적인 칩이다.

실제 설계가 의도된 설계대로 진행되었는지에 대한 충실도(fidelity)는 합성기의 질(이는 잘 알 수가 없다)과 설계자의 능력에 좌우된다.

(1) 합성 가능한 HDL을 작성해라.

(2) 합성과정과 사용된 툴을 이해하라.

(3) 합성과정을 조종해라.

(4) 합성과정이 의도된 것을 생산했는지 증명해라.

(5) Back-end place and route 툴을 올바르게 지도해라. 이러한 툴들은 논리 복제, 결합,

논리 기능의 제거 I/O thresholds, 출력 slew rates와 같은 I/O 모듈의 파라미터 세팅, 클램핑(clamping) 다이오드들의 부재 혹은 존재, cold-spare 기능성 등을 변경시킬 수 있다. 이 툴이 논리 합성기만큼 추상적이고, 복잡하지는 않을지라도, Back-end design process안의 과정들을 이해하지 못한다면 설계 에러를 일으킨다.

FPGA 설계의 제한사항 중 하나는 하나의 클럭 에지를 사용하는 동기 논리를 모두 분석하도록 정적 타이밍 분석이 설계되었다는 점이다. 설계 정확성을 증명하기에는 동적 논리 시뮬레이터로는 부족하다. 비동기 설계 기술들(보통 비동기 설계 기술을 요구하지는 않지만)은 툴을 이용하여 분석하기가 상당히 어렵고, 에러가 발생할 수 있기 때문에 추천하지는 않는다. 따라서 검토 과정의 중요한 부분중의 하나가 에러 발생이 쉽고, FPGA 설계에 이용되서는 안될 설계 기술을 추출해 내는 것이다.

1) 보드 스키메틱들의 검토

FPGA 어플리케이션은 FPGA의 전기적 환경에 대한 이해 없이 적절하게 검토될 수 없다. 아래 리스트들은(이 리스트들이 완벽한 것은 아니다.) 반드시 검토되어야 할 몇몇 사항들을 보여주고 있다.

(1) 가장 중요한 점은 특별한 핀, 예를 들면, TRST*를 적절히 다루는 것이다. 사용하지 않는 클럭 핀들의 요건에 대한 FPGA 스펙(spec)과 디바이스 구성(configuration) 혹은 프로그래밍 핀들과 같은 특별한 핀들을 검토해라. 그리고 그 핀들이 적절히 보드에서 종단되었는지 증명해라.

(2) 흔하게 사용되지 않는 로드를 찾아라(예를 들면, 큰 커패시턴스 혹은 비논리 로드등)

(3) 흔하게 사용되지 않는 소스를 찾아라(예를 들면, 미심쩍은 논리 레벨들, 지나친 과도기(transition) 시간들, 논리군(families)의 혼합, 서로 다른 전원으로부터 공급받는 디바이스들 등등)

(4) FPGA와는 독립적으로 power-up혹은 power-down 될 수 있는 회로와 각 디바이스들의 cold sparing 능력에 주목해라.

(5) 동시에 스위칭하는 출력의 개수와 FPGA의 I/O 링 주위에 그것들의 분포를 결정해라.

(6) PCB 트레이스(trace)들의 길이와 신호들이 어떻게 종단되는지 결정하고, 오버슈트(overshoot)와 언더슈트(undershoot)의 명세(specification)를 만족하는지 확실히 해라. 특히 PCB를 떠나는 모든 신호들을 주의깊게 조사해라.

(7) 바이패스 커패시터들과 전원/접지 면에 대한 제조사의 권고사항들을 잘 따랐는지 확실히 해라. 예전에 검토결과로 어느 곳에는 바이패스 커패시터가 하나도 없고, 다른 곳에 있는 커패시터는 그 성능을 제대로 발휘하는 못하는 경우를 포함하여

충분하지 못한 커패시턴스와 라우팅(routing)을 갖는 보드들이 발견되었다.

2) 합성기 출력 로그 파일 읽기

Synplicity에 의하여 쓰여진 출력 로그 파일인 .srr 파일은(Synplicity는 HDL 파일들을 읽고, 프로세싱한다.) 검토자에게 설계에 대한 꽤 많은 사항들을 알려준다. 경험상, 설계자들은 합성 후 .srr 파일들을 거의 읽어보지 않는다.

(1) .srr 파일의 첫번째 부분은 두개의 패스(pass)들을 나타낸다. 첫 번째에서는 Synplicity가 VHDL 모듈들과 상태 머신들을 찾고, 두 번째에서는 상태 머신을 재방문하여, 리셋 로직이 만들어진다. (reset logic is created for any which the designer gave the "safe?attribute to deal with illegal states.). 첫 번째 패스(pass)에서 발견되는 상태 머신 이름들에 주목해라. 그리고 리셋 논리가 생성되지 않은 모든 상태 머신의 이름들도 주목해라. 모든 상태 머신들은 다룰 수 있는 비정상 상태 (illegal state)들을 갖고 있어야만 한다. 왜냐하면 만약 다룰 수 없다면, 비정상 상태(illegal state)들이 부적절한 행위을 일으킬 수 있기 때문이다. 각 상태 머신에 대한 요건들이 반드시 결정되어야만 한다. 그리고 주기적인 로컬 리셋이나 POR 혹은 다른 리셋 명령어들에 의해서 비정상 상태들이 논리상으로 다뤄져야만 한다. 설계자들은 글리치(glitch)들에 대한 영향, 그러한 글리치로부터의 복구등에 집중하지 않고, 단지 회로의 정상적인 기능에만 집중하는 경우를 볼 수 있다. 예를 들어 전원 과도기(transient)들, 방사(radiation), 혹은 ESD로부터 글리치들이 발생될 수 있다. 그러나 리셋 논리가 생성되었다고 해서 FPGA가 비정상 상태(illegal state)일 때, 올바른 기능을 수행 할 것이라는 것을 의미하는 것은 아니다. Synplicity의 합성기로부터 생성된 리셋 논리의 미묘한 점은 어떤 조건에서는 half-edge 플립플롭(예를 들어, 상승(rising) 플립-플롭에서 falling edge 플립-플롭이 사용되는 경우)이 리셋을 만들기 위해서 사용된다는 것이다. 설계자들은 대부분 이를 인식하지 못한다. 왜냐하면 Synplicity가 이를 밝혀내지 못하기 때문이다. 그리고 이에 대한 타이밍 분석이 되지 않는다. 이 타이밍 분석은 주기적이지 않고 상당히 변화될 수 있는 상태 머신의 클럭의 순환주기(duty cycle)에 달려있다. (This timing analysis relies on the duty cycle of the state machine's clock, which may vary considerably, and not the period, which is generally quite accurate, as crystal controlled clock oscillators are the norm.)

(2) 다음 섹션은 부품 복제(replication)에 대해서 논의한다. 부품 복제(replication) 중에서 가장 중요한 것은 플립-플롭 복제이다. 그리고 플립-플롭 복제(replication) 중에서 가장 중요한 부분은 비동기 신호를 동기화 시키는 플립-플롭의 복제(replication)이다 물론 동기화의 플립-플롭의 복제(replication)가 허용된 것은 아니다. 일반적으로 만약 과도기적 사건(transient event)들로 인해 논리적으로 동일한 플립-플롭들이 서로 다른 값을 갖는다면, 복제된 플립-플롭들은 부적당한 동작을

일으킬 수 있다. 따라서 플립-플롭 복제는 피해야만 한다. 그럼에도 불구하고 만약 설계에 복제된 플립-플롭이 사용되었다면, 각 인스턴스(instance)에 대한 허용성(acceptability)이 반드시 철저하게 분석되고 문서화되어야 한다. 이 작업은 노동 집약적(labor intensive)이고 에러가 수반되며 반드시 각 합성기가 동작된 이후에 수행되어야 한다.

(3) 다음 섹션은 사용되는 논리 타입의 리스트들을 보여준다. 아래 타입들이 사용되었는지를 알기 위해서는 FPGA 제조사의 매크로(macro) 라이브러리(library)에 사용된 플립-플롭들과 비교해라.

- 셋(set)이나 클리어(clear)가 없는 플립-플롭들. 이것은POR이나 리셋 명령어로 리셋이 되지 않는 회로를 지칭한다.
- 셋(set)이나 클리어(clear)를 갖고 있는 플립-플롭들. 이것은 가능한 비동기적 설계 기술을 가리킨다. (셋(set)/클리어(clear)의 부재가 비동기적 설계 기술을 가리키는 것은 아니다.)
- 래치들, 반드시 타이밍을 직접 체크해야 한다.
- 반대 에지 플립-플롭들(예를 들면, 주로 상승 에지로 이루어진 설계에서 하강 에지 플립-플롭의 사용), 이는 클럭 대칭(symmetry)에 제한사항(constraint)을 발생시키고 타이밍 증명기(verifier)로 분석하기에 어렵다. 위에서 주목했듯이, the "safe? Attribute 의 사용으로 몇몇 반대 에지 플립-플롭들이 생겨날 수 있고 설계자들은 이 플립-플롭의 존재여부에 대해서는 종종 모른다. 위에서 언급한 사항들이 반드시 설계 에러를 일으키는 것은 아니지만, 반드시 체크되어야 한다. 몇몇 HDL 코딩 에러들로 인해 예상치 못한 래치들 혹은 셋(set)/클리어(clear) 플립-플롭들이 생겨날 수 있다.

(4) 다음 섹션은 합성기에 의해서 발견된 클럭들에 대해서 논의한다. 위에서 논의된 논리 타입 리스트들은 클럭 리소스들 중 어느 것이 사용되었는가에 대해서 주목할 것이다. 그리고 "clock found?or "clock inferred?와 같은 statement들을 제공한다.

- 만약 로컬 클럭들이 사용된다면, (즉, 글로벌 클럭 리소스를 사용하지 않는 클럭들) 그것들은 여기에 보여질 것이다. 예를 들면, 3개의 클럭 드라이버를 갖고 FPGA에는 4개의 클럭이 있을 때. 로컬 클럭은 잠재적으로 허용할 수 있는 것보다 훨씬 더 높은 스큐(skew)를 갖고 있다. 따라서 순차적으로 서로 인접해 있으며, 같은 에지에 트러거되는 플립-플롭들의 클럭킹에 사용되서는 안된다.
- 만약 routed 클럭이 RT54SX-S, RTSX-SU, or A54SX-A 다바이스에 사용된다면, 회로가 적당한 스큐 허용오차(tolerance) 설계 기술이 사용되었음을 반드시 보여야 한다.
- 표는 어느 클럭 에지들이 그들 사이에서 논리를 갖고 있는지 보여줄 것이다. 예를 들면, HCLK의 상승 에지에서 하강 에지, 혹은 다른 클럭들의 에

지들. 클럭 도메인을 교차하는 논리와 두 에지가 사용되는 클럭의 대칭(symmetry) 요건들을 면밀히 조사해라.

.srr 파일의 나머지 부분들은 배치와 배선(place and route)전에 계산된 타이밍 분석 정보를 담고 있다. 따라서 미심쩍은 값이다. 정확환 타이밍 분석은.abd파일에 접근하여 Timer에 의해서 보여질 것이다 Timer는 최소값을 보장하지 않을 것이다. 그러나, 하프(half)-에지 클럭 플립-플롭들 혹은 자기 안에 있는 어떠한 비동기적 기술들에 대해서는 분석하지 않을 것이다. 그러한 인스턴스(instance)들은 Timer와 함께 손수 분석해야 한다.

.srr 파일을 읽을 때, 파일안에 있는 어떠한 경고들도 주의깊게 주목해라. 설계는 경고가 있었음에도 합성할 수 있다. 그러나 각각의 경고는 각 합성이 실행된 이후, 반드시 dispositioned되야 한다.

3) 백-엔드(Bacf-end) 툴: .adb 파일 이용

.abd 파일은 실질적인 넷리스트(netlist), 타이밍 분석, 핀 정보, 기타 등을 표함한 설계의 상세한 내용들을 담고 있다. 비록 보기에 불편할지라도(대부분 스키메틱 생성기가 만들어 냈듯이), 검토자가 스키메틱 상에서 설계를 볼 수 있도록 하기 위해서 넷리스트(netlist) 뷰어(viewer)를 넣어야 한다. 위에서 강조했듯이, HDL description 보다는 여기에 주어진 스키메틱이 좀더 실질적인 설계이다.

(1) 타이밍 분석이 했던 온도, 전압 방사(radiation) 세팅을 체크해라. 이러한 것들은 프로그램 방사(radiation) 요건이 무엇이든 간에, 온도 전압에 대해서는 완전하게 군사용 범위가 되야 한다. 만약 분석이 축소된 온도와 전압 범위에 대하여 이루어졌다면, 축소(reduction)를 정당화하기 위한 분석이 반드시 이루어져 한다. 온도는 보드의 열 제어 표면(thermal control surface)의 온도가 아니라 다바이스의 접점 온도(junction temperature)라고 툴이 가정한다는 사실에 주목해라.

(2) 얼마나 많은 타이밍 마진이 있는지 Timer를 동작시켜라. 위에서 언급한대로 모든 군사 범위가 사용되었을지라도, 노화(aging), 부정확성(inaccuracy), 기타 등에 대한 약간의 마진이 있어야만 한다. 전파 지연의 경우 A ?/ -> 10%의 마진이면 적당하다.

(3) 어떤 I/O 옵션들이 선택되었는지를 결정하기 위해서 Pinedit를 동작시켜라. 선택된 I/O옵션들과 스키메틱 상에서 보여지는 입력 출력과 비교하면서, I/O 옵션의 선택이 적절했는지 증명해라.

(4) 2.12.4.2 절에서 발견된 이슈들을 해결하기 위해서, 특히 2.12.4.3(다)항에서 이상한 플립-플롭들의 사용을 이해하기 위해서 Netlist Viewer를 열어서 스키메틱을 봐라.

(5) Netlist viewer에서 스키메틱을 통해 게이팅(gating)된 셋(set)들과 리셋(reset)들을 자세하게 찾고, 모든 셋가능한(settable) 그리고 리셋가능한(resettable) 플립-플롭들이

유효한 리셋과 연결되었는지, 그리고 비동기적 설계 기술에 포함되지 않도록 확실히 해라. 리셋 혹은 POR 입력에서 출발하여, 리셋 라인들은 하이라이트(highlight) 될 수 있고 모든 페이지를 통해 따를 수도 있다.

(6) Netlist viewer에서 임무와 안전에 결정적인 회로들이 올바르게 구현되었는지 확실히 해라. HDL 합성기가 클럭 생성 회로에서 정적 해저드(hazard)와 같은 회로를 구현했는지 관찰되야 한다.

4) POR과 리셋 회로와 Power-up 조건에 대한 검토

이상적인 POR은 전원 공급기를 켜자마자 asserted되고 전압이 유효한 전압 수준으로 될 때까지 asserted를 유지한다. 몇가지 요인들로 인해서 POR이 power-up 주기에 있는 포인트 그 이상으로 asserted되는 것이 필요할 수도 있다.

(1) 보드에 오실레이터가 있다면, 오실레이터가 적절한 동작을 하기 시작할 때까지(이는 전원 공급이 유효한 수준까지 도달된 후, 수십 밀리초 정도 소요될 것이다.), asserted를 유지해야만 한다.

(2) 리셋되기 위해서 오실레이터를 필요로 하는 플립-플롭들이 있다면, 플립-플롭들이 리셋될 때까지 리셋은 반드시 asserted되야 한다.

이것 이외에도, FPGA의 임계성(criticality)과 기기 혹은 우주선에 해를 끼칠 수 있는 잠재성으로 인해, 리셋과 power-up/down 조건에 대한 면밀한 검토가 필요로 할 수도 있다. Power-up 시간 중 일부 시간 동안은 FPGA는 회로가 아니라 단순히 연결되지 않은 게이트들의 집합체이다. 따라서 과도기(transient)가 출력에 나타날 수도 있다.

Power-up/power-down, 혹은 전압 저하(brown-out)동안에 해를 끼치거나, 의도하지 않은 동작을 일으킬 수 있는 외부 전압은 이러한 기간 동안에 안전하게 동작됨을 증명하기 위해서 주의깊게 검토해야 한다. 예를 들어, 무장(arm)과 발사(fire) 메커니즘을 구성하는 회로는 무장(arm)신호와 발사(fire) 신호가 power-up, power-down이 동시에 일어나는 FPGA에서 시작되어서는 안된다. 또한 충분한 검토없이, FPGA가 POR신호들을 다른 회로로 넘겨서도 안된다 이는 종종 문제의 원인이 되었다. 외부 컴포넌트들은 특별한 리셋 요건이 존재하는지를 결정하기 위해 검토되야 한다. 이러한 부류 중에서 EEPROM이 주목할 만하다. 왜냐하면 power-on, power-off와 전압저하(brown-out)와 같은 기타 다른 과도기(transient) 동안 EEPROM은 반드시 보호되어야 하기 때문이다.

5) 정확한 동작을 증명하기 위해 전체적인 설계에 대한 검토

설계 검토 중 더 어려운 부분은 회로가 명세서(specification)대로 동작하는지를 증명하는

것이다. 위에 봤듯이, 제한되어 있는 것은 아니지만, HDL을 들여다 보고 HDL이 진술된 대로 회로가 동작하는지를 결정하는 것들이 증명에 포함된다. HDL과 관련된 문제중의 하나는 비록 설계가 합리적인 크기의 모듈들로 분해될지라도, 모듈들 사이의 연결 가능성을 결정하기가 어렵다는 점이다. 이러한 문제를 다루는 한가지 방법은 다음과 같다.

(1) Import the HDL 모듈들을 Actel Libero로 import해라.
(2) 각 모듈의 심볼을 만들어라.
(3) ViewDraw(Libero 툴셋 안에 있음)를 사용하여 심볼들을 큰 시트에 놓고 그것을 출력해라.
(4) 모듈들 사이의 연결을 그리기 위해서 색깔이 있는 펜이나 하이라이터(highlighter)를 사용해라.

이것은 FPGA 설계에 대한 높은 수준의 관점(view)을 제공해 줄 것이다. 각 모듈은 계층적 설계 안에서 하나의 컴포넌트 역할을 하기 때문에 문서화해야 한다. 이상적으로, 컴포넌트들은 카운터, 디코더, 논리 유닛, 시퀀서(sequencer), 기타등과 같은 디지털 시스템의 빌딩 블록들의 필수적인 구조에 기반을 둘 것이다. 임의의 기능을 하는 컴포넌트들은 신뢰성 시스템의 설계뿐 아니라 그것들의 증명도 어렵고 에러가 일어날 수 있게 끔 만든다. 하드웨어를 마치 소프트웨어로 다루는 설계 접근 방법 때문에, 이러한 일들이 발생한다.

검토의 나머지 부분은 모듈을 조사하고, 다양한 기능이 어떻게 구현되었는지, 요건들을 만족하는지를 정하기 위해서 명세서를 읽는 것이다. 만약 이들의 동작이 모호하다면 테스트 벤치들을 작성하고 몇몇 모듈들을 시뮬레이션하거나, 혹은 모듈의 한 부분에 대해서도 시뮬레이션 하는 것이 유용할 수도 있다.

FPGA와 보통 설계자들이 분석하지 않은 외부 컴포넌트들 사이의 인터페이스에 대한 타이밍에 세심한 주의를 기울여라.

제 5 장 FPGA 코딩 지침

제 1 절 개요

원자력발전소에 적용을 목적으로 개발되는 시스템에는 높은 안전성 및 신뢰성이 요구된다. 특히, 안전-필수(Safety-Critical) 등급의 시스템은 가장 높은 안전성을 요구받고 있다. 따라서 FPGA 기반의 디지털 시스템이 안전-필수 등급의 시스템에 적용되기 위해서는 시스템이 안전하고 신뢰성이 있게 개발해야 하고 이를 입증할 수 있어야 한다. 특히, FPGA 기반 시스템의 경우, 시스템의 모든 주요 기능들이 FPGA에 구현되기 때문에, FPGA에 구현되는 로직의 안전성 및 신뢰성이 시스템의 안전성 및 신뢰성을 결정한다고 볼 수 있다.

본 장에서는 원자력발전소의 FPGA 기반의 계측제어계통을 설계하기 위하여 사용되는 하드웨어 명세 언어(Hardware Description Language)에 대한 코딩지침을 기술한다. 이러한 코딩지침은 설계자가 FPGA에 요구된 기능을 구현하기 위하여 하드웨어 명세 언어를 사용하여 코드를 작성 시, 코드에 오류가 코드에 삽입될 수 있는 가능성을 최소화할 수 있도록 설계초기 단계에서부터 참고할 수 있도록 하는 것이다.

본 장에서 기술되는 코드 작성지침은 FPGA 코드 작성단계에서 참고될 수 있다. 그러나 하드웨어명세언어로 작성되는 코드의 구조 및 알고리즘은 구조 및 상세설계 단계에서 결정되고, 본 코드 작성지침은 설계관련 지침도 제공하고 있으므로, 설계자는 FPGA 개발을 위한 구조 및 상세 설계 단계에서부터 참고할 수 있다.

현재 가장 많이 사용되는 하드웨어 기술 언어는 VHDL과 Verilog이다. 본 코딩지침에서는 VHDL로 작성된 코드를 지침의 예로 사용하였으나, Verilog를 이용한 코드 설계자도 본 코딩 지침을 사용할 수 있다.

제 2 절 하드웨어 명세 언어

일반적으로 FPGA에 기능을 구현하기 위하여, IEEE 1076의 기준에 따른 VHDL, 또는 IEEE 1364의 기준에 따른 Verilog와 같은 하드웨어 명세 언어로 작성된다. 하드웨어 명세 언어는 설계자가 하드웨어 설계의 거동을 모델링 할 수 있도록 해주는 중요한 설계도구이다. 또한 하드웨어 명세 언어는 하드웨어 설계를 위한 기능적 및 물리적 분할을 가능하게 해준다.

```
process (reset, CLK20M)
begin
    if reset='1' then
        CntCLK20M <= (others=>'0');
    elsif rising_edge(CLK20M) then
        CntCLK20M <= CntCLK20M + '1';
    end if;
end process;
```

그림 5.1 VHDL 코드 예

VHDL은 복잡도가 소수의 게이트부터 다수의 복잡한 집적회로 상호연결까지 다양한 디지털 시스템의 동작을 기술하거나 시뮬레이션하기 위행 사용될 수 있는 다목적 하드웨어 명세 언어이다. VHDL은 디지털 시스템을 동일 표준으로 기술하기 위행 미국 국방성의 자금지원으로 개발되었다. VHDL이 개발되었을 때, 주 목적은 하드웨어를 명료하게 기술하고 문서화하기 위한 메커니즘을 갖는것이었다. 고-수준의으로 표현된 하드웨어를 합성하는 것은 원래 목적 중의 하나가 아니었다. 그 후, VHDL 언어는 IEEE 표준이 되었고 계속적으로 개정되었다. 그림 5.1은 VHDL로 작성된 코드를 보여준다.

Verilog는 미국 국방성이 VHDL을 새로 만들기 위해 자금을 지원했던 동일한 시기에 기업체에 의해서 개발되었다. Verilog는 1984년 Gateway Design Automation사에 의해 특허가 있는 HDL로 소개되었다. Synopsys사는 1988년경에 Verilog를 위한 합성도구를 개발하였고 1995에 IEEE 표준이 되었다. 그림 5.2는 Verilog로 작성된 코드를 보여준다.

```
Always @(posedge clock)
begin
    temp_a = a;
    temp_b = b;
end
```

그림 5.2 Verilog 코드 예

하드웨어 명세 언어의 모델링 능력은 1)시간적인 모델 능력, 2) 데이터 추상화 능력, 3) 기능 모델 능력, 4) 구조 모델 능력과 같은 4가지의 특성을 통해서 기술할 수 있다. 그림 5.3은 VHDL의 모델링 능력을 기술하고 있다. 회색으로 표시된 영역이 VHDL에 의해 지원되는 모델링 능력이다. 그림 5.4는 Verilog의 모델링 능력을 기술하고 있다. 회색으로 표시된 영역이 Verilog에 의해 지원되는 모델링 능력이다.

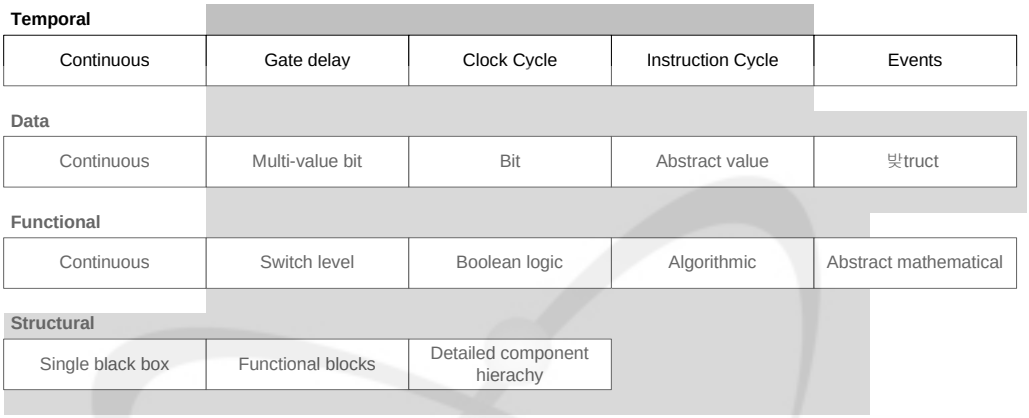


그림 5.3 VHDL 모델링 능력

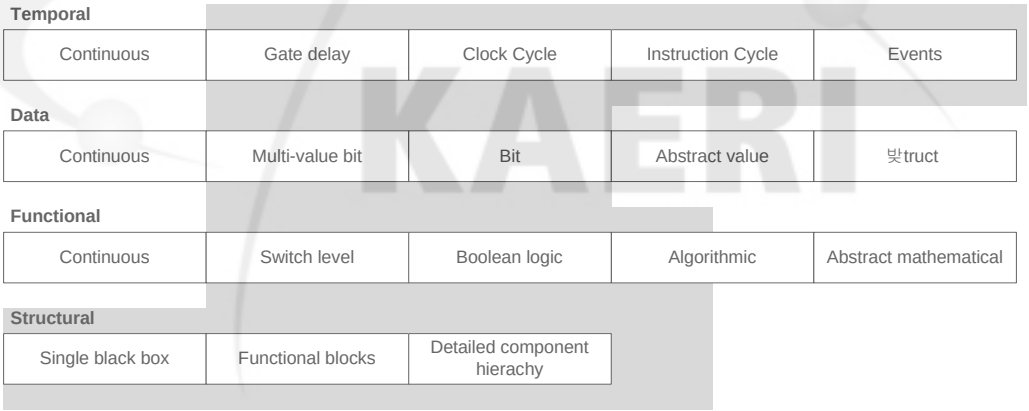


그림 5.4 Verilog 모델링 능력

본 코딩지침은 다음과 같은 코드의 속성을 고려하여 코드의 작성지침을 제공한다. 각 속성은 하위의 상세 속성들을 포함하고 있다.

기능 정확성(Functional Correctness)

타이밍 정확성(Timing Correctness)

합성 가능성(Synthesizability)

유지보수성(Maintainability)

기능 정확성이란 HDL 코드가 예측가능하고, 일관되고, 정확하게 요구된 기능을 수행할 수 있는 코드의 속성이다. 기능 정확성을 위한 코딩지침은 3절에 기술된다. 3절에서는 HDL 코드가 HDL 코드에 오류가 삽입될 수 있는 여지를 최대한 줄일 수 있도록 설계자가 '설계의도'를 명백하고 명시적인 방법으로 코드를 작성하여 HDL 코드가 기능 정확성을 만족시킬 수 있도록 하기 위한 지침을 제시하고 있다. 3절에서는 기능 정확성을 문장 법칙(Syntax) 측면과 설계 구조(Structure) 측면으로 구분하여 코딩지침을 제시한다.

타이밍 정확성이란 HDL 코드가 예측가능하고, 요구된 응답시간을 만족하면서 하드웨어 로직으로 구현될 수 있는 코드의 속성이다. 타이밍 정확성을 위한 코딩지침은 4장에 기술된다. 4절에서는 설계자가 로직의 경쟁문제(race problem), 글리치(glitch), 또는 다른 타이밍 관련 문제를 갖지 않도록 코드를 작성하도록 하기 위한 지침을 제시하고 있다. 4장에서는 타이밍 정확성을 경쟁문제, 클록 게이팅(clock gating), Zero Time 글리치, 클록 도메인 글리치 측면에서의 코딩지침을 제시한다.

합성가능성이란 RTL 수준의 HDL 코드가 게이트(gate)나 플립플롭(flipflop)과 같은 실제 하드웨어 로직으로 변환될 수 있는 코드의 속성이다. 합성가능성을 위한 코딩지침은 5절에 기술된다. 5절에서는 합성가능성을 동기 프로세스와 비동기 프로세스 측면으로 구분하여 코딩지침을 제시한다.

유지보수성이란 필요에 의해서 HDL 코드의 변경 시, 설계자가 오류를 삽일할 가능성을 줄이도록 하는 코드의 속성이다. 6절에서는 파일 관리, 코드 형식, 설명문 및 이름 명명 측면에서 코딩지침을 제시한다.

제 3 절 기능 정확성

본 절에서는 HDL 코드의 기능 정확성을 위한 코딩지침을 제시한다. 3.1절에서는 문법규칙 측면에서의 코딩지침을 제시하며, 3.2절에서는 설계구조 측면에서의 코딩지침을 제시한다.

1. 문법규칙(Syntax)

1) 피연산자의 길이

■ 모든 연산에 사용되는 피연산자는 같은 길이(width)를 가져야한다. 즉, 설계자가 두 개 이상의 피연산자를 이용하여 연산을 수행할 경우, 의도하지 않은 오류가 삽입되지 않도록 같은 길이의 피연산자를 이용하여 연산을 수행하도록 코드를 작성해야 한다.

■ 연산의 결과가 저장되는 변수 또는 신호는 연산결과를 충분히 수용할 수 있을 만큼 충분한 길이 또는 크기를 가져야 한다.

아래의 예는 바람직한 코드의 작성 및 바람직하지 않는 코드작성의 예를 보여준다.

```
SIGNAL x: STD_LOGIC_VECTOR (31 DOWNTO 0) ;
SIGNAL y: STD_LOGIC_VECTOR (31 DOWNTO 0) ;
SIGNAL z: STD_LOGIC_VECTOR (31 DOWNTO 0) ;

-- good example
z             <= x AND y ;
z(11 DOWNTO 3)     <= x(10 DOWNTO 2) AND y(8 DOWNTO 0) ;

-- bad example
z <= x(9 DOWNTO 1) AND y(8 DOWNTO 1) ; -- error: unequal operand width
z <= x & y ; -- error: unequal width between z and result of operation
```

2) 의도되지 않는 래치(latch) 생성

■ 의도하지 않은 래치가 생성되지 않도록, 모든 조건문은 완전해야 하고 중복조건이 없어야 한다. 즉, 설계자가 조건문(case 문, 또는 if-then-else 문)을 작성할 때, 모든 가능한 상황에 대해서 코드들 작성하여야 한다.

HDL 코드에 불완전한 조건문을 작성하면 설계된 로직에 의도되지 않은 래치가 생성될 수 있다. 아래의 예와 같이 1-bit 신호, enable이 1이면, data_out에는 data_in의 값이 할당된다. 그러나 enable이 0이면, data_out은 data_out이 현재 가지고 있는 값을 그대로 유지한다. 즉, 아래의 코드는 보기에는 2-1 멀티플렉서(multiplexor) 설계를 위한 코드 같지만 enable이 가질 수 있는 1 이외의 값에 대하여 data_out에 할당되는 값이 정의되지 않았기 때문에 설계자가 의도했던, 의도하지 않았던 래치를 갖는 로직, 즉, 메모리의 기능을 갖는 순차회로가 생성된다. 따라서 설계자가 메모리 기능을 갖는 회로를 원했다면, 이를 명확하게 명시적으로 코드를 작성하여야 하고, 설계자가 메모리 기능을 갖는 회로를 원하지 않으면, 조건문의 모든 경우에 대하여 명확하게 코드를 작성하여야 한다.

example : PROCESS (enable, data_in)

```
BEGIN
    IF enable = '1' THEN
        data_out <= data_in ;
    ENDIF ;
END PROCESS example ;
```

아래의 예는 IF 문을 사용하여 2-1 멀티플렉서를 설계하기 위한 코드의 작성 예이다. 아래의 예와 같이, enable이 1 이외의 값을 가질 경우에 data_out에 0(또는 1)의 값을 할당하여, 조건문을 완성함으로써, 의도하지 않은 래치가 생성되는 것을 방지할 수 있다.

```
example : PROCESS (enable, data_in)
BEGIN
    IF enable = '1' THEN
        data_out <= data_in ;
    ELSE
        data_out <= '0' ;
    ENDIF ;
END PROCESS example ;
```

■ 클록 기반의 동기회로는 프로세스문의 감지 리스트(sensitivity list)에는 비동기 리셋 및 클록 신호만이 포함되어야 하며, 조합회로만으로 구성된 프로세스문의 감지 리스트에는 출력 값을 변화시키는 원인이 되는 모든 신호가 포함되어야 한다.

조합회로만으로 구성된 프로세스문의 경우, 감지 리스트에 입력신호가 포함되지 않으면, 의도되지 않은 래치가 생성되어 순차회로가 생성될 수 있다. 아래의 코드는 x의 값이 변하더라도 y의 값에 전혀 영향을 주지 않는다. 즉, y는 z의 값이 변하기 전까지는 x의 변하기 전의 값에 의해 영향을 받으므로, 이러한 코드는 일종의 메모리 특성을 갖는 회로를 생성하게 된다.

```
example : PROCESS (z)
BEGIN
    y <= x & z ;
END PROCESS example ;
```

따라서 위에서 작성된 코드의 코드는 다음과 같이 작성될 수 있다.

```
example : PROCESS (x, z)
BEGIN
    y <= x & z ;
END PROCESS example ;
```

클록 기반의 동기회로를 설계하기 위한 프로세스문의 감지리스트에 리셋 및 클록 신호에 다른 신호가 포함될 경우, 시뮬레이션 속도가 낮아진다.

■ 설계자 오류 또는 합성도구에 의해 변수는 의도하지 않은 추가적인 래치를 생성할 수 가 있으므로, 특별한 장점이 있는 경우를 제외하고는 변수를 사용하지 말 것을 권장한다. 변수를 사용 할 경우에는 프로세스 문에서 변수가 읽혀지기 전에 먼저 특정 값이 할당되도록 코드를 작성해야 한다.

만약, 프로세스문에서 변수가 특정 값으로 할당되기 전에, 읽혀지도록 코드를 작성하면, 이러한 코드는 해당 변수가 이전의 할당된 값을 유지하기 위하여 의도하지 않은 래치를 생성할 수 있으며, RTL 모델의 시뮬레이션 결과와 합성된 모델의 시뮬레이션 결과가 다를 수 가 있다. 아래의 코드는 동기 프로세스에서 특정 값이 할당되기 전에 변수가 읽혀지도록 코드가 작성되어 의도하지 않은 래치가 생성되는 경우를 보여준다.

```
example : PROCESS (rst, clk)
    VARIABLE jk_v : STD_LOGIC_VECTOR (1 DOWNTO 0) ;
BEGIN

    IF (rst = '1') THEN
        q <= '0' ;
    ELSEIF (clk'EVENT AND clk = '1') THEN
-- reading variable, jk_v, before assigning a value implies registers on j and k
-- inputs
        CASE jk_v IS
            WHEN "00"       =>  q <=    q ;
            WHEN "01"       =>      q <= '0' ;
            WHEN "10"       =>      q <= '1' ;
            WHEN "11"       =>      q <=   NOT q ;
            WHEN OTHERS     =>      q <= 'X'
        END CASE ;
    END IF ;

    jk_v := (j & k) ;

END PROCESS example ;
```

아래의 코드는 비동기 프로세스에서 특정 값이 할당되기 전에 변수가 읽혀지도록 코드가 작성되어 RTL 모델의 시뮬레이션 결과와 합성된 모델의 시뮬레이션 결과가 달라지는 경우를 보여준다.

```
example : PROCESS (a, b, c)
    VARIABLE var :  STD_LOGIC_VECTOR (1 DOWNTO 0) ;
```

```
BEGIN
    -- In RTL simulation, an event in a, b, or c would fire the process,
    -- and the evaluation of the output will be based on a previously
    -- evaluated value of the variable var. The synthesized model evaluates
    -- the variable prior to making the assignment.

    IF var = "00" THEN
        q <= c (1 DOWNTO 0) ;
    ELSE
        q <= c (3 DOWNTO 2) ;
    END IF ;

    var := a & b ;

END PROCESS example ;
```

3) 포트 연결

■ 포트의 연결은 신호의 위치에 의한 것보다 이름에 의한 연결을 사용하여 코드를 작성해야 한다.

HDL에서 제공하고 있는 컴포넌트의 포트를 연결방법은 두 가지가 있다. 첫 번째 방법은 파생된 컴포넌트의 포트와 정의된 컴포넌트의 포트들과 같은 위치에 있는 신호들끼리 연결하는 위치연결(positional mapping)에 의한 방법이다. 다른 방법은 컴포넌트의 포트와 현재 설계 레벨의 신호의 이름을 모두 명시하여 신호를 연결하는 이름연결(nominal mapping)에 의한 방법이다. 그러나, 코드에 오류가 삽입될 가능성을 최소로 하기 위하여 이름에 의한 연결방법을 권장한다. 아래의 예는 이름에 의한 연결방법 및 위치에 의한 방법을 모두 보여주고 있다.

```
ENTITY example IS
    PORT (
            a1 : IN STD_LOGIC ;
            a2 : IN STD_LOGIC ;
            a3 : IN STD_LOGIC ;
            a4 : IN STD_LOGIC ;
            x : OUT STD_LOGIC
            ) ;
END example ;

ARCHITECTURE structural OF example IS
    COMPONENT logic_or IS
        PORT (
            a : IN STD_LOGIC ;
            b : IN STD_LOGIC ;
```

```
        c : OUT STD_LOGIC
            ) ;
    END COMPONENT ;

    SIGNAL n1 : STD_LOGIC ;
    SIGNAL n2 : STD_LOGIC ;
BEGIN
    U_1: logic_or  PORT MAP (a => n1, b => n2, c => x) ;
    -- In case of positional mapping
    -- U_1: logic_or    PORT MAP (n1, n2, x) ;
    U_2: logic_or  PORT MAP (a => a1, b => b2, c => n1) ;
    U_3: logic_or  PORT MAP (a => a3, b => b4, c => n2) ;
END ARCHITECTURE structural ;
```

4) 설계 구성

■ 코드 작성 시 특별한 이점이 없는 한, 루프문 사용을 피하여야 한다.

HDL은 LOOP, FOR, WHILE, RPEAT 같은 루프문을 제공하고 있으나, 이러한 루프문은 하드웨어 고유의 특성보다는 소프트웨어 특성과 비슷하여, RTL 설계 시, 이러한 루프문을 사용하면 하드웨어와 RTL 설계와의 연관성을 이해하기 어려우며, 확인 및 검증 작업을 복잡하게 한다. 따라서 이러한 루프문은 특별한 이점이 없는 한, 사용을 피하여야 한다.

■ 설계자는 RTL 수준의 코드를 작성할 때, HDL에서 제공하는 논리 게이트 프리미티브, 트랜지스터 프리미티브 및 사용자 정의 프리미티브 사용하지 말아야 한다.

Verilog HDL은 기본적인 논리 게이트(AND, NOR, NOT), 트랜지스터(PMOS, NMOS, CMOS, TRAN, TRIREG) 및 풀-업/다운 저항(PULLUP, PULLDOWN)을 프리미티브로 미리 제공하고 있다. 이러한 프리미티브는 게이트 수준 모델링 또는 스위치 수준의 모델링으로 RTL 수준의 모델링을 위해서는 사용을 피하여야 한다.

■ force/release, fork/join, event 및 wait 같은 하드웨어 특성보다 소프트웨어 특성이 큰 모델링 기법은 사용하지 말아야 한다.

2. 설계 구조

1) 루프 구조

■ 조합회로는 루프구조를 갖지 않도록 설계해야 한다. 또한 래치로 구성된 순차회로가 루프구조를 갖지 않도록 설계해야 한다.

그림 5.5는 루프구조가 생성된 조합회로의 예를 보여준다. 조합회로의 루프구조는 내재적으로 매우 높은 위험성을 갖는 구조이다. 루프구조를 갖는 조합회로의 동작은 일반적으로 루프구조 안에 있는 로직의 상대적인 전파지연(propagation delay)에 의존한다. 그러나 전파지연 시간은 고정되어 있지 않고, 바뀔 수 있어서, 루프구조를 갖는 조합회로의 동작도 이에 따라 변경된다. 또한 이러한 루프구조는 소프트웨어 설계도구에 의해 무한 반복계산을 수행하게 할 수 있다.

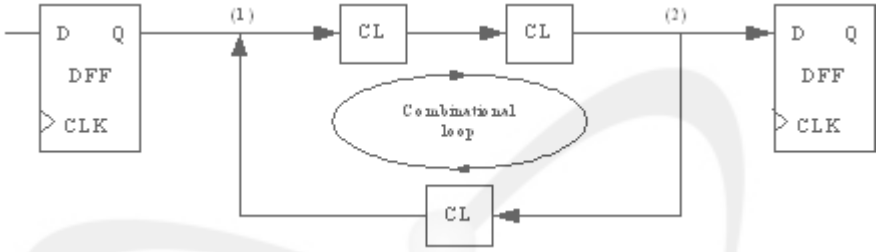


그림 5.5 루프구조 조합회로

그림 5.6과 같이 특정신호가 산술연산의 오른쪽(입력)과 왼쪽(출력)에 사용될 때, 이러한 루프구조를 갖는 조합회로가 생성될 수 있다. 즉, 신호 c는 AND 연산의 출력 및 OR 연산의 입력으로 사용되고, 신호 d는 AND 연산의 입력 및 OR 연산의 출력을 사용되었다. 이러한, 코드는 아래의 그림과 같은 루프구조를 생성하게 된다.

```
c <= b AND d ;
d <= a OR c ;
```

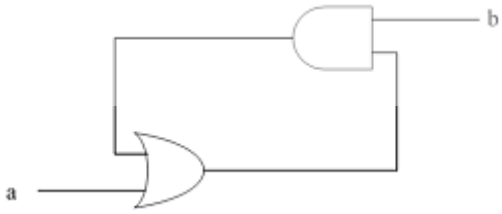


그림 5.6 루프구조의 조합회로 코딩 예

그림 5.7은 루프구조가 생성된 래치회로의 예를 보여준다. clock1이 high이고 clock2가 low이면 회로는 루프가 형성되며, 이런 경우에 회로는 조합회로의 루프구조와 같이 동작이 불안전해 진다.

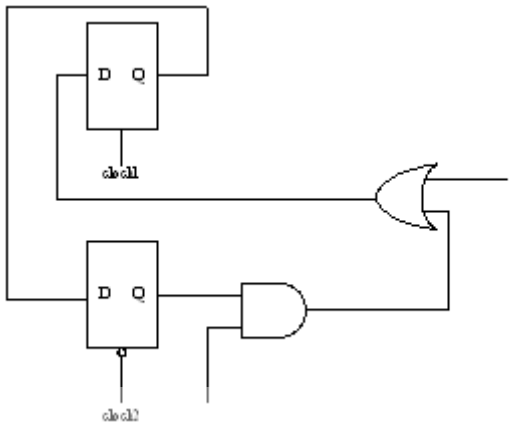


그림 5.7 루프구조 래치회로 예

2) 버스 설계

■ 여러 장치 간의 데이터 송수신을 위한 버스 설계 시, 버스에 연결된 모든 게이트에는 삼상태 버퍼(Tri-State Buffer)를 사용하여야 한다. 저항을 사용하여 풀-업(Pull-Up) 또는 풀-다운(Pull-Down) 버스를 설계할 시에는 버스가 stuck 되지 않도록 설계해야 한다.

■ 버스를 사용하는 권한은 오로지 한 장치만이 갖도록 설계해야 한다. 즉, 동시에 두 장치가 버스의 사용권한을 갖지 않도록 설계해야 한다.

보통, 두 개의 게이트나 플립플롭의 출력을 같이 연결하면, 회로는 올바르게 동작하지 않는다. 또한, 이러한 설계는 회로를 손상시킬 수 있다. 따라서 장치를 사이에 공용으로 사용되어 여러 장치 간에 데이터를 송수신하도록 하는 버스를 설계하기 위해서는 버스를 사용하지 않는 장치는 출력을 고 임피던스로 만들어 버스의 사용권을 양도하고, 버스를 사용하는 장치만이 버스에 연결되어야 한다. 이러한 버스를 설계하기 위한 한 가지 방법으로 삼상태 버퍼를 사용하는 것이다. 삼상태 버퍼는 그림과 5.8과 같이 여러 종류가 있다.

삼상태 버퍼를 사용하여 버스 설계되어 있는 경우, 버스에 연결되어 있는 모든 장치가 동시에 하이 임피던스 상태가 되면 전기적으로 버스에 아무것도 연결되어 있지 않는 것과 동일한 상태가 되어서 소자 파괴의 위험이 생기게 된다. 그렇게 때문에 하이 임피던스 상태를 수반하는 버스에는 저항을 이용하여 풀-업 또는 풀-다운을 연결한다. 그러나 이러한 설계는 버스가 High나 Low 값으로 stuck 되지 않도록 풀-업 또는 풀-다운을 위

한 적절한 저항 값을 선택하여 버스를 설계해야 한다.

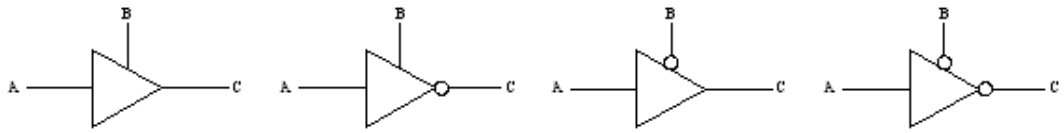


그림 5.8 삼상태 버스 구조

3) 순차회로 설계

■ 래치(Latch)보다는 플립플롭(Flipflop)을 이용하여 순차회로를 설계할 것을 권장한다.

순차회로를 구성하는 기본 요소는 래치와 플립플롭이다. 래치는 설계 시에 다양한 어려움을 동반한다. 래치는 플립플롭과 같이 메모리 기능을 갖지만, 이 둘은 기본적으로 차이가 있다. 래치가 인에이블(enable)되어 있을 때, 입력과 출력사이에 직접적인 통로가 생긴다. 이런 경우, 데이터 입력에 발생한 글리치(Glitch)는 곧바로 출력으로 전파된다. 또한 래치 인에이블을 위한 신호에도 글리치가 발생 할 수 있으며, 이러한 경우에는 전체 회로가 오동작 할 수 있다. 그리고 래치는 합성된 회로의 타이밍 분석이 매우 복잡하여 모든 조건에서 정확한 동작을 확인하기 어렵다. 따라서 순차회로의 설계 시, 특별한 목적이 아닌 한, 플립플롭 기반으로 순차회로를 설계하는 것을 권장한다.

4) 순차회로 초기화

■ 제어로직에 포함된 모든 순차 요소(sequential element) 들은 정해진 값으로 초기화가 가능하도록 설계되어야 한다. 즉, 초기에 전원이 공급되었을 때, 제어로직은 정해진 상태로 초기화되어야 한다.

제 4 절 타이밍 정확성

본 장에서는 HDL 코드의 타이밍 정확성을 위한 코딩지침을 제시한다. 4.1절에서는 경쟁(race) 상태 관련 지침, 4.2절 및 4.3절에서는 클록설계 관련 지침, 4.4절 및 4.5절에서는 글리치(Glitch)에 관련된 코딩지침을 제시한다.

1. 경쟁(race) 상태

가장 일반적으로 로직에서 발생할 수 있는 타이밍 관련 문제는 경쟁 상태가 발생하는 경우이다. 로직의 최종 상태가 여러 상태변수에 의해 결정되고, 이러한 상태 변수들이 변하는 순서에 따라 다른 상태가 된다면 경쟁 문제가 발생한다. 이러한 경우, 로직의 최종상태는 예측할 수 없으며, 시뮬레이터 또는 물리적인 하드웨어에 따라 다른 결과를 보여준다. 이러한 경쟁 문제는 회로에 단속적인(intermittent) 오류를 발생시켜, 문제를 확인 및 해결하기가 극도로 어렵다. 이러한 경쟁 문제가 발생하지 않도록 회로를 설계해야 한다.

■ (Verilog) 같은 클록 에지 또는 사건에 의해 트리거 되는 여러 블록(block)들에서 동일한 변수에 값을 할당하지 않도록 코드를 작성해야 한다.

아래의 Verilog HDL 코드의 예와 같이 동일한 클록에지에서 트리거 되는 두 개의 always 블록에서 동일한 변수, x에 다른 값이 할당되면 변수 x의 값은 시뮬레이터에 따라 다른 값을 가지는 경쟁 문제가 발생한다. 즉, 시뮬레이터에 따라 할당문 “x = 1'b1”이 먼저 실행될 수도 있고, 아니면, 할당문 “x = 1'b0”이 먼저 실행될 수 있다. 따라서 아래의 코드에서 x의 최종 결과는 시뮬레이터에 따라 ‘1’이 될 수도 있고 ‘0’이 될 수도 있는 비결정론적 특성을 갖는다.

```
always @(posedge clock)
   x = 1'b1 ;

always @(posedge clock)
   x = 1'b0 ;
```

■ (Verilog) 이벤트-카운팅(event-counting) 문제가 발생하지 않도록 코드를 작성해야 한다.

아래의 예는 이벤트-카운딩에 의한 경쟁 문제가 발생하는 경우를 보여준다. 즉, 아래의 코드는 x 또는 y의 값이 변경되면 event_number의 값을 1씩 증가하는 코드이다. 이 코드는 x와 y의 값이 동시에 변경되는 경우에 발생한다. x와 y의 값이 동시에 변경되면, event_number는 시뮬레이터에 따라 다른 값을 가질 수 있어서, 그 결과가 비결정론 적이다. 즉, 시뮬레이터에 따라, event_number의 값이 1 만큼 증가할 수도 있고, 2 만큼 증가할 수도 있다.

```
always @(x or y)
   event_number = event_number + 1 ;
```

■ (Verilog) 특정 변수가 동시에 쓰기-읽기가 되지 않도록 코드를 작성해야 한다.

아래의 예는 특정 변수에 다중 쓰기-읽기가 되도록 코드가 작성되어 레이스 문제가 발생하는 경우를 보여준다. 아래의 코드의 경우, 클록의 rising edge에서 y의 값이 변경(y = y + 1)은 연속적으로 x의 값이 변경(assign x = y + 2)을 발생시키고, 동시에 z의 값이 변경된다. 이 경우, z의 값은 변경 전의 x 값을 갖는지, 변경 후의 x 값을 갖는지 알 수 없다. 즉, z의 값은 비결정론적으로 된다.

```
assign x = y + 2 ;

always (posedge clock) begin
    y = y + 1 ;
    z = x ;
end
```

일반적으로 Verilog HDL에서 이러한 경쟁 문제를 해결하기 위해서는 블록킹(blocking) 할당문 대신에 논블록킹(nonblocking) 할당문을 사용해야 한다. 논블로킹 할당문은 읽기 동작과 쓰기 동작이 분리되어 실행된다. 즉, 논블로킹 할당문이 동시에 실행될 때, 읽기 동작동안 모든 논블로킹 할당문의 모든 오른쪽 변수의 값을 계산하여 임시로 저장한 후에, 쓰기 동작동안 임시로 저장된 값이 왼쪽의 변수에 할당된다. 결과적으로, 읽기-쓰기가 동시에 실행될 수 없다. 따라서 위의 코드는 아래의 코드와 같이 수정되면 경쟁문제를 해결할 수 있다.

```
assign x <= y + 2 ;

always (posedge clock) begin
    y <= y + 1 ;
    z <= x ;
end
```

그러나 논블로킹 할당문을 사용하더라도 동시에 쓰기-쓰기 경쟁 문제는 해결할 수 없다. 따러서, 아래의 예와 같이 논블로킹 할당문을 사용하더라도 수행의 결과는 비결정론적으로 되어 x의 최종결과는 예측할 수 없게 된다.

```
always @(posedge clock)
    x <= 1'b1 ;

always @(posedge clock)
    x <= 1'b0 ;
```

2. 클록 게이팅(Clock Gating)

글리치(Glitch)는 설계로직에 발생할 수 있는 원하지 않은 노이즈 펄스이며, 디바이스의 물리적 특성과 사용 환경에 따라 펄스의 폭이 변할 수 있다. 따라서, 글리치는 사용조건에 따라 어떤 때는 제거되기도 하고, 어떤 때는 로직에 일시적인 오류를 발생시킬 수 있다.

■ 실질적으로 전력소비를 줄일 수 있는 회로를 설계하기 위한 경우가 아니라면, 클록 게이팅을 사용하지 않기를 권장한다.

게이트 클록은 게이팅 회로에 의해서 제어되는 인에이블 신호를 사용해서 클록 신호를 ON 및 OFF 시킨다. 클록이 OFF 되면, 관련된 클록 도메인은 동작이 정지되어 기능을 수행하지 않게 된다. 따라서 게이트 클록은 전력소모를 줄이기 위한 회로 설계에 유용한 기술이며, 일반적으로 사용되는 설계방법 중에 하나이다. 그러나 정확하게 설계되지 않으면 래치 및 플립플롭을 잘못 트리거 할 수 있는 글리치를 발생시킬 수 있다. 그림 5.9의 예와 같이 클록의 상승에지에서 1번 플립플롭(Gating Flipflop)의 출력(Q, 게이팅 신호)이 High to Low로 바뀐다면, 클록부터 플립플롭의 출력까지의 지연에 의해서 클록 신호의 상승에지 신호보다 늦은 시간에 게이팅 신호가 AND 게이트에 입력된다. 따라서, AND 게이트의 출력에는 좁은 폭의 펄스인 글리치가 발생할 수 있다.

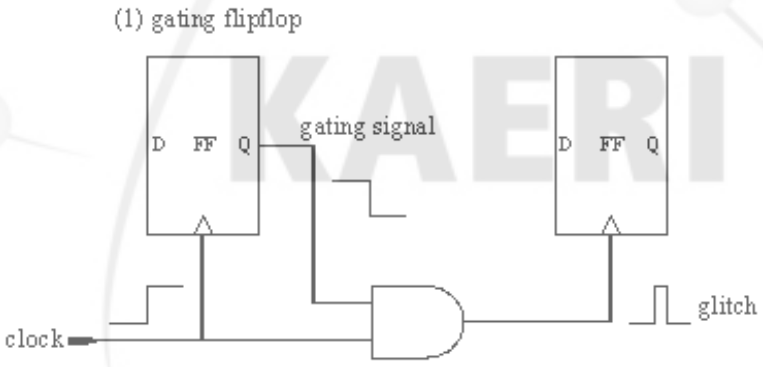


그림 5.9 잘못된 클록 게이팅 예

■ 클록 게이팅 설계에 사용될 때, 글리치를 방지하기 위하여 상승에지에서 게이팅 신호가 변경된다면, OR 게이트를 사용해야 하며, 하강에지에서 게이팅 신호가 변경된다면 OR 게이트를 사용해야 한다.

그림 5.10은 하강에지에서 게이팅 신호(enable)가 변경되는 경우에 대한 적절한 클록 게이팅 설계 예이다.

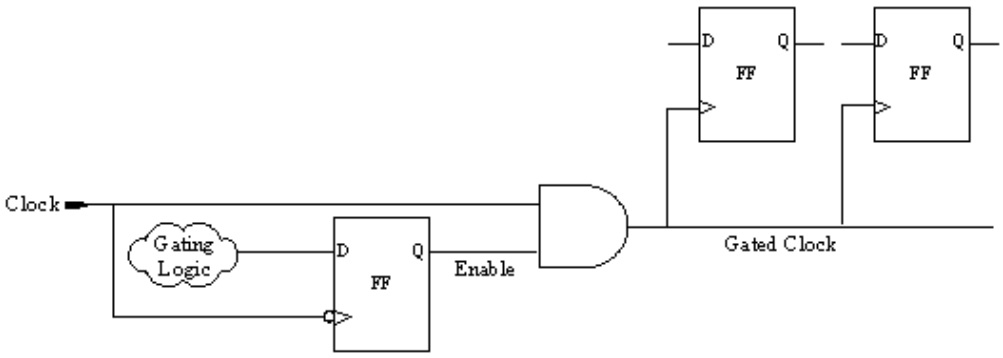


그림 5.10 적절한 클록 게이팅 예

3. 내부로직에서의 클록 생성

■ 로직의 내부에서 클록을 생성하지 않도록 코드를 작성해야 한다.

내부적으로 생성된 클록을 사용할 경우, 기능 및 타이밍 문제가 발생할 수 있다. 조합회로를 이용하여 생성된 클록은 글리치가 발생하여 오동작의 가능성이 있다. 또한, 조합회로에 의한 시간지연은 타이밍 문제를 발생시킬 수 있다. 만약, 조합회로의 출력이 클록신호 또는 비동기 리셋 신호로 사용된다면, 글리치가 발생할 수 있다. 동기회로에서 레지스터의 데이터 입력에 발생하는 글리치는 큰 영향을 주지 않는다. 그러나 클록입력에 발생하는 글리치는 회로에 지대한 영향을 줄 수 있다. 좁은 폭의 글리치는 레지스터의 최소 펄스 폭 요건을 위배할 수 있다. 또한 즉 클록입력에 글리치가 발생하였을 때, 레지스터의 데이터 입력이 변경된다면 셋업(Setup) 및 홀드(hold) 시간이 만족되지 않을 수 있다. 비록 이러한 설계가 타이밍 요건을 만족하더라도, 레지스터 출력이 기대되지 않은 값으로 변경될 수 있어서 기능에 오동작을 할 수 있다.

4. 시간 0 글리치 및 0 시간 글리치

■ (Verilog) 시간 0에서 어떠한 상태 변경이 발생하지 않도록 코드를 작성해야 한다.

“시간 0 글리치"는 시뮬레이션이 시작될 때(시간이 0)의 상태 변경이 발생하도록 코드가 작성되었을 때 발생한다. 예를 들어, 클록이 시간 0에서 1로 할당되었을 때, 클록의 값이 unknown 값에서 1로의 상태변경이 상태 변경으로 간주되는지는 시뮬레이터에 따라 다르게 해석될 수 있다. 따라서, 시간 0에서 상태변경이 일어나지 않도록 클록에 값을 할당하는 것이 바람직하다.

■ (Verilog) 한 클록 주기 안에서 동일한 변수에 다중의 다른 값을 할당하지 않도록 코드를 작성해야 한다.

“0 시간 글리치”는 동일한 변수에 다른 값을 동시에 할당하는 경우에 발생한다. 아래의 예와 같이, current_state의 값이 ACK(또는 IDEL)인 경우에, next_state는 항상 RESET에서 IDLE(또는 REQ)로 곧바로 변경된다. 즉, next_state 와 glitch_line에 0 시간 글리치가 발생한다. 이러한 0 시간 글리치를 제거하기 위해서는 한 클록 사이클 안에서 동일안 변수에 다중의 다른 값을 할당하는 것을 피하여야 한다.

```
always @(posedge clock) begin
    next_state = RESET;   // initialize next_state to RESET
    case (current_state)
        ACK: next_state = IDLE;
        IDEL: next_state = REQ;
    end

assign glitch_line = next_state; //executed every time next_state changes
```

5. 도메인 교차 글리치

■ 다른 클록 도메인으로부터 입력되는 신호는 동기화되어야 한다.

다른 클록 도메인이 연결되는 로직에는 쉽게 글리치가 발생할 수 있다. “클록 도메인”이란 동일한 클록에 의해 동작되는 로직의 그룹을 말한다. 두 클록 도메인의 연계로직은 두 클록 도메인에서 오는 입력신호를 입력으로 사용하는 로직이다. 예를 들어, 두 개의 입력을 받는 AND 게이트가 특정 클록을 사용하는 로직으로부터 하나의 입력을 받고 이와는 다른 클록을 사용하는 로직으로부터 다른 하나의 입력을 받을 경우, AND 게이트는 두 클록 도메인의 연계로직이다. 이러한 경우, 두 개의 입력신호가 동기화가 되어있지 않으면, 두 도메인에서의 상태 변경은 시간적 차이가 있으며, 이러한 임의의 작은 시간적 차이는 AND 게이트의 출력에 글리치를 발생시킬 수 있다. 따라서 다른 클록 도메인에서 입력되는 신호는 그림 5.11과 같이 적절한 동기화가 이루어져야 한다.

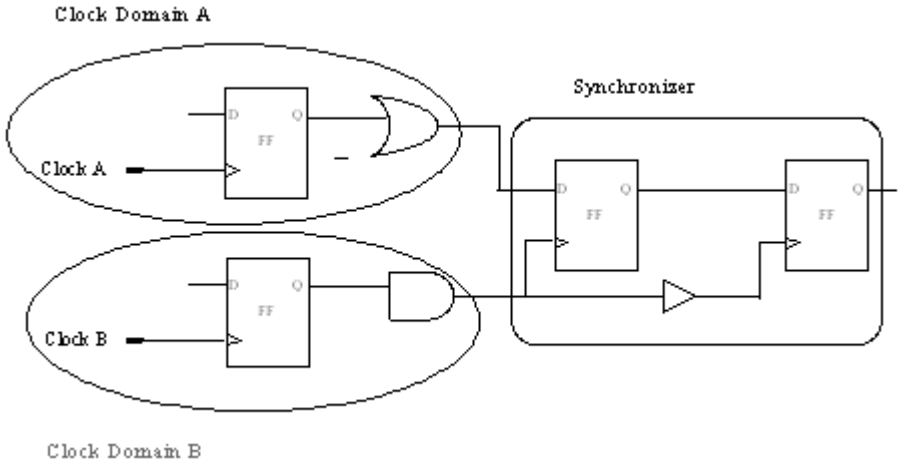


그림 5.11 다른 클록 도메인의 동기화 예

제 5 절 합성성

설계자는 PLD 또는 FPGA에 하드웨어 로직을 구현하기 위하여 VHDL 또는 Verilog HDL을 사용하여 레지스터 전송 수준(Register Transfer Level)의 HDL 코드를 작성한다. 이렇게 작성된 코드는 하드웨어로 구현되기 전에 로직을 합성하기 위한 소프트웨어 도구에 의해 게이트-수준(Gate-Level)의 넷-리스트(Net-List)로 변환된다.

그러나 VHDL 또는 Verilog HDL이 제공하는 모든 구성 요소가 합성도구에 합성되는 것은 아니며, 이러한 하드웨어 언어가 제공하는 일부의 구성 요소만이 합성에 사용될 수 있다. 따라서 설계자는 이러한 측면을 고려하여 합성 가능한 HDL 코드를 작성해야 한다.

로직합성 도구의 합성능력이 향상될수록 로직합성 도구에 의해 합성 가능한 HDL 구성요소는 증가하며, 합성도구에 따라 다른 합성 능력 및 성능을 가지고 있다. IEEE 1076.6 "IEEE Standard for VHDL Register Transfer Level(RTL) Synthesis"은 VHDL로 코드 작성 시, 합성 가능한 VHDL의 구성 요소를 기술하고 있다. VHDL을 이용하여 코드를 작성하고자 하는 설계자는 본 기술기준을 참고할 것을 추천한다.

- 비동기 로직보다는 동기 로직으로 회로를 설계하는 것이 바람직하며, 비동기 로직을 사용한다면, 동기 로직과 엔티티를 분리하고 구조적(structural) 형식보다는 행위적(behavioral) 형식의 코드를 작성할 것을 권장한다.

과거에 설계 자원을 효율적으로 사용하기 위하여 설계자들은 리플 카운터 또는 펄스 생산기 같은 비동기 방식의 설계를 사용하였다. 그러나 비동기 설계방법은 디바이스의 전

달지연에 의존하여, 불완전한 타이밍 문제 및 글리치에 의한 기능오류 문제 등을 갖고 있다. 비동기 설계 구조는 타이밍 할당 및 제한치를 모델링하기가 어렵거나 또는 불가능할 수도 있다. 만약, 정확한 타이밍 제한치가 없다면 합성도구와 배치 및 배선(place and route)도구는 이러한 비동기 설계구조의 최적화를 할 수 없을 수 있어서, 그 결과가 완전하지 않을 수 있다. 이에 반해, 동기 설계는 클록 신호에 동기화되어 모든 로직이 작동한다. 즉, 모든 데이터 입력은 클록 신호가 상승에지 또는 하강에지일 때만, 출력의 값을 변경시킨다. 따라서 동기 설계의 데이터 입력에서 발생하는 글리치는 로직의 오동작을 발생시킬 확률이 거의 없다. 이러한 이유로, 비동기 설계에 의한 특별한 장점이 없는 한, 동기방법을 이용하여 로직을 설계할 것을 권장한다.

- 지연요소(Delay Element)는 합성 및 타이밍 검증 문제를 발생시킬 수 있으므로 RTL 설계에서 사용하지 말 것을 권장한다. 만약, 지연요소를 사용하여야 한다면, 최선 및 최악의 경우에 대한 타이밍이 고려되어야 한다.

로직 설계에서 하나의 팬-인 및 팬-아웃을 갖는 노드가 두 개 또는 그 이상이 직렬로 연결되어 있는 지연-체인은 입력에서 출력으로의 신호 전달을 시간적으로 지연하기 위해서 사용된다. 일반적으로 지연-체인은 비동기 설계에서 조합회로의 경쟁 문제를 해결하기 위하여 인버터를 이용하여 설계된다. 그러나 이러한 시간지연은 FPGA에 하드웨어 회로를 구현하기 위해 로직을 배치 및 배선(place and route)할 때 마다 달라진다. 즉, 같은 로직의 설계라도 배치 및 배선이 달라지면 시간지연이 달라진다. 또한 시간지연은 온도 및 습도 등의 사용 환경에 따라 달라질 수 있다. 따라서 이러한 시간지연을 위한 설계는 권장하지 않는다.

- 합성 및 타이밍 검증을 쉽게 할 수 있도록, 로직의 전체 설계에서 단일 클록에지(상승에지, 또는 하강에지)를 사용하는 플립플롭을 이용해야 한다. 또한, 하강에지보다는 상승에지에서 동작하는 플립플롭을 사용할 것을 권장한다. 상승에지 클록에 의해 구동되는 플립플롭과 하강에지 클록에 의해 구동되는 플립플롭을 모두 설계에 사용해야 한다면, 최악의 경우의 duty cycle 에 대한 타이밍 분석과 합성을 위한 모델링이 되어 있어야 하고, 이때의 duty cycle 값이 서술되어 있어야 한다. 또한, 각 설계를 다른 블록으로 분리할 것을 권장한다.

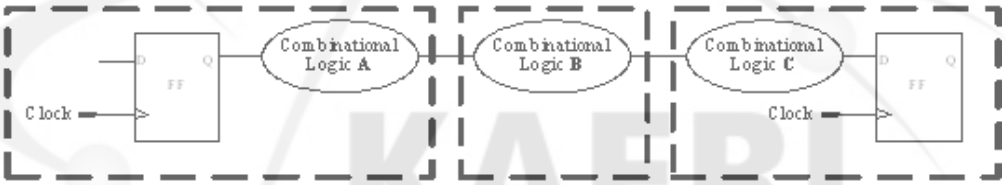
- 칩 안에 생성되는 모든 클록신호는 하나의 전용 블록에서 생성되어야 하며, 비동기 리셋신호 또한, 하나의 전용 블록에서 생성되어야 한다.

- 로직의 내부에서 리셋 신호를 생성하지 말아야 한다. 꼭 필요한 경우, 별도의 리셋 신호를 만들어 블록 외부에서 공급해야 한다.

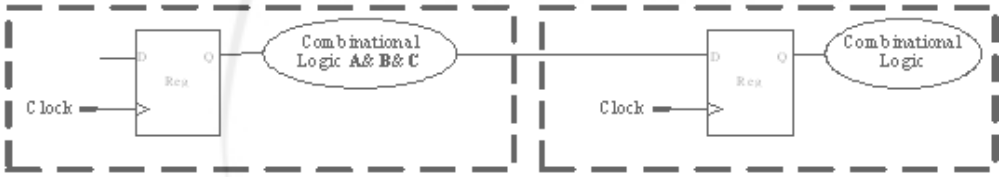
```
-- in a separate reset module
    z_rst <= rst or (a and b) ;

-- in the main module
    EX_PROC: PROCESS (clk, z_rst)
    BEGIN
        IF (z_rst = '1') THEN
            reg_sigs <= '0' ;
        ELSEIF (clk'event and clk='1') THEN
            ……
        ENDIF
    END PROCESS EX_PROC ;
```

■ 관련된 조합회로는 같은 모듈에 함께 있도록 설계할 것을 권장한다.

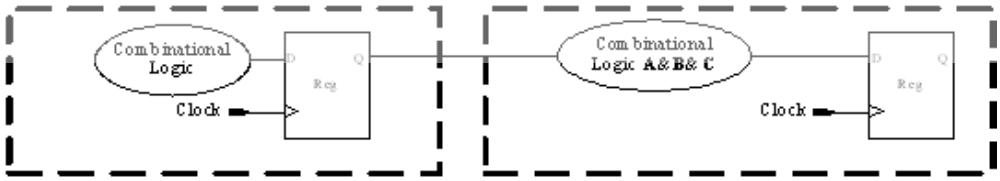
그림 5.12는 회로 설계에 대한 나쁜 예, 중간 예, 좋은 예를 보여준다. 조합회로의 출력은 레지스터를 이용하여 동기화하여 다음 블록으로 신호를 전달하는 것이 바람직하다.



(a) 나쁜 예



(b) 중간 예



(c) 좋은 예

그림 5.12 조합회로 설계 예

■ 합성 후 래치가 추출되지 않도록 해야 하며, 이를 위해 다음과 같은 코딩 방법을

따른다.

- 프로세스 문의 시작부분에 DEFAULT 값을 지정한다.
- 모든 입력 값의 경우에 대해 출력 값을 지정한다. (특히 CASE, IF 문 등)
- 우선 순위를 부여하는 조건문의 경우 마지막에 ELSEIF 대신 ELSE를 사용한다.

■ 최상위 엔티티의 포트의 방향 선언에서는 IN, OUT 및 INOUT 모드만을 사용해야 하며, 하위 블록들의 포트의 방향 선언에서는 IN 및 OUT 모드만을 사용해야 한다. 즉, 코드 내에서 출력 값을 읽기 위하여 버퍼 타입의 포트를 사용하지 말아야 한다.

■ 최상위 엔티티의 포트 선언, 하위 블록의 포트 선언 및 신호의 선언에서 STD_LOGIC 또는 STD_LOGIC_VECTOR 만을 사용해야 한다. 또한, STD_LOGIC 신호는 ‘0’, ‘1’, ‘Z' 및 'X' 값을 갖는데, 코드 내에 신호가 'X' 값을 갖지 않도록 해야 한다.

■ 선언부에 신호들에 대한 초기 값을 할당하지 말아야 한다. 신호에 대한 초기화는 리셋신호에 의하여 이루어지도록 해야 한다.

```
SIGNAL main_state : STD_LOGIC_VECTOR (7 DOWNTO 0) := "11110000" ;
```

■ 큰 우선순위(priority) 로직을 생성하는 것을 방지하기 위하여, 긴 IF-THEN-ELSE 문을 사용하는 것을 피하고, CASE 문을 사용할 것을 권장한다. 우선순위 로직을 생성하는 경우 nesting value는 적을수록 좋으며 10 이상 되지 않도록 권장한다.

■ IF-THEN-ELSE 문처럼, LOOP 문도 주의 깊게 설계되지 않으면 우선순위 encode 로직을 생성할 수 있으므로, 설계자가 로직을 설계하기 위해 LOOP 문을 사용한다면, 주위 깊게 설계를 해야 한다. 가능하다면, LOOP 문 보다는 같은 기능을 할 수 있는 다른 설계 방법을 사용하여 코드를 작성하는 것이 좋다.

■ 합성 시, 로직의 크기와 시간을 고려하여, CASE 문은 input과 output(input : 16, output : 8)을 포함하여 24 bit를 넘지 않도록 하며, CASE의 item은 100개를 넘지 않도록 권장한다. 또한, 의도되지 않은 래치를 생성하는 것을 방지하기 위하여, CASE 문에서 default 값으로 don't care 값을 지정해서는 안 된다.

■ 유한 상태 머신(Finite State Machine)을 이용한 동기회로를 설계할 경우, FSM 로직은

non-FSM 로직과 서로 다른 프로세스로 구분해서 설계해야 한다.

■ FSM의 HDL 코드는 조합로직과 순차로직을 분리하여 두 개의 다른 프로세스로 기술해야 한다.

제 6 절 유지보수 용이성

수 백 라인에서 수 만 라인에 이르는 코드를 유지보수하고 재사용하기 위해서, 설계자 및 설계조직은 통일되고 일관된 형식을 코드를 작성하기 위한 코드 작성지침이 필요하다. 이 절에서는 이러한 유지보수 편이성 측면에서 코드 작성지침을 기술한다.

1. 파일 체계 및 계층 설계

일반적으로 매우 큰 개발사업의 경우에는 수 만개의 RTL 파일을 생성되며, 유지보수의 용이성 및 재사용성을 높이기 위해서 각 개발 사업은 이러한 파일들을 적절히 관리할 수 있는 체계가 형성되어야 한다. 아래의 지침들은 개발사업의 관점에서 생성된 RTL 파일들의 적절한 관리체계를 확보하기 위한 지침들이다.

■ 전체 설계 파일들의 체계는 가장 상위수준의 기능 블록의 구조를 반영할 것을 권장한다.

예를 들어, 설계에서 가장 상위수준의 블록들이 M1, M2, M3, 및 M4이라면, 코드의 메인 디렉토리 아래에 각각의 블록에 대한 파일이 존재하는 하위 디렉토리 M1, M2, M3 및 M4가 있도록 전체 코드의 디렉토리를 구성 한다.

■ 최상위 수준에서는 어떠한 로직도 포함하지 말고 단지 각 블록의 인스턴스 생성 및 각 블록의 상호 연결정보만을 포함해야 한다. 모든 로직은 하위 블록에서 작성되어야 한다.

로직 설계자는 구현하고자 하는 기능을 설계하기 위하여 최상위 블록을 정의한다. 최상위 블록에는 로직이 구현되지 않는다. 즉, 기능을 구현하기 위한 로직은 최상위 블록을 구성하는 하위 블록에서 작성된다. 따라서 설계자는 최상위 블록에는 각 하위 블록의 상호 연결 정보만을 포함하여, 전체 하위 블록들의 구성 정보만을 기술한다.

■ 로직의 설계의 계층과 구현하고자 하는 시스템의 물리적 설계 계층과 일대일의 대응관계를 갖도록 설계할 것을 권장한다.

예를 들어, CPU를 설계하고자 할 때, 아래의 그림과 같이 로직의 설계 계층과 물리적 설계 계층이 일치하도록 하면, 큰 개발 사업의 경우에 다수의 설계자 간의 업무 할당이 용이할 뿐만 아니라, 확인 및 검증이 용이해진다.

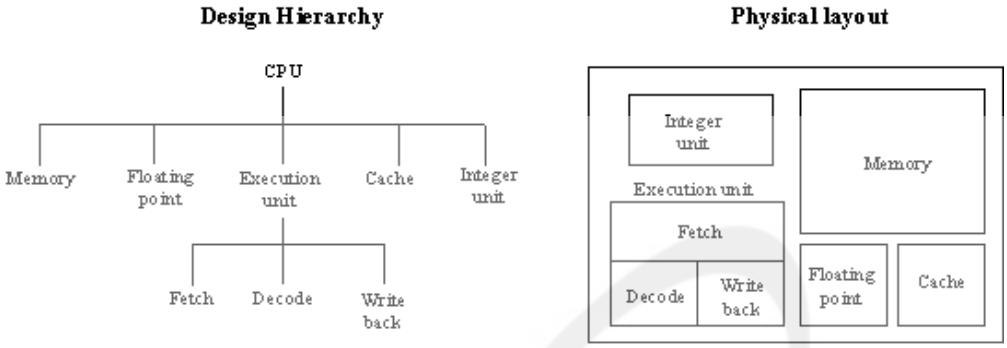


그림 5.13 코드 계층구조 및 하드웨어 계층구조

2. 코드 형식

1) 파일 이름

■ 하나의 파일에는 하나의 엔티티(entity) 및 하나의 아키텍처(architecture)만을 포함해야 한다. 또한 파일의 이름과 파일이 포함하고 있는 엔티티는 동일한 이름으로 명명하는 것을 권장한다.

```
source file 이름 : arm_test.vhd

ENTITY arm_test IS
    ……
END arm_test ;
```

■ 엔티티-아키텍처 조합구성(Configuration)은 합성을 목적으로 하는 코드에서는 사용되지 않는다. 그러나 일부 시뮬레이터는 조합구성을 필요로 할 수도 있으므로, 필요한 경우 조합구성은 엔티티 및 아키텍처가 있는 파일에 함께 포함하도록 해야 한다.

```
ENTITY adder IS
    ……
```

```
END adder ;

ARCHITECTURE adder_1 of adder IS
BEGIN
    ……
END adder_1 ;

CONFIGURATION adder_compare of adder IS
    ……
END adder_compare ;
```

2) 파일헤더 (File Header)

■ 특정 개발 사업에서 개발되는 모든 HDL 소스파일의 헤더는 같은 형식으로 구성되어야 하며, 파일의 설계자 및 해당 파일에서 설계된 내용(파일이름, 작성날짜, 버전 이력 등)이 포함되어야 하며 최소한 아래의 항목이 포함되어 있어야 한다.

(1) 사업명: 소스코드가 개발된 사업명

(2) 파일이름: 소스코드가 저장되어 있는 파일명

(3) 설계물 이름: 설계자가 본 소스코드에서 작성한 설계물 이름

(4) 설계물 설명: 소스코드에서 설계된 설계물의 기능과 동작 특성을 상세히 기록하여 설계물의 기능과 특성을 파악할 수 있도록 한다.

(5) 작성자: 소스코드를 작성한 설계자 이름. 공동 작업의 경우에는 설계자의 이름을 모두 기입하고 대표자가 맨 앞에 오도록 한다.

(6) 조직: 개발자의 소속 기관이나 회사명/부서

(7) 수정 날짜: 소스코드가 가장 최근에 수정된 날짜

(8) 코드 버전: 코드 버전

(9) 개정 이력: 개정 이력을 최초의 작성날짜 및 버전부터 시작하여 가장 최근의 작성날짜 및 버전까지 개정이력을 기술해야 하며, 각 버전 별로 수정 내용을 기술한다.

(10) 사용 시스템: 소스코드가 작성된 시스템 사양

(11) 시뮬레이션 도구: 시뮬레이션을 위해 사용된 도구 이름. 하나 이상의 시뮬레이션 도구를 사용한 경우는 모두 기록한다. 기능 시뮬레이션(function simulation)과 타이밍 시뮬레이션(timing simulation)에 사용된 도구가 다른 경우에는 이를 명시한다.

(12) 합성 도구: 합성을 위해 사용한 도구 이름. 합성에 성공한 도구를 모두 명기한다. 단 합성을 하지 않았거나 필요 없는 경우는 N/A로 표시하고 합성이 안 되는 경우는 N/S(Not Synthesizable)로 표기하고 합성에 실패한 도구 이름을 괄호에 기록한다.

3) 가독성을 위한 코딩

가. 간격 및 배치

■ 소스코드는 들여쓰기를 사용하여 읽기 쉽게 작성해야 한다. 들여쓰기는 1~4개의 space를 사용하되 그 수가 일정해야 한다.

```
data_flipflop: PROCESS (clk)
BEGIN
    IF (clk'EVENT AND clk = '1') THEN
        q <= d ;
    END IF ;
END PROCESS data_flipflop ;
```

■ 코드의 한 라인에는 글자 수가 78자를 넘지 않도록 한다.

■ 확실히 가독성이 증가할 경우에 한해서만 alias를 사용해야 한다.

■ 조건문에 대한 표현은 괄호 사용을 권장하며, 괄호 안에는 스페이스가 없고 괄호 밖에는 스페이스가 있도록 작성하는 것을 권장한다.

```
IF (rst_n = '0') THEN
```

■ 쉼표 전에는 스페이스가 없고 쉼표 후에는 스페이스가 있도록 작성하는 것을 권장한다.

```
converter : PROCESS (rst_n, clk)
```

■ "=>", "<=", ":=", ">", "<", "=", "/=", "+", "-", "&", "AND", "OR", "XOR" 같은 연산자의 양쪽에는 스페이스가 있도록 작성하는 것을 권장한다.

```
data_output <= ((('0' & a) + ('0' & b)) AND c) ;
```

■ 포트 선언은 완전히 한 라인에 기술되어야 하며, 여러 포트를 한 라인에 선언하는 것을 권장하지 않는다. 또한 포트 선언에서 사용되는 콜론은 정렬될 것을 권장한다.

```
ENTITY example_port IS
    PORT (
```

```
        rst_n       : IN STD_LOGIC
        clk     : IN STD_LOGIC
        digital_in  : IN STD_LOGIC_VECTOR (5 DOWNTO 0) ;
        digital_out     : OUT STD_LOGIC (3 DOWNTO 0)
        ) ;
END example_port;
```

■ 엔티티 포트의 신호선언 순서는 다음을 따를 것을 권장한다.

INPUT
- Resets
- Clocks
- Enables
- Other control signals
- Data and address lines

OUTPUTS
- Resets
- Clocks
- Enables
- Other control signals
- Data

아래의 예는 엔티티 포트의 신호선언에 대한 예를 보여준다.

```
ENTITY signal_multiply IS
    PORT (
        clk     : IN STD_LOGIC ;
        rst     : IN STD_LOGIC ;
        multiply_en : IN STD_LOGIC ;
        a       : IN STD_LOGIC ;
        b       : IN STD_LOGIC ;
        product     : OUT STD_LOGIC ;
        valid       : OUT STD_LOGIC
    ) ;
END signal_multiply ;
```

■ 신호 및 변수 선언은 한 라인에 하나의 신호(또는 변수)를 선언할 것을 권장한다.

```
SIGNAL clk      : STD_LOGIC ;
SIGNAL rst_n    : STD_LOGIC ;
SIGNAL add_en   : STD_LOGIC ;
…
VARIABLE count: BIT ;
```

■ 각 그룹의 신호 선언에서 사용되는 콜론은 정렬될 것을 권장한다.

```
-- control signals
```

```
SIGNAL select_0      : STD_LOGIC_VECTOR (31 DOWNTO 0) ;
SIGNAL select_1      : STD_LOGIC_VECTOR (31 DOWNTO 0) ;
SIGNAL state     : STD_LOGIC_VECTOR (31 DOWNTO 0) ;

-- address and data signals
SIGNAL read_address   : STD_LOGIC_VECTOR (31 DOWNTO 0) ;
SIGNAL write_address  : STD_LOGIC_VECTOR (31 DOWNTO 0) ;
SIGNAL write_data  : STD_LOGIC_VECTOR (31 DOWNTO 0) ;
```

■ 포트 맵 선언에서 “=>” 는 정렬될 것을 권장한다.

```
i_pokerhand : pokerhand
   PORT MAP (
      rst_n       => rst_n,
      clk     => clk,
      card_0_in => card_0_i,
      card_1_in => card_1_i,
      card_2_in => card_2_i,
      card_3_in => card_3_i,
      card_4_in => card_4_i,
      hand_out  => hand_i
      ) ;
```

■ 코드의 각 문장(statement)은 한 라인에는 한 개의 문장만을 사용할 것을 권장한다.

```
WHEN 00     => outc <= a ;
WHEN 01     => outc <= b ;
WHEN 10         => outc <= c ;
WHEN others     => outc <= d ;
```

■ 2개 이상의 연산의 경우, 우선순위가 명백하더라도 코드를 읽기 쉽게 괄호를 사용할 것을 권장한다.

```
example : FOR i IN 1 TO 3 LOOP
   cout        := (a(i) and b(i)) or (a(i) and cin) or (a(i) and cin) ;
   sum(i) := a(i) xor b(i) xor cin ;
END LOOP example ;
```

나. 설명문 (Comments)

■ 본인 및 다른 어떠한 설계자가 후에 해당 코드를 쉽게 해석할 수 있도록 설계자가 필요하다고 판단되는 모든 코드에 설명문을 추가해야 한다.

■ 프로세스, 함수, 및 프로시저의 앞에 각 블록의 목적을 기술하는 설명문을 추가할 것을 권장한다.

■ 프로세스, 함수, 및 프로시저 내의 신호, 변수 및 문장에 대하여 설명문을 삽입할 것을 권장한다. 이러한 설명문은 논리(syntax) 또는 의미(semantics)를 단순하게 설명하는 것이 아니라, 신호, 변수 및 문장의 의도된 기능에 대해 기술해야 한다.

다. 이름 명명

■ 엔티티, 신호 및 프로세스의 이름은 이름만으로 해당 엔티티, 신호 및 프로세스의 기능을 파악할 수 있도록 이름을 명명할 것을 권장한다.

■ 엔티티, 신호 및 프로세스의 이름은 16자를 초과하지 않도록 하며, 이름은 A~Z, 0~9 및 '_'만을 사용하여야 한다. 또한, '_'를 연속적으로 사용하거나 이름이 '_'로 끝나지 않도록 명명할 것을 권장한다.

■ 아래와 같이 설계자가 명명하는 모든 이름은 소문자를 사용할 것을 권장한다.
- 엔티티, 아키텍처, 엔티티-아키텍처 구성 및 팩키지
- 신호, 변수 및 상수
- 프로세저 및 함수
- 프로세스, instantiation 및 generation lables

```
SIGNAL   a   : IN STD_LOGIC ;
SIGNAL   b   : IN STD_LOGIC_VECTOR (7 DOWNTO 0) ;
VARIABLE   temp   : OUT INTEGER RANGE 0 TO 9 ;
```

■ 아래와 같은 설계자가 명명하는 것이 아니라, HDL에서 제공하는 표현들은 대문자를 사용할 것을 권장한다.
- HDL 예약어
- 라이브러리

■ VHDL이나 Verilog는 서로 변환이 가능해야 하므로, HDL(VHDL 및 Verilog)의 예약어를 이름에 사용하지 않아야 한다.

■ 아키텍처의 이름은 아래의 set중에 하나를 선택하여 사용할 것을 권장한다.

behavioral	: 물리적인 로직이지만 RTL 도구로 번역되지 않는 로직
rtl	: 물리적인 로직이고 RTL 도구로 변역되는 로직

structural : 로직이 아니고, 물리적 연결을 수행함

gate : 게이트 수준의 netlist

simulation : 시뮬레이션 모델

testbench : 테스트 벤치

ARCHITECTURE behavioral OF example_model IS
ARCHITECTURE rtl OF example_model IS
ARCHITECTURE structural OF example_model IS
ARCHITECTURE gate OF example_model IS
ARCHITECTURE simulation OF example_model IS
ARCHITECTURE testbench OF example_model IS

■ 가능한 신호의 이름을 명명할 때는 각 개발 사업에서 규정한 규약을 따를 것을 권장한다. 아래의 예는 일반적은 신호 명명의 예를 보여준다.

일반적인 레지스터 출력 : signalname_r
지연 레지스터 출력 : signalname_d
지연 체인 : signalname_d0, signalname_d1, ...
동기화 체인 : signalname_s0, signalname_s1, ...
토글 출력 : signalname_t
falling edge 레지스터 출력 : signalname_f
Level based control, enable : signalname_e
Triggering control, plus : signalname_p
active low signal : signalname_n
Internal signal : signalname_i
Combinatory signal : signalname_c
Variable : signalname_v
Asynchronous signal : signalname_a

■ 클록 신호의 이름은 "clk"를 사용해야 한다. 만약, 여러 개의 클록 신호가 사용된다면, 모든 클록 신호의 이름에는 "clk"를 접미어로 사용해야 한다.

■ 특정신호가 다른 계층 간에 사용될 때는 해당 신호의 이름은 같아야 한다. 특히, 클록 신호는 같은 이름을 사용해야 한다.

■ 리셋 신호의 이름은 "rst"를 사용해야 한다. 만약, 여러 개의 리셋 신호가 사용된다면, 모든 리셋 신호의 이름에는 "rst"를 접미어로 사용해야 한다.

■ 포트 이름에는 신호의 방향성(in 또는 out)을 나타낼 수 있는 접미어를 사용할 것을 권장한다.

■ 코드의 명확성을 위해서 PORT와 GENERIC에 대한 신호 연결은 위치연결(positional mapping) 방법보다 이름연결(named mapping) 방법을 사용해서 신호를 연결해야 한다.

```
GENERIC MAP (width => word_length)

PORT MAP (
    a   => in1,
    b   => in2,
    ci  => carry_in,
    sum   => sum,
    co  => carry_out
    )
```

■ 다중 비트 신호 또는 다중 비트 포트는 (x TO y) 보다는 (y DOWNTO x)를 사용할 것을 권장한다.

```
data_in : IN STD_LOGIC_VECTOR (data_width-1 DOWNTO 0) ;
…
```

■ 컴포넌트 실체(component instance)의 이름은 컴포넌트의 이름에 컴포넌트의 실체 번호를 나타내는 접두어 “u#_"를 붙여서 명명할 것을 권장한다.

```
u1_logic_or   : logic_or PORT MAP (....)
u2_logic_or   : logic_or PORT MAP (....)
u3_logic_or   : logic_or PORT MAP (....)
```

■ 모든 프로세스 블록에는 이름(label)을 명명해야 하며, 각 프로세스 블록의 이름은 <name>_proc로 할 것을 권장한다.

```
reg_proc : PROCESS (clk, rst)
        BEGIN
            ...
        END PROCESS reg_proc ;
```

■ 합성 후, post-simulation 실행 시 충돌을 막기 위해, ASIC libray에 존재하는 같은

instance cell 이름은 사용하지 않아야 한다.

4) 이식성을 위한 코딩

■ 벡터의 크기는 고정된 숫자를 사용하기 보다는 매개변수(parameter)를 사용하는 것을 권장한다.

```
-- bad example
data_in : IN STD_LOGIC_VECTOR (7 DOWNTO 0) ;

-- good example
data_in : IN STD_LOGIC_VECTOR (datawidth-1 DOWNTO 0) ;
```

■ 기술 독립적인 라이브러리를 사용해야 한다.

■ 같은 로직 설계에 active low 로직과 active high 로직을 섞어 쓰지 않아야 하며, active high 로직을 권장한다.

■ 일반적으로 내부 로직에 삼상태 신호의 사용을 피해야 한다. 하지만 내부 모니터를 위한 삼상태 신호는 허용한다.

■ 대부분의 합성도구는 특별한 경우를 제외하고는 나누기 연산을 합성하지 못한다. 따라서 나누기 연산을 사용하지 않아야 한다.

■ 16 bit 이상의 곱셈 연산을 효율적인 합성 결과가 나오지 않으므로 직접 연산을 통한 합성보다는 라이브러리를 이용하거나 별도의 엔티티를 통해 논리 연산으로 구현하는 것이 좋다.

■ IEEE standard type과 이를 기반으로 한 type과 subtype만 사용해야 한다. bit 또는 bit_vector는 사용하지 않는다.

■ 모든 매개변수(parameter) 값과 함수(function)에 대한 정의는 분리된 파일로 만들고 이름은 <Design Name>_package.vhd 할 것을 권장한다.

■ GENERATE 문과 BLOCK 문은 사용을 자제한다.

■ CONSTANT 선언된 것을 변경하는 코드를 사용하지 않는다.

제 6 장 결론 및 활용방안

원자력 산업의 경우, 기존의 원자력발전소에 사용되고 있는 아날로그기술 기반의 계측제어 시스템은 노후화로 인해 보수 및 교체가 요구되고 있다. 그러나 기술지원 및 부품조달의 어려움으로 인해 아날로그 기기의 계속 사용에 어려움이 있으며, 기존 원자력발전소의 유지 및 보수에 어려움을 겪고 있다. 이러한 이유로 인해, 국제적으로 기존의 아날로그기술 기반의 계측제어 시스템을 디지털기술 기반의 계측제어 시스템으로 교체하고 있으며, 신규 원자력발전소에도 디지털기술 기반의 계측제어 시스템을 채택하고 있다.

현재 디지털기술 기반의 계측제어 시스템은 마이크로프로세서 기반의 하드웨어에 소프트웨어를 탑재하는 방식과 기존의 아날로그 기기의 설계를 FPGA(Field Programmable Gate Array) 기반의 설계로 대체하는 방식으로 개발되고 있다.

소프트웨어 기반의 디지털 계측제어계통은 소프트웨어로 구현되므로 계통의 기능을 구현하는데 매우 유연성이 좋고, 데이터의 전송과 처리, 저장능력 및 정확도에서 아날로그기반 기술에 비해 매우 우수한다. 또한, 하드웨어의 드리프트가 거의 없어서 교정에 드는 비용이 거의 없다. 그러나, 계통에 탑재되는 소프트웨어의 신뢰도 및 안전성을 정량적으로 평가하기 어렵다. 이러한 이유로 인해, 소프트웨어의 신뢰도 및 안전성을 확보하기 위하여 소프트웨어의 모든 개발단계(계획, 요구사항, 설계, 구현)마다 소프트웨어에 오류가 삽입되는 것을 방지하기 위하여 소프트웨어의 확인 및 검증을 수행하며, 확인 및 검증을 수행하는데 비용이 많이 든다. 또한, 소프트웨어의 고장으로 인한 시스템의 영향을 분석하기 어렵고, 사이버 보안에 대한 문제가 존재하고 있다. 이외에도 소프트웨어 기반의 디지털 시스템은 기존은 아날로그 기반 시스템의 유지 및 보수의 어려움으로 인해 도입되고 있지만, 마이크로프로세서 및 메모리 등의 부품의 생산중단으로 인해 유지 및 보수의 문제가 여전히 존재하고 있다.

상대적으로 FPGA 기반의 디지털 계측제어 시스템은 소프트웨어 기반의 디지털 계측제어 시스템 보다는 개발이 용이하고, 응답시간이 빠르며, 결정론적 특성의 구현이 용이하고, 기기 및 부품 단종에 유연하게 대처할 수 있다. 또한, FPGA 로 구현되므로 소프트웨어로 인한 신뢰도, 안전성, 확인 및 검증 비용이 들지 않으며, 사이버 보안에 대한 문제도 거의 존재하지 않는다. 이러한 이유로 인해, 국내외적으로 FPGA 기반 디지털 계측제어계통의 개발에 관심이 높아지고 있다.

본 과제에서는 프로그래머블 논리 소자를 이용한 원자력발전소의 계측제어시스템의 개발 가능성을 파악하기 위하여 FPGA의 구조 및 취약성을 조사하였고, 국내외의 FPGA를 이

용한 계측제어 시스템의 개발 사례를 조사하였다. 또한, FPGA를 설계하는 데 필요한 설계 지침 및 코딩 지침을 제시하였다.

본 프로그래머블 논리 소자의 기술현화 분석 결과, 충분히 프로그래머블 논리 소자를 원자력발전소의 안전필수 계측제어계통 개발에 도입할 수 있는 것으로 판단된다. 프로그래머블 논리 소자의 적용 효과는 다음과 같다.

- 소프트웨어 기반의 디지털 시스템에 비해 구조가 간단해 져, 개발 비용이 감소한다.
- 소프트웨어 기반의 시스템은 한 플랫폼에서 다중의 안전기능을 수행하지만 프로그래머블 논리 소자 기반의 시스템은 다중의 기능을 각각의 프로그래머블 소자로 구현되므로 공통원인 고장에 영향이 감소한다.
- 소프트웨어를 사용하지 않으므로 소프트웨어 관련 문제(사이버 보안, 소프트웨어 공통원인 고장, 확인 및 검증 비용 등)를 해결할 수 있다.
- 소프트웨어 기반 시스템보다 월등한 성능을 확보할 수 있다.

본 보고서에 포함된 설계지침 및 코딩 지침은 추후 프로그램머블 논리 소자를 이용한 디지털 시스템의 개발에 활용될 수 있을 것으로 판단된다.

제 7 장 참고문헌

[1] 김주민, “쉽게 배우는 VHDL 이론 및 실습,”, 그린출판, 2003

[2] Charles H. Roth, “Digital System Design Using VHDL,” Cengage Learning, 2008

[3] Bruce Wile, “Comprensive Functional Verification”, Eelsevier, 2005

[4] Vyacheslav Kharchenko, “Experience of RPC <<Radiy>> is designing, manufacturing and implementation of FPGA-based NPP I & C systems”, First Workshop on The Applications of Field-Programmable Gate Arrays in Nuclear Power Plants, October, Chatou, France, 2008

[5] Patrizio Piasentin, “FPGA … a solution for High-Rel and long life time application,” First Workshop on The Applications of Field-Programmable Gate Arrays in Nuclear Power Plants, October, Chatou, France, 2008

[6] Claude Esmenjaud, “Drivers and barriers for use of FPGA technology in safety I&C systems in NPPs, First Workshop on The Applications of Field-Programmable Gate Arrays in Nuclear Power Plants, October, Chatou, France, 2008

[7] Pascal Pampagnin, “DO254 ED 80 Design Assurance Guidance for Airborne Electronic Hardware,” First Workshop on The Applications of Field-Programmable Gate Arrays in Nuclear Power Plants, October, Chatou, France, 2008

[8] David Dangla, “FPGA for space applications,” First Workshop on The Applications of Field-Programmable Gate Arrays in Nuclear Power Plants, October, Chatou, France, 2008

[9] Jean Gassino, “Objectives of the new Standard IEC 62566 Draft status,” First Workshop on The Applications of Field-Programmable Gate Arrays in Nuclear Power Plants, October, Chatou, France, 2008

[10] Jung S. KOH, “Licensing Experience for FPGA/CPLD in Digital-Based Safety Systems in Korea,” First Workshop on The Applications of Field-Programmable Gate Arrays in Nuclear Power Plants, October, Chatou, France, 2008

[11] Toshifumi Hayashi, “Qualification and Application of FPGA-Based Safety-Related I&C Systems”, First Workshop on The Applications of Field-Programmable Gate Arrays in Nuclear Power Plants, October, Chatou, France, 2008

[12] Miljko Bobrek1, “Safe FPGA Design Practices for Instrumentation and Control in Nuclear Plants”, First Workshop on The Applications of Field-Programmable Gate Arrays in Nuclear Power Plants, October, Chatou, France, 2008

[13] Mikhail Yastrebenetsky, “Safety Assessment Of FPGA-based ESFAS For Kozloduy NPP”, 6th American Nuclear Society International Topical Meeting on Nuclear Plant Instrumentation, Control,

[14] and Human-Machine Interface Technologies, April 5-9, 2009

[15] Miljko Bobrek, “FPGA Design Practices For I&C In Nuclear Power Plants”,

[16] Patrick Salaün, “FPGA/ASIC: A Promising Technology For Future Of I&C Systems In Power Industry”,

[17] Jingke She, “Application Of FPGA To Shutdown System No.1 In CANDU”

[18] Rossnyev Alvarado, “Approach To Designing FPGA-Based Digital I&C Systems For Nuclear Applications”

[19] Ievgenii Bakhmach, “Implementation Principles Of FPGA-based ESFAS For Kozloduy NPP,”

[20]

[21] Design Guidelines and Criteria for Space Flight Digital Electronics, NASA

[22] Ian Grout, "Digital Systems Design with FPGAs and CPLDs," Elsevier Ltd, 2008.

[23] William K. Lam, "Hardware Design Verification: Simulation and Formal Method-Based Approaches," Prentice Hall, 2005.

[24] Janick Bergeron, "Writing Testbenches: Functional Verification of HDL Models," Springer, 2003.

[25] Ben Cohen, VHDL: Coding Styles and Methodologies, 2nd Edition, Kluwer Academic Publishers, 1999

[26] Volnei A. Pedroni, "Circuit Design with VHDL," MIT Press, 2004.

[27] Charles H. Roth, “Digital Systems Design Using VHDL," Cengage Learning, 2008

[28] Samir Palnitkar, "Verilog HDL: A Guide to Digital Design and Synthesis," 2001.

[29] Shahabuddin L. Inamdar, "VHDL Coding Style Guidelines and Synthesis: A Comparative Approach," Thesis for the Master degree, 2004.

[30] HDL Coding Guidelines: VHDL, 반도체설계자산연구센터, 2002.

서 지 정 보 양 식

수행기관보고서번호	위탁기관보고서번호	표준보고서번호	INIS 주제코드
KAERI/AR-836/2009			

제목 / 부제	원자력발전소 계측제어계통에 적용을 위한 CPLD/FPGA 기술현황분석

주저자(부서명)	최종균(계측제어.인간공학연구부)
연 구 자 및 부 서 명	권기춘, 이동영, 이장수, 이영준, 손광섭, 박기용, 이현철, 박원만, 장통일, 김동훈, 오인석, 이정운, 이용희, 허 섭, 김정택,황인구, 박재창, 이철권, 김장열, 김창회, 천세우

출 판 지	대전	발행기관	한국원자력연구원	발행년	2009
페 이 지	114 p.	도 표	있음(O), 없음( )	크 기	21 x 29.7 cm.
참고사항					
공개여부	공개(O), 비공개( )		보고서종류	기술현황분석보고서	
비밀여부	대외비( ), -__ 급비밀				
연구위탁기관			계약번호		

초록 (15-20 줄내외)

원자력 산업의 경우, 기존의 원자력발전소에 사용되고 있는 아날로그기술 기반의 계측제어 시스템은 노후화로 인해 보수 및 교체가 요구되고 있다. 그러나 기술지원 및 부품조달의 어려움으로 인해 아날로그 기기의 계속 사용에 어려움이 있으며, 기존 원자력발전소의 유지 및 보수에 어려움을 겪고 있다. 이러한 이유로 인해, 국제적으로 기존의 아날로그기술 기반의 계측제어 시스템을 디지털기술 기반의 계측제어 시스템으로 교체하고 있으며, 신규 원자력발전소에도 디지털기술 기반의 계측제어 시스템을 채택하고 있다. 현재 디지털기술 기반의 계측제어 시스템은 마이크로프로세서 기반의 하드웨어에 소프트웨어를 탑재하는 방식과 기존의 아날로그 기기의 설계를 FPGA(Field Programmable Gate Array) 기반의 설계로 대체하는 방식으로 개발되고 있다.

본 보고서는 디지털 계측제어시스템 개발을 위한 프로그래머블 논리 소자의 적용성을 분석하기 위하여 프로그래머블 논리 소자의 종류, 구조, 제조 방법 및 특성을 정리하였고, 국내외의 프로그래머블 논리소자 기반의 계측제어시스템의 개발동향을 정리하였다. 프로그래머블 논리 소자의 설계에 활용 가능한 설계지침 및 코딩 지침을 정리하였다.

주제명키워드 (10 단어내외)	프로그래머블 논리 소자, 계측제어계통, 확인, 검증

BIBLIOGRAPHIC INFORMATION SHEET

Performing Org. Report No.	Sponsoring Org. Report No.	Standard Report No.	INIS Subject Code
KAERI/AR-836/2009			

Title / Subtitle	Survey of the CPLD/FPGA Technology for Application to NPP Digital I&C System
Main Author	J. G. Choi (I&C/Human Factors)
Researcher and Department	K. C. Kwon, D. Y. Lee, J. S. Lee, Y. J. Lee, K. S, Son, G. Y. Park, H. C, Lee, W. M. Park, T. I. Jang, D. H. Kim, I. S. Oh, J. W. Lee, Y. H. Lee, S. Hur, J. T. Kim, I. K. Hwang, J. C. Park, C. K. Lee, J. Y. Kim, C. H. Kim, S. W. Cheon

Publication Place	Daejon	Publisher	KAERI	Publication Date	2009.08
Page	114 p.	Ill. & Tab.	Yes(O), No ( )	Size	21 x 29.7 Cm.
Note					

Open	Open(O), Closed( )	Report Type	
Classified	Restricted( ), -___Class Document		
Sponsoring Org.		Contract No.	

Abstract (15-20 Lines)

The majority of the I&C systems in today's NPPS are widely based on analog technology. Most of the existing I&C systems are facing obsolescence problem. The existing NPPs have difficulty in repairment and replacement for maintenance because manufacturers do not produce the analog devices and boards used in the I&C systems any more. Therefore, the existing NPPs are replacing the obsolete analog I&C systems with modern advanced digital systems. New NPPS are also adopting digital I&C systems because the economical efficiency and the usability of the systems become higher than the analog I&C systems.

The digital I&C systems are based on two digital technologies. One is the micro-processor based digital system. The other is digital system based on programmable logic devices such as FPGA(Field Programmable Gate Array)

In this research, we have collected and reviewed the types, structure, configuration technologies, vendors, and vulnerabilities of programmable logic devices in order to analyze the feasibility of adoption of safety-critical I&C systems based on programmable logic devices. In addition to this, we developed practical design guidelines and coding guidelines for programmable logic devices

Subject Keywords (About 10 words)	Programmable Logic Device, CPLD, FPGA, Instrumentation and Control System, Verification, Validation

KAERI