MI-01-38-00-0013-01-D05-00-011-1-0

우주기술개발사업

FPGA를 이용한 우주용 On-Board Computer의
소형화/경량화 기술 개발
Development of Small Size/Light Weght On-Board
Computer for Space Applications using FPGAs

개발주관연구기관
한국과학기술원
인공위성연구센터

과 학 기 술 부

제 출 문

과학기술부 장관 귀하

본 보고서를 "핵심우주기술개발사업" 과제 (세부과제: "FPGA를 이용한 우주용 On-Board Computer의 소형화/경량화 기술 개발") 의 보고서로 제출합니다.

2004. 5. 20.

주관연구기관명 : 한국과학기술원
인공위성연구센터

주관연구책임자 : 박 홍 영

연 구 원 : 김 병 국

〃 : 오 대 수

〃 : 정 성 인

〃 : 류 상 문

보고서 초록

<table>
<tr><td>과제관리번호</td><td>M1-01-38-00-0013-01-D05-00-011-1-0</td><td>해당단계 연구기간</td><td>3 차 년 도 (03.5.21-04.5.20)</td><td>단계 구분</td><td>(3) / (3)</td></tr>
<tr><td rowspan="2">연구사업명</td><td>중 사 업 명</td><td colspan="4">특정연구개발사업</td></tr>
<tr><td>세부사업명</td><td colspan="4">우주기술개발사업</td></tr>
<tr><td rowspan="2">연구과제명</td><td>중 과 제 명</td><td colspan="4">핵심우주기술개발사업</td></tr>
<tr><td>세부(단위)과제명</td><td colspan="4">FPGA를 이용한 우주용 On-Board Computer의 소형화/경량화 기술 개발</td></tr>
<tr><td>연구책임자</td><td>박 홍 영</td><td>해당단계 참여연구원수</td><td>총 : 5 명
내부 : 2 명
외부 : 3 명</td><td>해당단계 연구비</td><td>정부: 30,000 천원
기업: . 천원
계: 30,000 천원</td></tr>
<tr><td>연구기관명 및 소속부서명</td><td colspan="2">한국과학기술원
인공위성 연구 센터</td><td>참여기업명</td><td colspan="2"></td></tr>
<tr><td>국제공동연구</td><td colspan="5">상대국명 : 상대국연구기관명 :</td></tr>
<tr><td>위 탁 연 구</td><td colspan="5">연구기관명 : 연구책임자 :</td></tr>
<tr><td colspan="4">요약(연구결과를 중심으로 개조식 500자이내)</td><td>보고서 면수</td><td></td></tr>
</table>

본 과제의 목표는 FPGA를 이용하여 우주용 OBC(On-Board Computer) 보드를 소형화/경량화하고, 위성에 실제 적용할 수 있는 수준의 OBC를 개발하는 것이다.

세부 항목으로 CPU 32비트, 50MIPS이상, Fault Tolerant 구조를 가지며, FPGA를 통한 EDAC 회로 구현, 100K bps 이상의 통신 채널 구현, Watchdog Timer 구현, 각종 제어 신호 구현을 포함한다.

본 과제는 우주환경에 대한 조사 및 우주 환경을 위한 부품 규격 등의 기준을 기반으로, 32비트 지원 가능한 Power PC 603e CPU 기반의 보드를 개발하였다.

개발한 보드의 특징은
외부 클럭이 25MHz이며, CPU는 내부 클럭 150Mhz로 동작하여 50MIPS 이상을 지원하며,
메인 CPU와 보조 CPU를 이용한 CPU의 SEU 오류 검출 가능한 Fault Tolerant 한 구조이며,
TMR Logic이 FPGA에 구현되어 메모리를 보호하여 SEU를 극복하는 구조이며,
FPGA를 통하여 115.2Kbps의 SCC(Serial Communication Controller)를 구현하여 100K bps 이상의 통신 채널이 가능하며, 추가적으로 송수신 데이터의 패킷단위 처리와 CAN 코어 구현을 통한 통신이 가능하며, WatchDog Timer를 FPGA 내부에 두어 프로그램 오류에 대한 극복이 가능하며,
보드 체크를 위한 LED와 LCD가 있다.

색 인 어 (각 5개 이상)	한 글	인공위성, 탑재컴퓨터, 우주 방사선, 인공위성연구센터, 인쇄회로기판
	영 어	Satellite, On-Board Computer, FPGA, Cosmic Ray, PCB, SaTReC

요 약 문

Ⅰ. 제 목

FPGA를 이용한 우주용 On-Board Computer의 소형화/경량화 기술 개발

Ⅱ. 연구개발의 목적 및 필요성

1. 연구 개발의 목적

FPGA를 이용한 우주용 OBC의 소형화/경량화 기술 개발을 통하여 우주용 회로 설계 기술 확보 및 향상, 우주환경이 전자 회로에 미치는 영향에 대한 이해 증가, 우주용 FPGA 회로 설계 기술 확보, 우주용 고 신뢰도 시스템 기술 확보, 우주용 시스템의 경량화/소형화 기술 습득, 내방사선 회로 및 시스템 설계 기술 습득 등의 기술 확보를 하는 것이 하나의 목적이다. 또한 우리나라에서 아직까지 위성의 OBC(On-Board Computer, 탑재컴퓨터)로서 50 MIPS 정도의 빠른 CPU를 탑재하지 못하고 있는데, 본 과제를 통하여 좀 더 빠른 위성 OBC를 개발하는 데 기반이 되는 것이 본 과제의 목적이다.

2. 연구 개발의 필요성

FPGA를 이용한 위성 OBC 설계는 기존 많은 부품의 기능을 FPGA Code로 구현할 수 있기 때문에 부품의 단종에 따른 설계 변경효과를 최소화 할 수 있다. 또한 우주환경에서 발생할 수 있는 SEU(Single Event Upset), SEL(Single Event Latchup) 등에 내구성을 가지는 FPGA Coding 기법의 개발은 고 신뢰도 OBC 설계뿐 아니라 위성에서 사용되는 대부분의 모듈 설계에 적용할 수 있어 향후 고 신뢰도 우주용 모듈 개발에 있어서 폭넓게 사용될 수 있다. 더욱이 FPGA를 이용하면 기존의 여러 부품을 하나의 FPGA에 집적화 할 수 있어 위성의 각 모듈을 소형화/경량화 하는 데 있어서도 매우 중요하다.

III. 연구개발의 내용 및 범위

구분	연구 내용	연구개발 범위
1차년도 (2001)	FPGA를 이용한 소형/경량화된 OBC 개념 및 구조 설계	○검증을 위한 시험 보드 제작 ○OBC 개념설계:32bit CPU ○CPU Core의 VHDL Code화 검토 ○OBC 시스템 구조 및 접속 설계 ○FPGA 구현 Logic 범위 설정 및 타당성 검토 ○Fault Tolerant 구조 설계
2차년도 (2002)	FPGA를 이용한 로직 구현	○ SEU, SEL을 고려한 고신뢰도 VHDL 구현 기법 연구 ○ EDAC 회로의 VHDL 구현 ○ Communication Controller 회로의 VHDL 구현: 100KBps 이상 ○ Watchdog Timer 및 각종 제어로직의 VHDL 구현
3차년도 (2003)	OBC 시제품 제작 및 성능 분석	○시제품 제작 ○회로기판의 소형화 설계 ○신뢰도 계산 ○기능시험 및 성능 분석

IV. 연구개발결과

본 과제의 결과는 크게 두 가지가 있다. 하나는 FPGA를 이용하여 우주용 OBC를

소형화/경량화 하여 실제 위성에 적용할 수준의 OBC 보드를 개발한 것이다. 또 하나는 인공위성의 환경에 대한 연구와 SEU 영향 분석을 한 결과이다.
OBC 보드의 개발에 대한 결과는 3개의 OBC 보드 개발, PCB 설계, FPGA 로직 구현을 포함한다. 인공위성 환경에 대한 연구는 지구 주변 우주환경과 발사 환경 등에 대한 기술과 우리별 1호와 최근 개발된 과학기술위성1호에서의 SEU 발생률을 바탕으로 개발된 보드의 Wash주기에 대한 분석 한 것이다.

Ⅴ. 연구개발결과의 활용계획

본 과제를 통하여 확보된 위성용 OBC 소형화/경량화 기술은 소형화/경량화가 필수적으로 요구되는 과학위성 2호 및 추후 "국가 우주 개발 중장기 계획"에 따라 개발예정인 과학위성 시리즈의 OBC 개발에 직접적으로 활용 가능하다. 또한 OBC를 개발하면서 습득된 대부분의 기술은 위성의 타 모듈을 개발하는데 곧바로 적용할 수 있으므로 차후 개발될 과학위성 시리즈의 각 모듈의 개발에 응용할 수 있다.

뿐만 아니라, "우주 개발 중장기 계획"에 따라 앞으로 개발될 다목적 실용위성 시리즈의 각 모듈 개발에도 적극 활용 할 수 있을 것으로 예상 된다

SUMMARY

In this study, We developed Satellite On-Board Computer(OBC) which is small size and light weight by using FPGAs.

Developed OBC is fault tolerant structure system which has 32 bit CPU over 50 MIPS, and EDAC(Error Detection and Correction) logic is built in FPGA and more than 100kbps communication channel is also built in and watchdog logic is embedded.

First, We studied space environment and electronic parts standard which can use in space and we finally developed Power PC 603e based OBC board.

The feature of OBC board is as follow.
External clock is 25MHz and Internal CPU clock is 150MHz and support more than 50 MIPS.
SEU(Single Event Upset) error of CPU can be detected by having additional secondary CPU.
External memory is strongly protected by using TMR(Tripple Modular Redundancy) method in FPGA.
OBC board has Fault tolerant structrure which has dual CPU, TMR Memory and Watchdog logic.

High speed serial commnuaction logic is developed in FPGA. so more than 100kbps communication can use and also can use transmit/receive data pakcet processing.
CAN (Control Area Network) core logic is also built in FPGA and can communicate another CAN Network Unit.
OBC board has debugging parts such as LEDs and LCD with which we developed more easily.

CONTENTS

목 차

제 1 장. 연구개발과제의 개요

1 절 연구 개발의 목적

최근 고 신뢰도 고 성능 FPGA의 등장에 따라 위성 회로 설계에도 FPGA를 이용하는 방법들이 많이 연구되고 있다. 또한 FPGA의 방사능 효과에 대한 연구 결과가 최근 보고 되고 있으며, 그 결과를 토대로 위성 회로에 적용가능성 및 회로 설계방법 등에 관한 연구가 진행되고 있다. 따라서 앞으로 FPGA는 우주용 회로 설계의 상당 부분을 차지할 전망이다. 특히, 우주 방사능에 의한 효과가 적은 저궤도 위성의 경우 FPGA를 이용한 회로 설계는 앞으로 상당히 많이 이루어질 것으로 예상된다. 일부 선진국의 경우 Re-Programmable한 FPGA를 이용하여 On-board상에서 FPGA내부의 Code를 변경함으로써 시스템 구조를 바꾸는 연구까지 진행되고 있는 실정이다.

그러므로 FPGA를 이용한 우주용 OBC의 소형화/경량화 기술 개발을 통하여 우주용 회로 설계 기술 확보 및 향상, 우주환경이 전자 회로에 미치는 영향에 대한 이해 증가, 우주용 FPGA 회로 설계 기술 확보, 우주용 고 신뢰도 시스템 기술 확보, 우주용 시스템 경량화/소형화 기술 습득, 내방사선 회로 및 시스템 설계 기술 습득 등의 기술 확보를 하는 것이 하나의 목적이다. 또한 우리나라에서 아직까지 위성의 OBC로서 50 MIPS 정도의 빠른 CPU를 탑재하지 못하고 있는데, 본 과제를 통하여 좀 더 빠른 위성 OBC를 개발하는 데 기반이 되는 것이 본 과제의 목적이다.

2 절 연구 개발의 필요성

본 과제의 목표는 FPGA를 이용하여 우주용 On-board Computer(OBC)를 소형화/경량화하고, 위성에 실제 적용할 수 있는 수준으로 OBC를 개발하는 것이며, 연구의 필요성을 기술적인 측면, 경제 산업적 측면으로 나누어 기술할 수 있다.

1 기술적 측면

위성의 OBC(On-board Computer)는 고성능의 계산 능력뿐만 아니라 안정성, 신뢰성이 요구되며, 기존 부품의 단종에 따른 회로 변경의 위험성을 최소화할 필요가

있다. 특히 과학기술위성과 같이 초소형 위성에 사용되는 OBC는 자세제어뿐 아니라 전력계통의 제어에도 사용되는 경우가 많으므로 고 신뢰성을 유지하면서 OBC의 성능을 향상시키는 것이 중요하다.

FPGA를 이용한 위성 OBC 설계는 기존의 많은 부품의 기능을 FPGA Code로 구현할 수 있기 때문에 부품의 단종에 따른 설계 변경효과를 최소화 할 수 있다. 또한 우주환경에서 발생할 수 있는 SEU(Single Event Upset), SEL(Single Event Latchup)등에 내구성을 가지는 FPGA Coding 기법의 개발은 고 신뢰도 OBC 설계뿐 아니라 위성에서 사용되는 대부분의 모듈 설계에 적용할 수 있어 향후 고 신뢰도 우주용 모듈 개발에 있어서 폭넓게 사용될 수 있다. 더욱이 FPGA를 이용하면 기존의 여러 부품을 하나의 FPGA에 집적화 할 수 있어 위성의 각 모듈을 소형화/경량화 할 수 있기 때문에 본 연구는 매우 필요하다.

2 경제·산업적 측면

우주용으로 사용되는 대부분의 부품은 매우 고가이며, 또한 구하기도 어려운 경우가 많다. 따라서 우주용 OBC의 설계에 있어서 FPGA의 사용은 하나의 FPGA에 각 단위부품 수십 개의 기능을 구현할 수 있어 전체 OBC 제작비용을 줄일 수 있는 이점이 있다. 특히 고 신뢰도의 FPGA Coding 기법을 이용한 고가 부품의 FPGA로의 구현은 상당한 비용 절감 효과를 거둘 수 있을 것으로 예상된다. 예를 들어 Communication Controller와 같은 부품의 경우 고가이며 많은 기능을 내장하고 있으나 정작 사용자가 사용하는 기능은 그중 일부에 불과하다. 따라서 이들 기능을 FPGA로 구현하면 고가의 Communication Controller를 구매하는 대신 FPGA Coding 기법을 이용하여 저렴하게 신뢰성을 확보하면서 원하는 기능을 구현 할 수 있다. 즉 FPGA를 이용한 OBC 설계기술은 부품 소요 경비를 줄일 수 있어 여러 단위 부품구매에 따른 외화 유출을 줄일 수 있다.

또한, 여러 부품의 집적화에 따라 OBC 및 위성의 각 모듈을 소형화/경량화 할 수 있고, 결국 위성 전체의 무게를 줄일 수 있다. 위성의 소형화/경량화는 위성의 전력 소모량과 무게 및 크기를 줄일 수 있어 위성 발사 비용 등 전체 위성 개발비용을 줄일 수 있는 효과를 얻을 수 있다.

3 절 연구 개발의 범위

1 연구개발의 최종범위

본 과제는 FPGA를 이용하여 우주용 On-board Computer(OBC)를 소형화/경량화 하고, 위성에 실제 적용할 수 있는 수준의 OBC를 개발 한다.

구체적인 개발 범위는 다음과 같다.

- CPU: 32bits, 50MIPS 이상
- EDAC: FPGA 구현
- Communication Controller: 100K bps 이상, FPGA 구현
- Watchdog Timer: FPGA 구현
- 각종 제어 신호: FPGA 구현
- Fault Tolerant 구조
- 무게: 900g 이하
- 크기: 200x300x30mm
- 공급전압: 5±0.25V
- Module Interface: 9.6K, 19.2K, 59.6K, 100K bps RS422 Protocol

2 연차별 연구개발 범위

FPGA를 이용하여 우주용 On-board Computer를 소형화/ 경량화 하기 위해 3차년도에 걸쳐 보드를 개발한다. 1차 년도에는 FPGA를 이용한 소형/경량화된 OBC 개념 및 구조 설계하는 것이 개발 범위이며, 2차 년도에는 FPGA를 이용한 로직 구현이 개발 범위이고, 3차 년도에는 시제품에 해당하는 보드를 개발하는 것이 개발 범위이다. 이에 해당하는 내용에 대해서는 다음의 표에 나타나 있다.

구분	연구개발범위	연구개발내용
1차년도 (2001)	FPGA를 이용한 소형/경량화된 OBC 개념 및 구조 설계	ㅇOBC 개념설계:32bit CPU ㅇCPU Core의 VHDL Code화 검토 ㅇOBC 시스템 구조 및 접속 설계 ㅇFPGA 구현 Logic 범위 설정 및 타당성 검토 ㅇFault Tolerant 구조 설계
2차년도 (2002)	FPGA를 이용한 로직 구현	ㅇ SEU, SEL을 고려한 고신뢰도 VHDL 구현 기법 연구 ㅇ EDAC 회로의 VHDL 구현 ㅇ Communication Controller 회로의 VHDL 구현: 100KBps 이상 ㅇ Watchdog Timer 및 각종 제어로직의 VHDL 구현
3차년도 (2003)	OBC 시제품 제작 및 성능 분석	ㅇ시제품 제작 ㅇ회로기판의 소형화 설계 ㅇ신뢰도 계산 ㅇ기능시험 및 성능 분석

제 2 장. 국내외 기술 개발 현황

1 절 위성용 보드 개발 현황

1 외국의 경우

위성 개발에 있어서 OBC(On-Board Computer, 탑재컴퓨터)의 개발은 필수적이므로 외국 우주 선진국의 경우 오랜 동안 OBC 설계 기술을 축적해 왔다. 이들 외국 기관으로는 NASA의 JPL, 유럽의 ESA, 일본의 NASDA, 기업으로는 미국의 Lockheed Martin, TRW, 그리고 초소형 위성의 경우 영국의 Surrey 대학 등을 들 수 있으며, 이들의 OBC 설계 기술은 완숙 단계에 있다고 할 만하다.

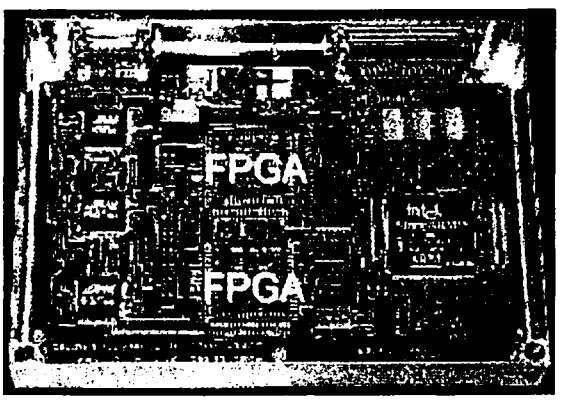


그림 1 Surrey Satellite Technology Co. 에서 개발되어 UoSat 시리즈에 탑재된 OBC

FPGA기술의 우주용 모듈설계에의 적용은 그동안 우주에서의 활용 가능성이라던가 사용되어진 사례들에 대하여 별로 알려진 바가 없어서 안정성에 불안감을 가져 왔다. 하지만, 최근 ACTEL사와 같은 FPGA 제조업체들은 Rad. Harden, Rad. Tolerant, HiRel FPGA등 다양한 등급의 FPGA를 개발 공급하고 있으며, 특히 이들 제품들이 Atlas II, Echostar, SBIRS-High, International Space Station, Hubble Space Telescope, Mars Pathfinder 등에서 적용된 사례가 보고 되고 있다. 또한 1990년 이후부터는 FPGA가 명령 및 데이터 처리, 자세제어 서브시스템 등에서도 적용되어진 것으로 파악된다. 예를 들어 영국 Surrey Satellite Technology, Co.의 최근 UoSat 시리즈에서는 FPGA를 이용한 OBC가 탑재되었다.

따라서 외국 선진국의 경우 FPGA를 이용한 본격적인 우주용 OBC의 개발은 1990년도 이후부터 진행되고 있다고 판단된다.

2 국내의 경우

한국과학기술원 인공위성연구센터에서는 1999년부터 발사된 우리별 3호와 2003년 발사한 과학위성1호의 개발을 계기로 독자적으로 OBC를 설계, 개발하였으며, 특히 과학위성 1호에서는 ACTEL사의 FPGA를 활용하여 EDAC 회로 및 여러 제어 신호를 구현하려는 시도를 하였다.

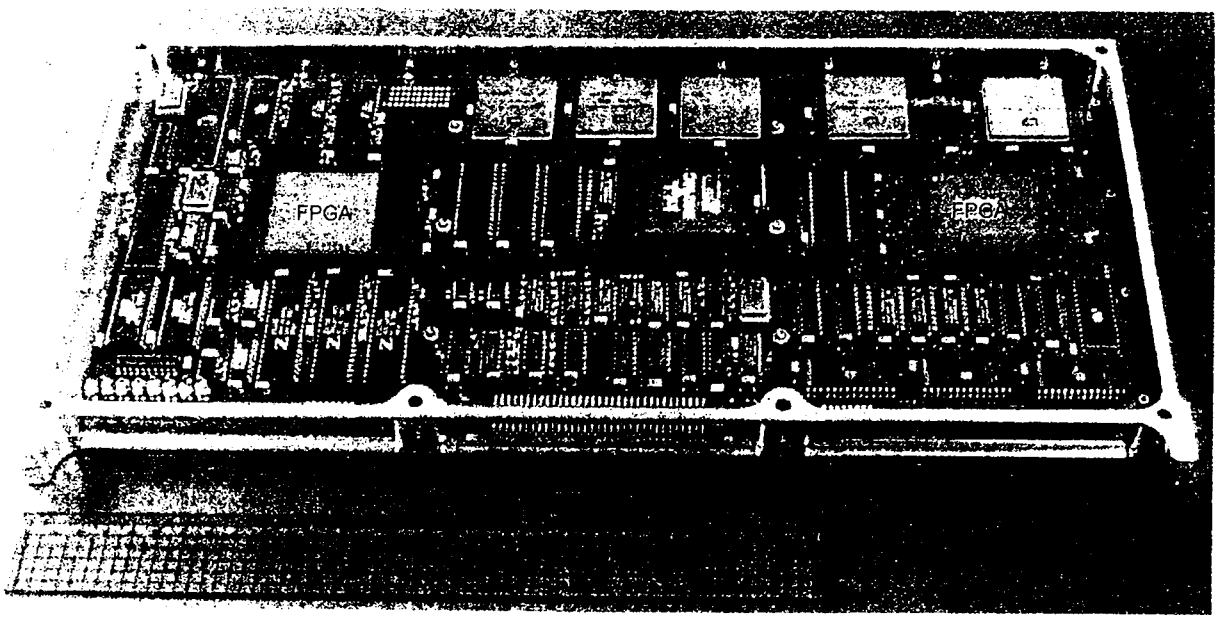


그림 2 인공위성연구센터에서 개발한 과학위성 1호 OBC

3 조사연구개발사례에 대한 평가

외국 우주 선진국의 경우도 FPGA를 이용한 우주용 OBC의 개발은 최근에 이루어진 것으로 보이며, 따라서 국내에서의 이 기술의 축적은 FPGA를 이용한 우주용 모듈의 설계 기술에 있어서 선진국과의 격차를 줄일 수 있는 좋은 계기가 될 것으로 보인다.

제 3 장. 연구개발수행 내용 및 결과

1 절 인공위성의 동작 환경 및 규격 연구

1 개요

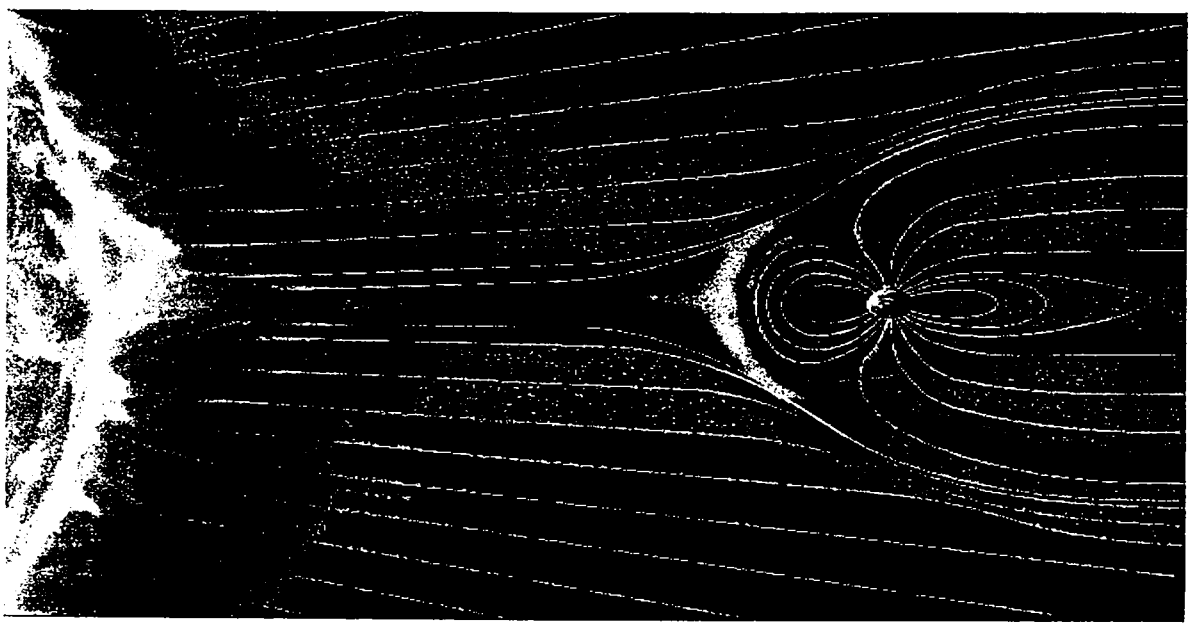
위성 탑재용 전자 장비가 갖추어야 할 요구 조건으로 지상의 동작 환경에 대한 요구 사항에 추가해서 우주 환경과 궤도 진입에 요구되는 발사 환경에 만족하여야 한다.

우주 환경 중 특히 전자 장비가 지상의 요구 조건과 비교해 요구되는 사항은 고 진공 상태의 공간에 주변 입자의 밀도가 적은 관계로 태양에서 나오는 복사열에 의해 빛이 비취는 부분과 그늘진 부분의 온도 차이가 아주 크다. 또한 지상처럼 대기의 대류 현상에 의한 열전달이 이루어지지 않기 때문에 전자 장비 중 열이 많이 나는 소자는 전도에 의하여 열전달이 이루어 질 수 있도록 설계하여야 한다. 따라서 사용될 전자 소자의 동작 온도는 지상에서 보편적으로 사용되는 상용 소자의 동작 범위인 0℃에서 70℃보다 큰 -55℃에서 125℃까지의 범위를 갖는 소자들을 택하여 전자 장비의 동작 신뢰도를 높이도록 한다. 우주 환경 중 전자 장비가 고려해야 될 또 하나의 요인은 우주 방사선이다. 지구 자기권에 의해 지구 주변에 고립된 입자들은 그 분포 밀도가 크지는 않으나, 전자소자의 수명을 결정하는 중요한 요인이다. 따라서 위성의 수명 기간 동안 우주방사선이 전자 소자들에 계속해서 누적되는데, 이 누적되는 총 방사선 량(Total Ionizing Dose)에 대하여 견딜 수 있도록 설계된 전자 소자들을 사용하여야 한다. 그러나 이러한 내 방사선 소자들에 대한 기술은 모든 전자 소자들에게 적용할 수 있는 단순한 문제가 아니라, 사용되는 물질과 소자 내부의 구조에 따라 그 특성이 매우 다르게 나타난다. 특히 최근의 반도체 기술의 개발 동향은 새로운 기술들이 매우 빠르게 개발되고 있기 때문에 새로운 기술이 적용된 소자들에 대하여 내 방사선 설계를 적용하기가 어려운 상황이다. 따라서 이러한 경우 부분적으로 전자 장비 주변에 국부적으로 보호 벽을 둠으로써 방사선의 투과량을 줄이는 방법을 쓰기도 하지만, 방사선을 차폐하기 위한 물질들은 위성의 무게를 증가시키는 요인이 됨으로 한없이 두껍게 하지는 못한다. 우주 방사선에 대한 고려 중 다른 중요한 한 가지는 고 에너지 입자들에 의한 순간 이상 동작 현상들이다. 싱글 이벤트 영향(Single Event Effects)이라고 불리어지는 이 현상은 저

궤도 위성의 경우 남대서양 상공에 낮게 분포하고 있는 에너지를 띤 양성자 입자에 의해 집중적으로 발생하며(SAA: South Atlantic Anomaly), 우주에서 날아오는 고에너지 입자들에 의해서, 정지 괘도 위성 등 우주에서 동작하는 모든 전자 소자들에 나타나는 현상이다. 이중에서 대표적인 현상 중의 하나인 싱글 이벤트 업셋(SEU: Single Event Upset)은 전자 장비의 소자에 기록된 정보를 '1'에서 '0'으로 '0'에서 '1'로 바뀌는 현상으로 위성에 탑재된 컴퓨터의 프로그램 메모리에 이러한 현상이 나타나면, 위성이 오 동작하게 되는 치명적인 현상이다. 따라서 위성에 탑재되는 모든 장비는 이러한 현상에 대비하여 보호 회로들을 사용한다.

싱글 이벤트 업셋(SEU)에 대한 보호회로는 일반적으로 프로그램 메모리 소자 등 메모리에 기록된 데이터 중 일부가 임의적으로 변하더라도 복구할 수 있도록 에러 정정 코드를 부호화 하여 별도의 메모리 소자에 기록하였다가 해당 메모리 소자의 내용을 컴퓨터가 읽어갈 때, 실제 데이터와 에러 정정을 위해 저장한 부호 코드를 검색하여 이상이 있는지를 확인하며, 이상이 있을 때에는 에러 정정 코드 기법에 의해 바로 수정이 되도록 시스템을 구현한다. 이러한 방법은 하드웨어적으로 에러를 수정하도록 하는 방법과 소프트웨어적으로 처리하는 방법이 있다.

2 지구 주변 우주 환경



인공위성을 비롯하여 우주공간에서 동작하는 모든 시스템은 우주 환경에 노출되게 되는데, 이 우주 환경에 대한 연구는 지난 수십 년 전부터 계속되고 있지만, 거대 공간에서 일어나는 복잡한 자연현상이기 때문에 아직도 학문적인 이해가 부족한 상태이다. 미국, 구소련, 유럽 연합 등 선진 우주 개발국의 연구결과로 지구 주변의 환경이 하나씩 밝혀지고는 있으나 복잡한 자연현상에 대한 연구는 계속되고 있다. 우리나라는 우리별 1, 2 및 3호와 다목적 실용 위성 1호의 과학 탑재체 개발로 우주 환경 관측을 위한 초기 단계 수준의 기술을 확보하고 있으나 지구 주변의 거대 공간에 대한 자료는 선진 우주 개발국의 자료를 기반으로 목적하는 위성의 궤도에 따른 우주 환경을 분석하여 전자 회로 등 설계에 반영하고 있다. 또한, 지구 주변의 환경에 대해서도 계속해서 새로운 모델들이 제시되고 있고 관측된 자료들을 토대로 보다 정확한 우주 환경을 예보하기 위해 많은 과학자들이 노력하고 있다.

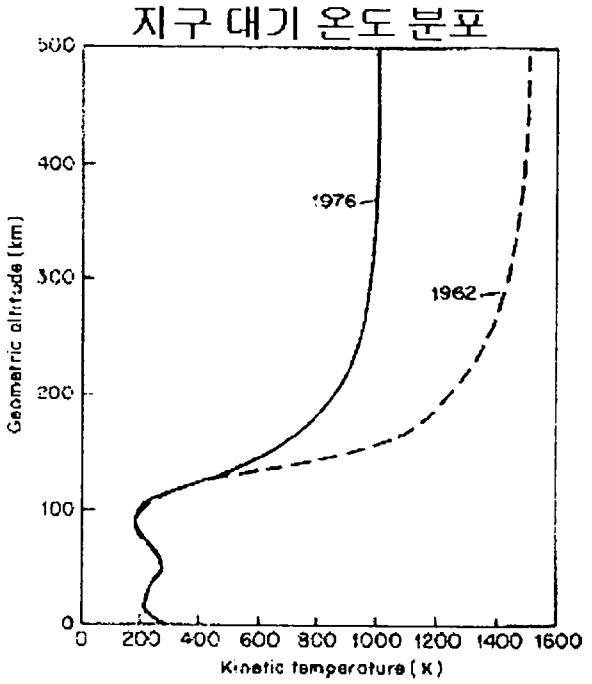
본 연구에서는 태양과 우주공간으로부터 기인하는 여러 가지 우주 환경 요소들에 대해 전자 장비들에 영향을 끼치는 주요 요소들에 대하여 살펴보았다.

가 태양풍

태양계의 모든 행성은 태양의 상태에 따라 많은 영향을 받게 된다. 특히 태양풍은 태양으로부터 나오는 고속의 플라즈마(Plasma)의 흐름으로 지구 주변에서 이 태양풍의 속도는 약 450km/s 이며 밀도는 약 9 protons/cm3 이다. 또한 태양의 흑점 폭발과 함께 고 에너지 입자들이 지구 주변으로 흘러오게 된다.

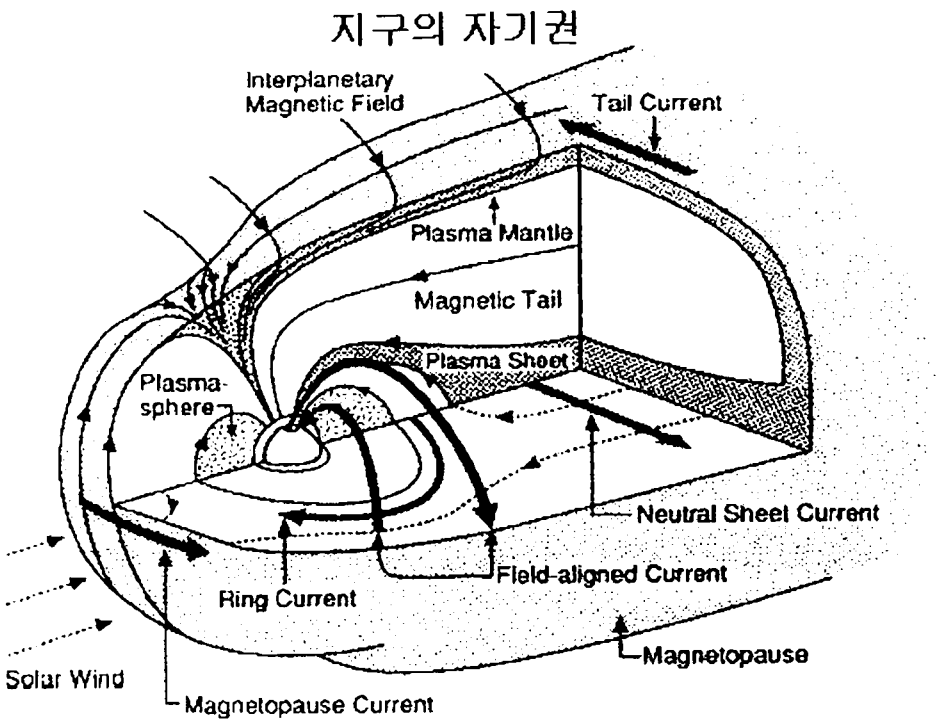
(1) 지구 주변의 대기 분포

미국의 1976 표준 대기 보고서에 의하면, 정지 궤도 위성이 있는 고도 36,000km 에서의 밀도는 성간 물질의 밀도와 거의 같은 수준으로 약 10^{-20}kg/m3 이고 압력은 약 10-15 Pa 정도로 지상의 대기압 1.013 X 105 Pa 과 비교하면 지상에서 구현하기 힘든 고 진공 상태임을 알 수 있다. 그림은 지구 주변의 대기 분포에 따른 온도 분포도이다.



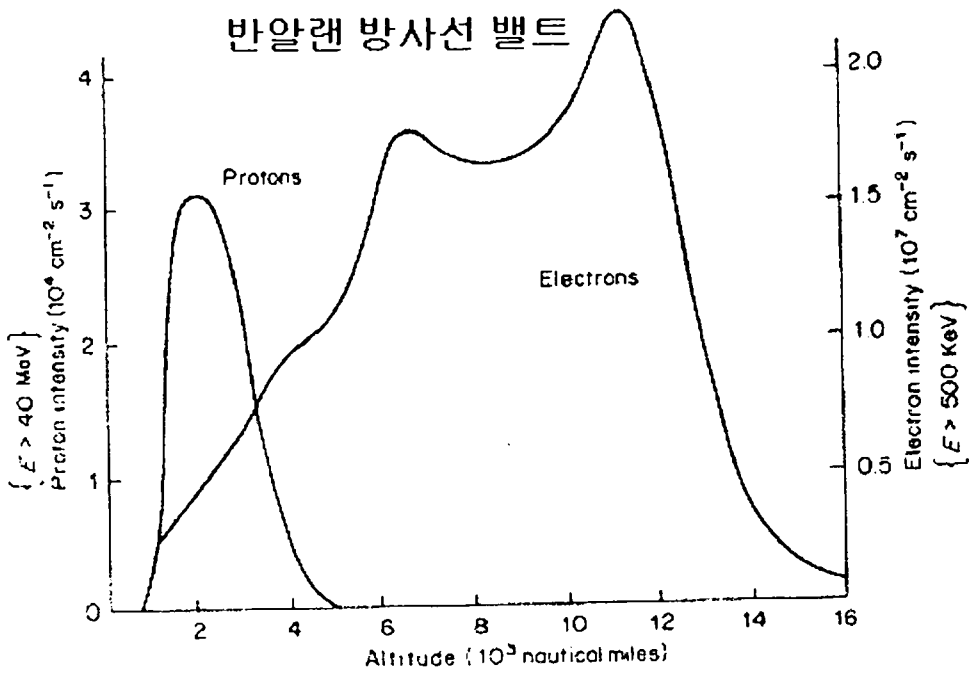
나 지구의 자기장

그림에서 볼 수 있듯이 높은 고도의 자기장의 구조는 매우 복잡하나, 저고도의 자기장은 관측이 가능하다. 이 지구 자기장에 의해서 다음에 설명하는 이온 방사선대가 형성 되는 등, 지구 주변의 환경에 큰 영향력을 행사하는 요소 이다.



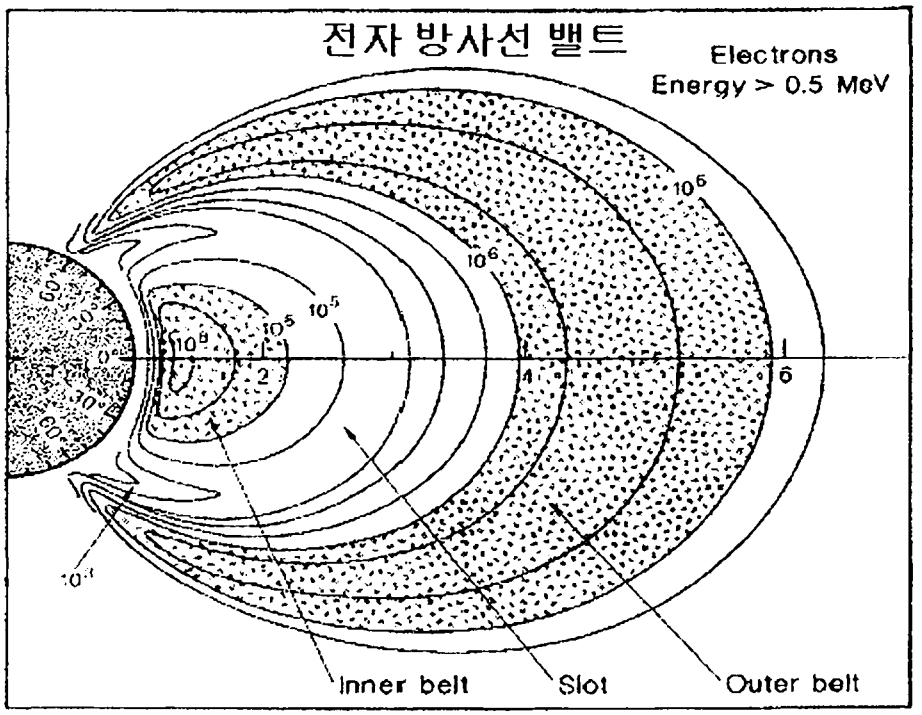
다 이온 방사능(Ionizing radiations)

(1) 반 알랜 방사선 밸트(The Van Allen radiation belts)



지구를 도넛 모양으로 감싸고 있으며 우리에게 '반 알랜 벨트'로 알려진 방사선 벨트는 에너지를 가진 양성자(Proton)와 전자(Electron)가 지구 자기장

에 의해 갇혀서 형성되있으며, 여기에는 또한 무거운 입자들인 헬륨(He), 질소(N) 및 산소(O) 이온들도 있다. 그림은 고도에 따른 양성자와 전자의 분포를 보여준다.



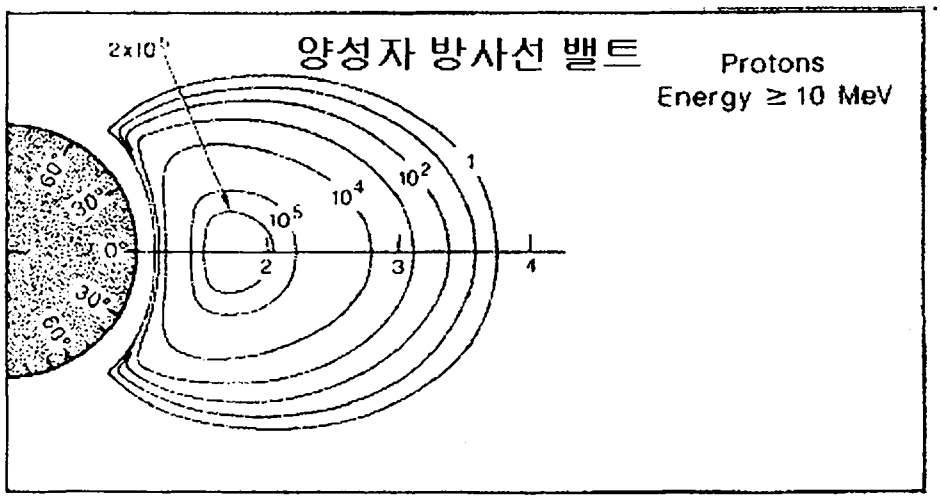
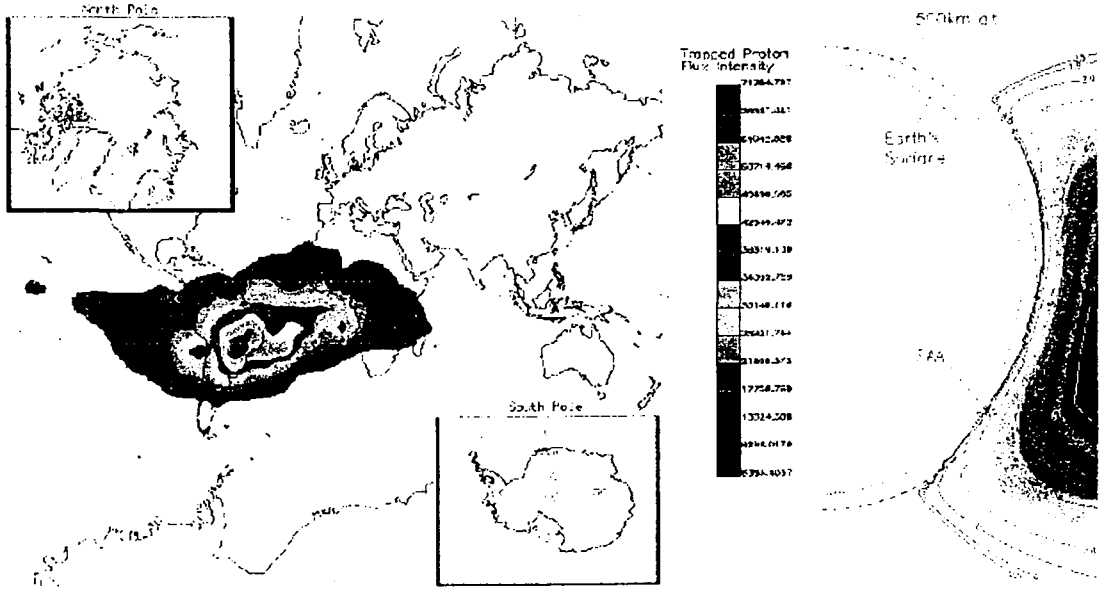


그림 전자와 양성자 벨트의 모양을 고도에 따라 에너지 분포로 나타낸 것이다. 그림 에서 알 수 있듯이 정지 궤도 위성이 위치하고 있는 고도 36,000km 상공에는 강한 전자 방사선 대역이 있음을 알 수 있으며, 주로 태양 동기 궤도를 갖고 있는 저

궤도 위성은 양성자 방사선의 영향 아래 있다.



저궤도에서 양성자 밸트와 위성에서 측정한 양성자 분포

그림은 양성자 방사선 벨트가 지구 자기장의 영향에 의해 남대서양 지역에 낮게 분포하고 있는 것을 보여주고 있다. 이러한 현상은 저궤도 인공위성이 이 지역을 지날 때 잦은 고장을 일으키는 요인으로서 남대서양 이상지역(SAA; South Atlantic Anomaly)으로 불려지고 있다. 이 지역은 양성자 입자에 의하여 위성의 컴퓨터, 메모리 등이 오동작을 일으키기 지역이다.

라 기타

이 외에도 은하에서 날아오는 우주선(Galactic cosmic radiation)들이 있으며 우주로부터 날아오는 입자들의 에너지는 태양에 의한 방사선 벨트에 갇히는 입자들보다 더 큰 고 에너지 입자들이 대부분으로 위성에 큰 영향을 끼칠 수 있다. 또한 우주공간에서는 정전기에 의한 표면 대전 현상이 있으며, 위성이 돌고 있는 궤도에는 우주로부터 날아오는 작은 운석들이 함께 존재하고 있어서 위성의 태양 전지판과 같은 부분에 영향을 끼치는 경우가 있다.

3 발사환경

위성체가 궤도에 진입할 때 받게 되는 발사 환경은 발사체가 지상에서 점화되는 순간부터 위성체가 목적하는 궤도에 진입할 때까지의 짧은 시간 동안 주어진다. 이

발사 환경은 위성체가 발사되는 발사체의 종류와 위성체와 발사체의 접속 방법, 및 위성체를 보호하고 있는 페어링(fairing)에 따라 달라진다.
일반적으로 발사체에 대하여 위성이 겪게 되는 진동과 충격들로는 기계 구조물에 대한 진동, 가속도에 의한 하중, 분리 과정의 파이로(Pyro) 충격, 대기권 상승 중 공기 마찰에 의한 열 충격, 대기압에서 진공 상태로 압력 변화 등이 있으며, 위성 탑재용 전자 장비는 이러한 진동 및 충격에 견딜 수 있도록 설계되어야 한다.
본 연구는 이러한 발사환경에 대한 대응 방법이라기보다 전자적 환경에 대한 대응 방법에 관한 연구를 하였으므로 세부적인 내용은 언급하지 않았다.

4 전자 부품의 규격

가 제조 회사의 전자 부품 등급 분류

전자 부품을 생산하는 회사들은 부품의 동작 온도 범위와 정해진 시험 항목의 적합성에 따라 제품의 등급을 표시한다. FPGA 제조업체인 ALTERA와 ACTEL사의 부품 분류 등급 기준을 보면 다음과 같다.

(1) ALTERA

- C-상용 부품(Commercial): 동작 온도 보증 범위가 0℃ ~ 70℃
- 산업용 부품(Industrial): 동작 온도 보증 범위가 -40℃ ~ 85℃
- M-군사용 부품(Military): 동작 온도 보증 범위가 -55℃ ~ 125℃
- M883B : MIL-STD-883에 의한 선별 시험된 부품
- MB : MIL-STD-883에 완전히 적용된 부품

(2) ACTEL

- 상용 부품(Commercial): 동작 온도 보증 범위가 0℃~ 70℃
- 산업용 부품(Industrial): 동작 온도 보증 범위가 -40℃ ~ 85℃
- M-군사용 부품(Military): 동작 온도 보증 범위가 -55℃ ~ 125℃
- B : MIL-STD-883에 의한 선별 시험된 부품
- E : E-Flow (Actel Space Level) : 우주용으로 시험된 부품

나 위성 탑재용 부품의 시험

제조업체에서 생산되는 부품을 위성에 탑재하려면 정해진 기준을 만족시켜야 한다. 이때 기준 규격을 만족하는지 여부를 시험하는 항목을 종합적으로 기술해 놓은 문

서가 미국의 표준 문서 MIL-STD-883이다.

MIL-STD-883은 전자 회로와 디바이스들이 군사용이나 항공 우주용 부품으로 사용되기에 적합하도록 기본적인 환경시험과 군용 및 우주 환경 조건 아래에서 시험하는 방법들을 통합하여 정리하고 있다. MIL-STD-883에서 기술된 각각의 시험 항목은 부품의 종류에 따라 민감한 정도가 다르므로 선별적으로 시험할 수 있으며, 항목 내부에도 등급이 나누어져 있어서 전체적인 선별과는 별도로 특정 부품이 민감한 항목에 강화된 시험을 수행할 수 있다.

(1) MIL-STD-883 의 시험 항목 분류

- 환경 시험 (Environmental Tests) : 1001에서 1999까지의 시험 방법
- 기계적 시험 (Mechanical Tests) : 2001에서 2999까지의 시험 방법
- 전기적 시험 (Electrical Tests) : 3001에서 4999까지의 항목으로 3000에서 3999까지는 디지털 회로에 대한 것이고 4000에서 4999까지는 아날로그 회로에 대한 시험 방법이다.
- 시험 절차 (Test Procedures) : 5000에서 5999까지의 항목

위에 기술한 것 외에도 다른 다양한 부품의 시험 및 절차 방식이 있다. 하지만 본 연구에서의 목적 상 주어진 환경을 극복하는 것이기에 더욱 자세한 사항은 기술하지 않았다.

5 위성 탑재용 장비의 설계 요소

가 전자 소자의 동작 온도

위성 탑재용 정보 보호 토큰의 전자 부품들은 위성이 겪게 되는 온도 환경에 견딜 수 있는 소자가 선정되어야 한다. 일반적으로 위성체는 각 서브시스템이 요구하는 온도에 맞추기 위해서 열 제어 시스템을 가지고 있으나, 통상 이러한 열 제어 시스템이 있더라도 위성체에 문제가 생겼을 경우 지상에서처럼 복구할 수 있는 방법들이 없기 때문에 전자 부품들은 위성체가 겪게 될 최악의 환경에 견딜 수 있는 부품이 선정된다. 앞에서 설명한 것처럼 위성에 사용될 부품의 온도 범위는 군사용 급으로 제작된 부품의 온도 범위를 만족하는 소자가 보편적으로 사용되고 있다. 이러한 군사용 급 부품 소자도 각 부품 제작 회사마다 규격이 다를 수 있으나 군사용

부품 소자의 경우 -55℃ ~ 85℃의 온도 범위에서 동작하도록 제작되어졌다.

나 총 방사선양(Total Ionizing Dose Effects: TID)

우주 환경이 지상의 환경과 다른 것 중 하나는 우주 방사선에 노출되는 점이다. 우주 환경은 원자력 시설물처럼 많은 양의 방사선에 순간적으로 노출되는 것은 아니고 적은 양의 방사선에 지속적으로 노출되어 있다는 점을 고려하여야한다. 따라서 위성의 설계 수명 동안, 사용되는 부품들이 정상적으로 동작하기 위해서는 위성의 궤도에 따른 방사선량을 예측하여, 예상 되는 수명 기간 동안 예측된 방사선에 노출되어도 동작할 수 있도록 하여야 하며, 현재의 기술로 아직 방사선에 영향을 받지 않는 소자 기술이 개발 되지 않은 부품들에 대하여는 사용될 부품 주변에 부분적으로 보호 물질을 설치하여 방사선의 영향을 줄이도록 설계하여야 한다.

6 싱글 이벤트 영향 및 극복 방법

가 싱글 이벤트 영향

위성이 동작하는 우주 궤도에는 앞에서 설명한 것처럼 우주 방사선들이 반도체 소자에 영향을 끼치는데 이러한 방사선 중에는 고 에너지 입자들이 존재 한다. 이 고에너지 입자의 밀도는 앞에서 언급한 것처럼 높지 않지만, 하나의 고에너지 입자가 반도체 소자에 부딪치게 될 경우 반도체 소자에는 다음과 같은 이상 현상을 일으키게 된다.

Acronym	Definition	Description
SEU	Single Event Upset	Change of information stored
SED	Single Event Disturb	Momentary disturb of information stored in memory bit
SET	Single Event Transient	Current transient induced by passage of a particle, can propagate to cause output error in combinational logic
SEDR	Single Event Dielectric Rupture	Essentially antifuse rupture
SEGR	Single Event Gate Rupture	Rupture of gate dielectric caused by a high current flow
SEL	Single Event Latchup	High current regenerative state induced in 4-layer device (latchup)
SES	Single Event Snapback	High current regenerative state induced in NMOS device (snapback)
MBU	Multiple Bit Upset	Several memory bits upset by passage of the same particle
SEFI	Single Event Functional Interrupt	Corruption of control path by an upset

SEU는 저장된 정보가 바뀌는 현상이며, SED는 저장된 정보가 순간적인 왜란에 의해서 잠시 바뀌는 현상이며, SET는 조합회로에서 입력이 변하지 않았음에도 불구하고 출력 값이 순간적으로 변하는 현상이며, SEDR은 아예 끊어져 버린 현상이며, SEGR은 고 전류에 의해서 게이트가 망가진 현상이며, SEL은 LatchUp 에 의해서 고전류가 흐르는 현상이며, SES는 SNAPBACK 에 의해서 NMOS 기기에서 고전류가 흐르는 현상이며, MBU는 여러 개의 비트가 정보가 바뀌는 현상이며, SEFI 는 UPSET에 의해서 제어 구조가 바뀌어 기능이 방해 받는 현상이다.

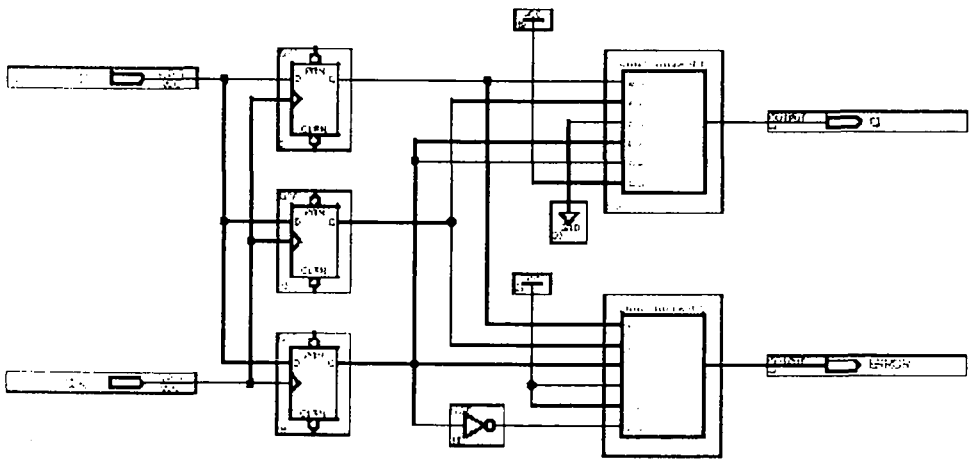
나 SEU의 극복

따라서 위성 탑재용 전자 장비들을 설계 할 때에는 이러한 싱글 이벤트 영향을 고려하여 고 에너지 입자가 반도체 소자에 부딪치게 될 경우에도 복구가 가능한 경우 바로 복구가 가능하도록 설계하여야 한다. 예를 들어 싱글 이벤트 업셋(SEU)은 고 에너지입자가 메모리 소자에 부딪치게 되면 기록된 정보가 1에서 0으로 0에서 1로 바뀌는 현상이 나타남으로 메모리 부를 설계할 때에는 이러한 현상에 대처하여 메모리에 기록된 정보를 부호화 하여 보호 코드를 갖도록 설계하고 시스템이 동작하

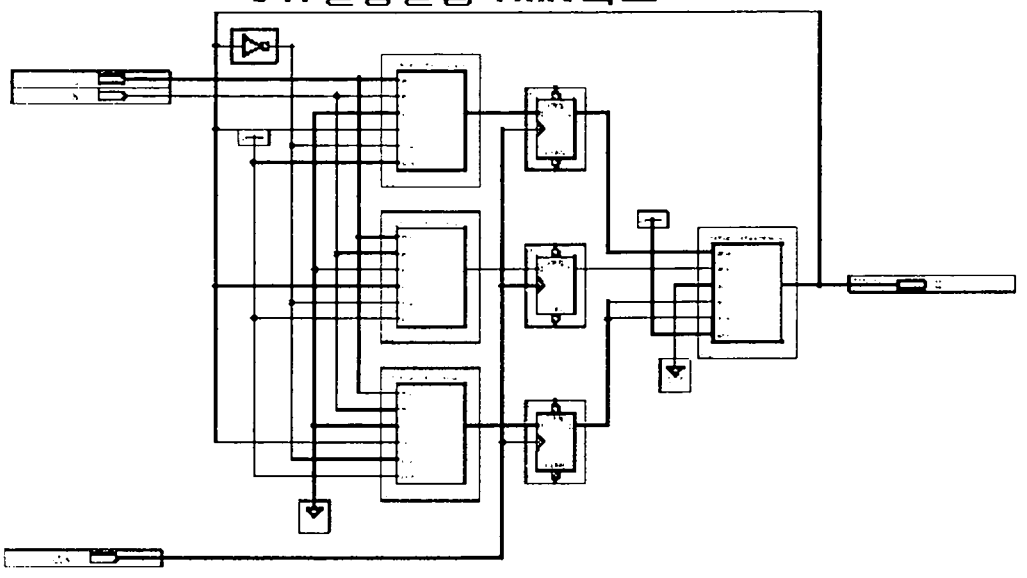
는 동안 메모리에 기록된 내용과 보호 코드에 기록된 내용이 일치하는지를 주기적으로 검사하여 싱글 이벤트 업셋에 대하여 영향을 받지 않도록 설계한다.

대표적인 예로는 프로세서 주변의 메모리를 제어할 때에 해밍코드(Hamming Code) 등을 사용한 오류 보호회로가 추가되는데, 이러한 시스템에서는 시스템에 필요한 메모리 소자 이외에 부가적인 에러 검출 및 정정용 메모리가 추가로 요구된다.
아울러 하드웨어 수준의 오류 정정을 위하여 사용되는 방법 중의 하나가 삼중 보호회로(TMR : Triple Modular Redundancy)로 불리어지는 회로로, 이는 위성에서 고 에너지 입자에 의해 발생될 수 있는 오류가 메모리 소자들의 한 개영역에서 발생한다는 점에 기반을 두고 만들어진 회로이다.
최근에는 사용자가 회로를 설계하여 하나의 칩에 집적하여 만들 수 있는 FPGA(Field Programmable Gate Array)소자들을 사용하면서 이러한 오류 정정 기법을 FPGA내의 SIGNAL에 대해서도 사용되고 있으나, 하나의 메모리 영역을 위해서 필요한 부가 회로가 많아지는 단점을 않고 있다. 이러한 예로 D-플립플롭(D-Filp Flop)과 J-K 플립플롭(J-K Filp Flop) 소자를 이용한 TMR 기법을 사용하기도 한다.
그림의 회로에서 알 수 있듯이 하나의 D-Flip Flop에 오류가 발생해도 출력단의 신호에는 정상적인 출력이 나옴을 알 수 있다

D 플립플롭 TMR 회로



J-K 플립플롭 TMR 회로



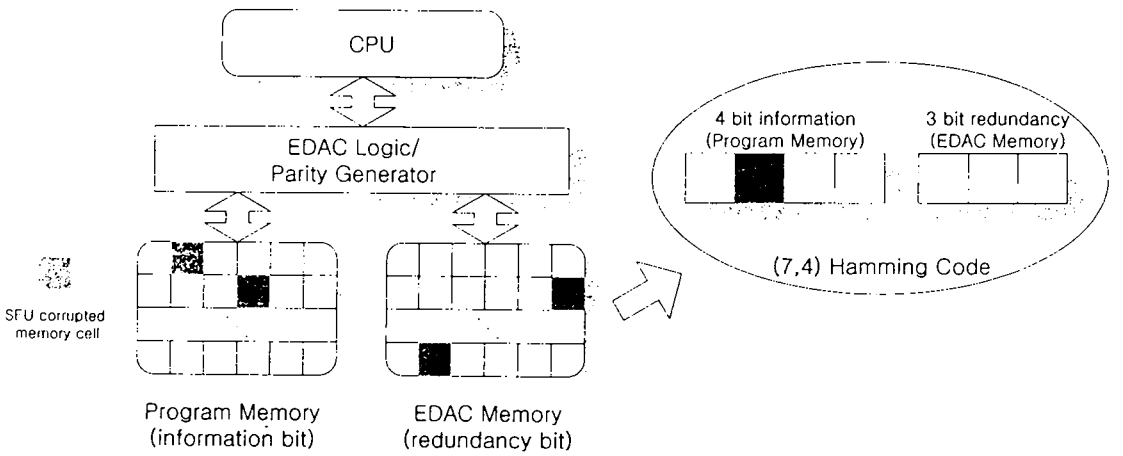
위성 탑재용 전자 회로의 오류 검출 및 정정 회로들로는 해밍 코드와 TMR회로 이외에도 통신과 관련하여 Parity, CRC(Cyclic Redundancy Check), Reed-Solomon Code, Convolution Code, Overlying Protocol등이 사용되고 있다.

Parity 방식은 여분의 비트를 이용하여 정보의 오류를 저장하는 방식으로 한 개 bit의 오류 검출하는 방식이다. CRC방식은 주어진 데이터 구조에 에러가 발생할 때 오류를 검출시 사용할 수 있는 방식이며,Hamming Code는 종류가 여럿 있지만, 과학 위성1호의 경우 한 개 Bit의 오류 복구 및 두 bit 오류의 검출할 수 있게 (7,4) Hamming Code를 사용하였다. Reed-Solomon Code은 여러 개의 bit 오류 및 byte 오류를 복구할 수 있는 방식이다. 과학위성1호에서 탑재체에서 얻어진 저장된 정보를 저장하는 MMS시스템에서 쓰고 있으며 (187,207) Reed-Solumn 코드를 이용하여 10바이트에 대한 BURST 오류에 대한 복구가 가능한 방식을 사용하고 있다. Convolution Code은 통신 채널에서의 오류 복구 하는 방식이며, Overlying Protocol은 각 시스템별 통신 규약에 따라 오류 복구 하는 방식으로 예를 들면 재전송 프로토콜 등이 있다.

다 우리별3호와 과학기술위성 1호의 EDAC 알고리즘

프로그램 메모리의SEU 극복을 위해 우리별 3호의 경우 8비트 정보비트(Information bit)마다 4비트의 여분 비트를 사용한 EDAC 알고리즘을 사용하였으

며, 8비트당 1비트의 에러를 복구하고 2비트 에러는 탐지할 수 있었다. 또한 과학위성 1호는 에러 복구 능력 및 FPGA로 EDAC 알고리즘 구현시 사용되는 로직 수, I/O 수 그리고 구현의 복잡성 등을 고려하여 (7,4) Hamming Code 를 이용한 EDAC 알고리즘을 사용하였다. 이 알고리즘은 4bit의 정보 비트(information bit)와 3bit의 여분 비트(redundancy bit)를 사용하여 1bit의 오류를 탐지 및 복구하며, 2bit의 오류는 탐지 할 수 있다. [그림]은 과학위성 1호 탑재 컴퓨터의 주 메모리 구조이며, 이것은 정보 비트를 저장하기위한 프로그램(program) 메모리와 여분의 비트를 저장하기위한 EDAC 메모리를 표현하였다.



이러한 EDAC(Error Detection And Correction) 알고리즘의 3비트의 정보 비트는 다음과 같다.

R(2) = I(3)+I(2)+I(1)

R(1) = I(2)+I(1)+I(0)

R(0) = I(3)+I(2)+I(0)

에러의 복구는 먼저 정보 비트와 여분 비트를 이용하여 3비트의 Syndrome을 구한 다음 테이블을 이용하여 에러가 발생한 비트를 반전 시킨다. 또한 에러 복구는 여분 비트들에서 발생한 에러에 대해서는 하지 않고, 정보 비트들에서 에러가 발생하였을 때만 하여 EDAC 알고리듬을 FPGA로 구현했을 때 사용되는 로직수를 최대한 줄이도록 한다.

S(2) = (I(3)+I(2)+I(1))+R(2)

S(1) = (I(2)+I(1)+I(0))+R(1)

S(0) = (I(3)+I(2)+I(0))+R(0)

Syndrom(2:1:0)	ErrorVector I (3:2:1:0)	R(2:1:0)	Digit In Error
000	0000	000	NONE
001	0000	001	7
010	0000	010	6
011	0001	000	4
100	0000	100	5
101	1000	000	1
110	0010	000	3
111	0100	000	2

2 절 부품의 선정

1 CPU 선정

우주용 On-Board Computer의 CPU 선택은 위성의 Mission 자체에 큰 영향을 끼치는 아주 중요한 문제이다. Power PC 603e를 우주용 탑재 컴퓨터의 CPU로 적절한지에 대해 충분히 살펴보아야 한다. 먼저 소형위성의 Mission을 수행하기에 충분한 CPU의 능력이 되는지 알기 위해 현재 개발 중인 과학기술위성 2호의 Mission에 대해 조사해 보면 지상국 명령 및 데이터 처리, 위성 자세 제어, 위성체 운용, 탑재체 운용, 전력부 제어 등의 작업을 수행한다. 상용급이나 실용급 인공위성의 경우 자세제어, 지상국 명령 및 데이터 처리, 전력 관리 등 인공위성의 운용 상 중요한 부분에 대해 각각 개별 탑재 컴퓨터를 적용하여 개발하지만 100Kg급의 과학기술위성 2호는 그 특성상 이와 같이 구성하기가 어려우므로 하나의 탑재 컴퓨터로 필요한 모든 작업을 수행해야 한다. 즉 이러한 탑재컴퓨터는 저전력 소모, 저중량, 소형화가 그 설계의 제한요건으로 고려되어야 하고 이러한 제한 요건을 만족하는 범위 내에서 고도의 신뢰성과 성능을 발휘할 수 있도록 설계되어야 한다. 이전에 발사된 과학기술위성 1호의 CPU를 예를 들면 Intel사의 군용급 80960이 사용되었다. 80960의 성능은 최대 25MHz에서 25MIPS이며 2.5W를 소비한다. 과학기술위성 1호에서는 제한된 소비 전력 때문에 80960이 10MHz 클럭으로 사용되었다. 탑재 컴퓨터의 허용 소비 전력을 증가하여 80960을 최대 클럭으로 사용하면, 탑재 컴퓨터의 성능을 향상 시킬 수가 있지만 Intel사의 군용 부품 생산중단으로 인하여 향후 사용이 불가능하게 되었다. 이러한 CPU 공급 상황을 극복하고 탑재컴퓨터의의 성능 향

상을 위하여 새로운 CPU의 선정이 필요하다. 새로운 CPU의 선정은 다음과 같은 선정기준을 정하여 선택하였다.

가 군용급 CPU

우주 복사 환경에 의한 SEP(Single Event Phenomena)를 고려한다면 복사 강화(Radiation Hardened) 또는 복사 허용(Radiation Tolerant) 버전의 CPU를 사용하여야 한다. 하지만 그 가격이 소형 인공위성 개발 프로그램의 예산에 대비해 비싸고 과거 우리별 1,2,3호와 과학기술위성 1호의 경험으로 저궤도 위성의 군용급 CPU로도 안정적인 운용을 할 수 있다고 판단되어 군용급 CPU로 제한하였다.

나 패키지 형태

인공위성에 사용될 부품의 패키지 형태는 부품의 실장 상태 유지를 위해서 고려해야 한다. 인공위성 발사체에 탑재되어 궤도에 올려지는 과정에서 겪는 강한 충격과 운용 중에 겪게 될 온도 변화에 견딜 수 있어야 한다. 이러한 상황에 적합한 패키지 형태는 PGA(Pin Grid Array)나 QFP(Quadrature Flat Package)이다. 하지만 CPU의 내부 집적도가 높아지고 그 크기의 소형화 요구가 강해짐에 따라 BGA(Ball Grid Array) 형태의 패키지가 많이 채용되고 있다. BGA 패키지는 패키지 재료와 부품이 실장되는 PCB(Printed Circuit Board) 재료의 열팽창 계수 차이로 인하여, 온도의 변화 범위가 큰 환경에서는 부품과 PCB의 접합부에 균열이 발생할 수 있다. 이러한 문제를 극복하기 위하여 CGA(Column Grid Array) 패키지가 개발되었으며 상용의 우주용 복사 강화 CPU가 CGA 패키지로 생산되고 있다. 이러한 점을 감안하여 PGA와 QFP 패키지를 우선 고려하고 이들 형태로 공급되지 않는 경우에는 CGA 패키지까지 허용하는 것으로 한다.

다 FPU (Floating Point Unit) 내장

FPU는 탑재컴퓨터에서 인공위성의 자세제어를 위한 알고리즘을 수행하기 위해 필수적인 요소이므로 FPU가 내장된 CPU로 대상을 제한한다. FPU가 내장되어 있지 않은 CPU에 대해서는 외부에 함께 사용할 수 있는 Floating Point Co-processor가 공급되는 경우가 있는데 이러한 형태는 지양하기로 한다. 부품의 수가 증가할수록 신뢰성은 감소하며 소비전력이 증가하기 때문이다.

라 상용 실시간 운영체제 지원 및 소프트웨어 개발 환경

탑재컴퓨터 하드웨어의 안정성 못지않게 중요한 것이 채용되는 운영체제의 안정성이다. 아무리 안정적인 하드웨어를 개발한다 하더라도 운영체제의 안정성이 확보되지 않으면 위성의 본래 임무를 수행하기 어려울 것이다. 따라서 현재 개발되어 각 방면에 적용되고 있는 실시간 운영체제가 지원되지 않는 CPU는 선정 대상에서 제외한다. 그리고 편리한 소프트웨어 개발 환경은 개발 소요기간 및 비용을 감소시키고 소프트웨어 신뢰성에도 영향을 미친다.

마 장래성 (재사용성)

안정성 및 성능이 검증된 탑재컴퓨터는 다른 인공위성 개발 프로그램에서 사용될 수 있다. 만일 CPU가 변경되면 관련하드웨어도 변경해야 하며 소프트웨어 구동을 위한 운영체제 역시 교체되어야 하므로 향후의 인공위성 개발 프로그램에 소요되는 시간과 비용이 증가하게 되고 하드웨어 및 소프트웨어의 신뢰성이 감소하게 된다. 따라서 CPU 선택시 향후 재사용 가능한 CPU 중 고기능을 갖추고 널리 사용되는 Core를 갖는 CPU를 선택하도록 한다.

이상과 같은 다섯 가지의 선정기준을 갖고 상용으로 공급되는 CPU 중 과학기술위성 2호의 탑재 컴퓨터에 사용할 CPU를 검토하였다. 현재 대중적으로 사용되는 CPU에는 Motorola사의 68XXX과 Power PC, Intel 사의 X86과 i960, Texas Instrument사와 Analog Device사의 DSP, 그리고 ARM, MIPS, SPARC 등이 있다. 이들 중 실시간 운영체제가 운용되어야 하는 인공위성 탑재 컴퓨터에는 부적합한 DSP와 군용급으로 생산하지 않는 X86, i960, ARM, MIPS, 그리고 SPARC를 제외하면 68XXX와 PowerPC가 남는다. 68XXX와 PowerPC 중 FPU를 포함하는 것은 68XXX 계열의 CPU 중에서는 68040이며 PowerPC 계열 중에서는 603e, 740 그리고 750이 있다. 68040은 PGA와 QGP 패키지로 공급되고 603e는 BGA와 CGA, 740과 750은 BGA 패키지로만 공급된다. 결국, 군용급, 패키지 형태, FPU, 상용 운용체제 지원의 선정기준을 적용하면 68040과 603e가 후보로 남게 되며 두 CPU의 개발환경은 큰 차이가 없다.

두 CPU에 대한 주요사항은 아래와 같다.

비교항목	603e	68040
성능	423MIPS @300MHz	26MIPS @33MHz
내부 클럭	300MHz	33MHz
Address	32bits	32bits
Data Bus	32/64bits	32bits
전원	3.3V	5V
소비전력	4/6W @300MHz	7/7.7W @33MHz
패키지	CGA	PGA, QFP

603e가 68040에 비해 최대 성능이 20배 정도에 달하고, 동일성능으로 운용시 소비 전력이 10분의 1이하임을 알 수 있다. 성능 및 소비 전력 면에서는 603e가 유리하지만, 패키지 형태에 있어서는 68040이 유리하다. 하지만 상용의 우주용 복사 강화 CPU가 CGA 패키지로 생산되고 있는 점을 감안하여 603e를 소형위성의 On-Board Computer 로 선정하면 되겠다. 과학기술위성 1호에 사용된 80960과 603e와 비교한 표는 아래와 같다.

성능	423MIPS@300MHz	25MIPS@25MHz
내부클럭	300MHz	25MHz
Address	32 bits	32bits
Data Bus	32/64 bits	32bits
전원	3.3V	5V
소비전력	4/6W @300MHz	2.1/2.5W @25MHz
패키지	CGA	PGA, QFP

비교항목

603e가 소비 전력 대비 성능에 있어서 80960 보다 뛰어남을 알 수 있다. Power PC 계열에는 603e보다 성능이 개선된 Core도 있고 동일 Core에 주변 장치를 내장한 CPU도 있다. 이들은 603e와 하드웨어적으로 비슷한 구조를 갖고 소프트웨어 호환성도 양호하기 때문에 603e를 적용하여 과학기술위성2호의 탑재 컴퓨터를 개발한다

면 성능 및 소비전력 면에서의 향상과 함께 향후 재사용 가능성에 대해서도 큰 효과를 볼 수 있으리라 여겨진다.

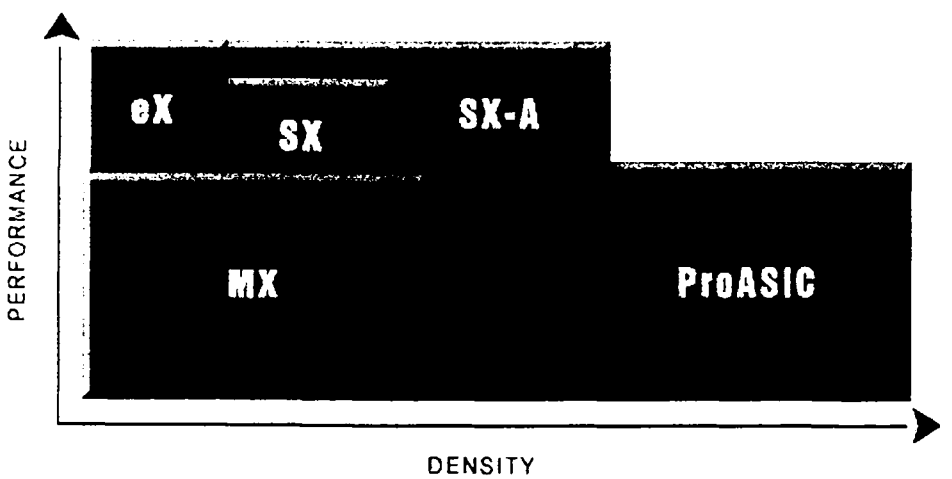
2 FPGA 선정

우주용 FPGA를 선택하기 위한 사항들은 무게, 부피, 전력, 내방사선성, 고 신뢰성 등을 들 수가 있다.

이번 "FPGA를 이용한 우주용 OBC 소형화, 경향화 기술 개발" 프로젝트에서의 FPGA 선택 기준은 기본적으로 위의 사항들을 준수하며 military spec의 존재 유무, 기존 heritage 유무, 가격 등의 조건과 특히 고집적도가 요구되므로 FPGA gate 수 등을 복합적으로 고려하여 FPGA를 선택하기로 한다.

우주용 FPGA는 Actel사와 Xilinx사가 주도를 하며 개발해 오고 있으며 실제 위성에서 응용된 사례가 많고 그 중에 Actel FPGA는 과학위성시리즈에서 heritage가 있으므로 Actel사의 FPGA 중에서 사용에 알맞은 부품을 선정하는 걸로 하였다.

2.1. Actel사 FPGA 종류



FPGA의 Density와 Performance 관계로 볼 수 있는 Actel FPGA Family 구성표이다.

1.1.1. General Purpose FPGAs

Model	Gates	Flip Flop	I/O	Temp Range	Speed Grades	

EX	3,000~12,000	128~512	84~132	C, I	-F, Std, -P	
SX	12,000~48,000	512~1,980	130~249	C, I, M	-F,Std,-P,-1,-2,-3	
SX-A	12,000~108,000	512~360	360	C, I, M	-F,Std,-P,-1,-2,-3	
MX	3,000~54,000	147~1,822	202	C, I, M	-F,Std,-P,-1,-2,-3	
ProASIC	150,000~1,000,000	56,320	712	C, I	Std	

일반적으로 사용하는 General Purpose FPGA Model이다. 본 과제에서 사용될 FPGA는 개발상의 편의를 위해 One Time FPGA을 쓰지 않고 ISP(In System Programming)기능이 있어서 프로그램을 계속 다운로드할 수 있는 ProASIC 모델을 사용을 한다. 그리고 향후 인공위성에 탑재되는 OBC의 FPGA는 우주환경에 강인한 SEU 특성이 좋은 FPGA를 사용하기로 한다.

2.1.2. Aerospace & HiRel

항공 우주용 및 고신뢰도 Actel FPGA Family의 목록을 나타낸다.

Model			Gates	Flip Flop	I/O	Temp Range	Speed Grades
RT54SX-S	RT54SX32S		48000	1,980	227	B,E	Std, -1
	RT54SX72S		108,000	4,024	212	B,E	Std, -1
Hirel SX-A	HiRel A54SX32A		48,000	1,980	249	C,M,B	Std, -1
	HiRel A54SX72A		108,000	4,024	360	C,M,B	Std, -1
54SX	RT54SX16		24,000		179	B,E	Std, -1
	A54SX16		24,000		180	C,M,B,E	Std, -1
	RT54SX32		48,000		227	B,E	Std, -1
	A54SX32		48,000		228	C,M,B,E	Std, -1
HiRel	3200DX	A32100DX	15,000	738	152	C,M,B,E	55MHz
		A32200DX	30,000	1,276	202	C,M,B,E	
	ACT3	A1425A	3,750	435	100	C,M,B,E	
		A1460A	9,000	976	168	C,M,B,E	60MHz
		A14100A	15,000	1,493	228	C,M,B,E	
	1200XL	A1280XL	12,000	998	140	C,M,B	50MHz
	ACT2	A1240A	6,000	568	104	C,M,B,E	40MHz
		A1280A	12,000	998	140	C,M,B,E	
	ACT1	A1010B	1,800	147	57	C,M,B,E	20MHz
		A1020B	3,000	273	69	C,M,B,E	
Radiation Hardended	RH1020		3,000	273	69	E	
	RH1280		12,000	998	140	E	
	RT1020		6,000		69	C,M,B,E	STD, -1
	RT1280A		24,000		140	C,M,B,E	STD, -1
Radiation Tolerant	RT1425A		7,500		100	C,M,B,E	STD, -1
	RT1460A		18,000		168	C,M,B,E	STD, -1
	RT14100A		30,000		228	C,M,B,E	STD, -1

2.2. Actel사 FPGA 특성 (SEU, SEL, TID)

2.2.1. Single Event Upset of FPGAs

Device	Tech-nology (m)	Manu-facturer	SEU LET Threshold (MeV cm^2/mg)	Sat X-Section(*1)
A1020B	1.0		28	2×10^{-6}
RH1020	1.0	ACTEL(L-M)		
A1280XL C	0.8	ACTEL(Winbond)	-	-
A1280XL S	0.8	ACTEL(Winbond)	-	-
A1280XL I/O	0.8	ACTEL(Winbond)	-	-
A1280A C	1.0	ACTEL(MEC)	28	2×10^{-6}
A1280A S	1.0	ACTEL(MEC)	5	8×10^{-6}
A1280A I/O In	1.0	ACTEL(MEC)	-	-
A1280A I/O Out	1.0	ACTEL(MEC)	28	-
RH1280 C	0.8	ACTEL(L-M)	22	8×10^{-6}
RH1280S	0.8	ACTEL(L-M)	4	9×10^{-6}
RH1280I/O	0.8	ACTEL(L-M)	-	-
A1460A C	0.8	ACTEL(MEC)	~30	$\sim 2 \times 10^{-7}$
A1460A S	0.8	ACTEL(MEC)	~8	1×10^{-6}
A1460A I/O	0.8	ACTEL(MEC)	~8	2×10^{-6}
A14100A C	0.8	ACTEL(MEC)	~28	1×10^{-6}
A14100A S	0.8	ACTEL(MEC)	~8	2×10^{-6}
A14100A I/O	0.8	ACTEL(MEC)	-	-

위의 표와 같이 C-Module은 S-Module보다 Radiation 특성이 매우 뛰어나기 때문에 Latch나 Flip-Flop을 사용할 때는 S-Module을 이용하는 것보다 C-Module을 이용하여 만드는 것이 SEU에 훨씬 강하게 나타난다는 것을 확인 할 수 있다.

참고로 과학기술위성1호의 OBC에 사용했던 FPGA는 A1460이었고 NC(Node

Controller) 보드에 사용한 FPGA는 A1280A였다.

2.2.2. Single Event Latchup of FPGAs

Device	Technology (m)	Manufacturer	SEL (MeV cm^2/mg)	SEDR(*1)
A1020B	1.0	ACTEL(MEC)	< 38	YES
A1020B	1.0	ACTEL(TI)	< 25	YES
RH1020	1.0	ACTEL(L-M)	NO	YES
A1280XL	0.8	ACTEL(Winbond)	NO	YES
A1280A	1.0	ACTEL(MEC)	NO(*2)	YES
RH1280	0.8	ACTEL(L-M)	NO	YES
A1460A	0.8	ACTEL(MEC)	NO	YES
A14100A	0.8	ACTEL(MEC)	NO	YES

- : Not Tested , ~ : Around

*1: Single Event Dielectric Rupture

*2 : Latchup detected only with MODE pin high.

2.2.3. Total Ionizing Dose of ACTEL FPGAs

Device	Technology (m)	Manu-facturer	TID Rad(Krad)	
A1020B	1.0	ACTEL(MEC)	>60	
A1020B	1.0	ACTEL(TI)	>60	
RH1020	1.0	ACTEL(L-M)	> 300	
A1280XL	0.8	ACTEL(Winbond)	2~4	
A1280XL	0.6	ACTEL(CSM)	2.5	
A1280A	1.0	ACTEL(MEC)	10	
RH1280	0.8	ACTEL(L-M)	> 300	
A1460A	0.8	ACTEL(MEC)	15	

A14100A	0.8	ACTEL(MEC)	15	
RT54SX16	0.6	ACTEL(MEC)	80	00T-Rt54SX16-T6HP12
RT54SX32	0.6	ACTEL(MEC)	80	00T-RT54SX32-T6JP01A

2.3. 결론

Actel FPGA의 Antifuse 방식을 이용한 Routing 기술은 타 기술에 비하여 전력적인 우위에 있으며 기본적으로 내방사선 특성이 우수한 것으로 보여 지며 타 위성에서의 응용 사례가 많은 것으로 알려져 있다.

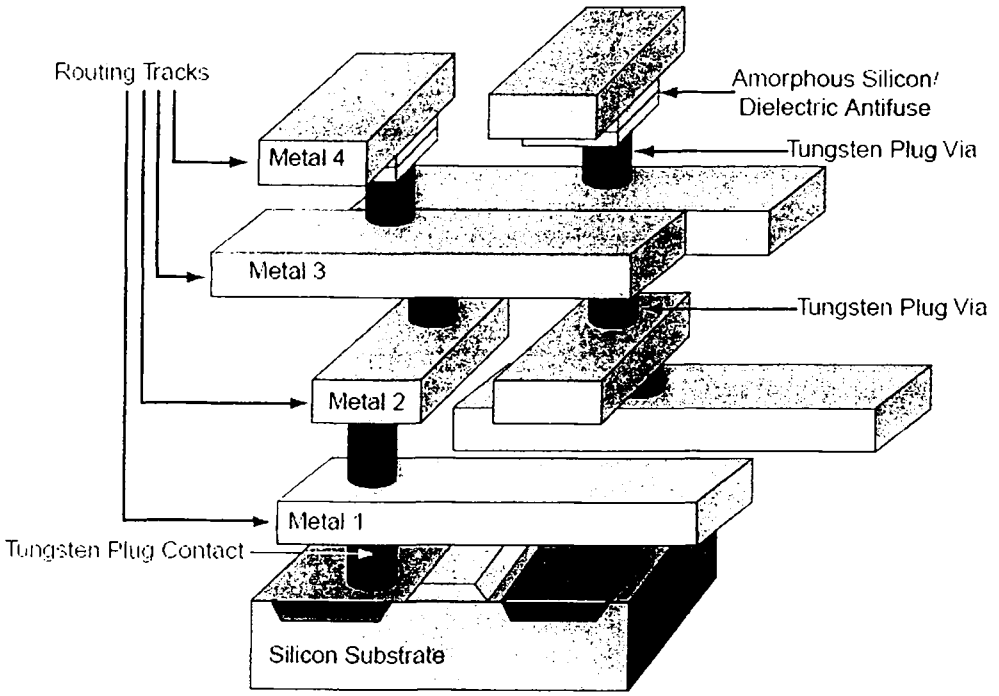
위에 열거된 FPGA 중에 Military Spec이 존재하지 않는 FPGA는 일단 배제를 하고, Radiation Hardened나 Radiation Tolerant FPGA는 Deep Space용으로 주로 쓰이고 가격이 너무 비싸므로 배제를 하고 내방사선 특성이 15kRad 이상의 것을 사용하기로 한다. 이 조건들을 만족하며 Gate수가 많은 모델인 HiRel A54SX72A를 사용하는 것이 가장 적합하다고 볼 수 있다.

좀 더 우주환경에 강인한 모델을 원한다면 RT54SX72S FPGA도 좋은데 HiRel A54SX72A와 같은 Architecture를 가지고 있으면서 FPGA 내부적으로 cell이 3중화가 되어 있어서 자체 TMR 기능을 가지고 있다.

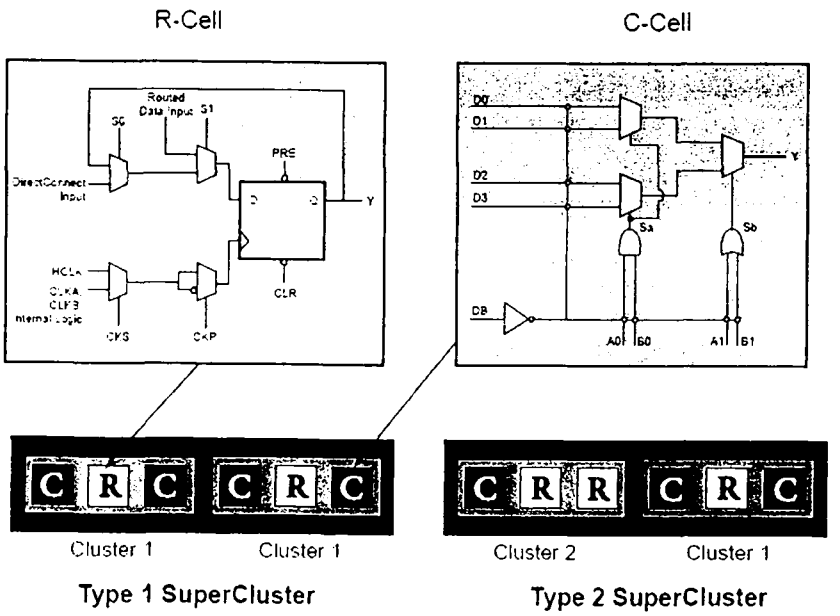
HiRel A54SX72A의 Spec은 아래와 같다.

Device	HiRel A54SX32A	HiRel A54SX72A
Capacity		
Typical Gates	32,000	72,000
System Gates	48,000	108,000
Logic Modules	2,880	6,036
Combinatorial Cells	1,800	4,024
Register Cells		
Dedicated Flip-Flops	1,080	2,012
Maximum Flip-Flops	1,980	4,024
Maximum User I/Os	228	213
Global Clocks	3	3
Quadrant Clocks	0	4
Boundary Scan Testing	Yes	Yes
3.3V/5V PCI	Yes	Yes
Clock-to-Out	5.3 ns	6.7 ns
Input Set-Up (External)	0 ns	0 ns
Speed Grades	Std, –1	Std, –1
Package (by pin count)		
CQFP	208, 256	208, 256

HiRel A54SX72A의 내부 구조이며 Total 4 layer의 Metal 층을 가지고 있으며 Antifuse 방식의 FPGA이다.

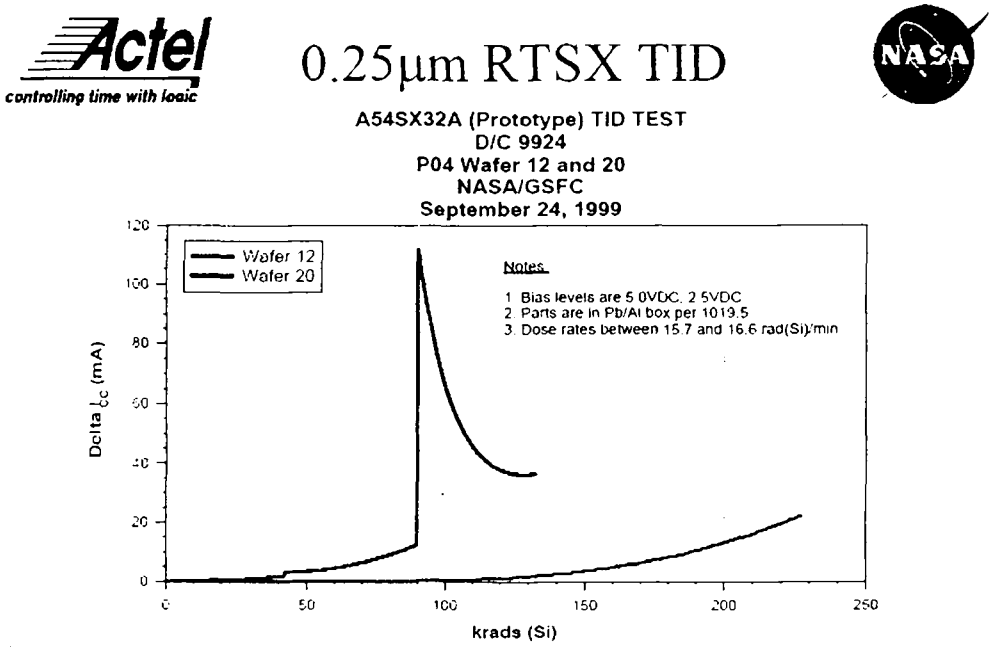


FPGA의 Cell들은 아래와 같이 기본적으로 R-Cell(Register Cell), C-Cell(Combinatorial Cell)이 Cluster 형태로 묶여져 있는 구조를 가지고 있다.



TID 특성에 대해서 실험한 결과는 아래와 같다.

A54SX72A보다 Gate 수만 작은 모델인 A54SX32A의 TID 특성을 실험한 결과는 아래와 같다.



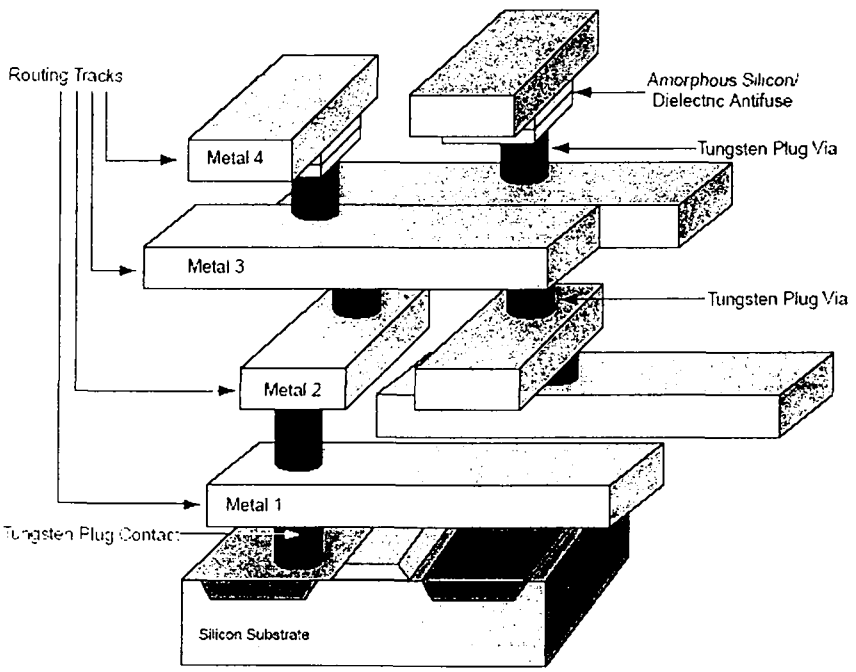
HiRel A54SX72A FPGA보다 좀 더 개선된 Radiation Tolerant Type으로 OBC에 사용하기에 적당한 FPGA는 RT54SX72S가 있으며 아래와 같은 Spec을 가진다.

RT54SX-S Product Profile

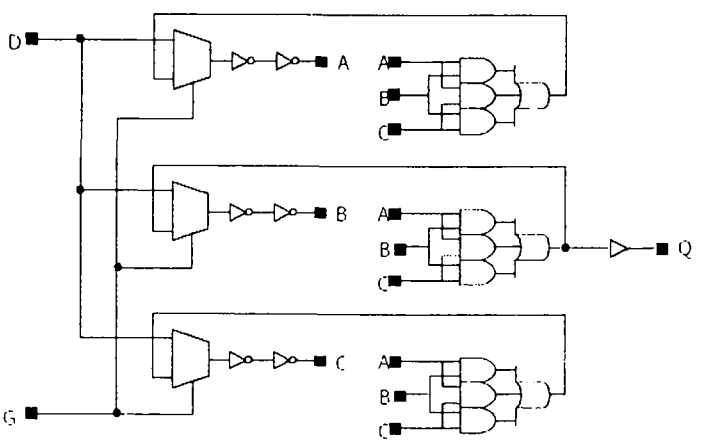
Device	RT54SX32S	RT54SX72S
Capacity		
Typical Gates	32,000	72,000
System Gates	48,000	108,000
Logic Modules	2,880	6,036
Combinatorial Cells	1,800	4,024
SEU Hardened Register Cells (Dedicated Flip-Flops)	1,080	2,012
Maximum Flip-Flops	1,980	4,024
Maximum User I/Os	227	212
Clocks	3	3
Quadrant Clocks	0	4
Clock-to-Out Delay	8.7 ns	11.0 ns
Input Set-Up Time (External)	-1.3 ns	-3.3 ns
Speed Grades	Std. -1	Std. -1
Package (by pin count)		
CQFP	208, 256	208, 256
CCGA		624

Radiation Tolerant Type의 FPGA인 RT54SX72S FPGA는 아래와 같이 cell의 구조가 3중화가 되어 있어서 내부적으로 TMR 기능을 수행한다.

내부의 구조는 HiRel A54SX72A와 같이 4층의 Metal Layer를 이루면서 Antifuse 방식의 구조이다.



SEU(Single Event Upset) Hardened Register Cell의 구조로서 아래의 그림과 같다.



NASA에서 내방사선 성능 실험을 한 결과는 아래와 같다.

SEE Test Results - RT54SX72S

- SEU
  - One error per run (total fluence=10^7 ions/cm^2) sometimes for LET > 50 MeV-cm^2/mg.
- SEL
  - Immune up to 110 MeV-cm2/mg (V_{CCI}/V_{CCA}=5.5/2.75V).
- SEDR
  - For Iodine (60 MeV-cm^2/mg) at normal incidence (which is the worst case), none at V_{CCA}=2.75V, 1 or 2 antifuse rupture at V_{CCA}=2.85V.
  - 1 rupture generates ~10mA permanent I_{CC} increase.
  - DUT were functional after multiple ruptures.
  - Using CRÈME96 model (100 mil Al shielding, Solar minimum), rupture rate < 3.94×10^{-9} ruptures/device/year (1 rupture per device per 250 million years).

3 SRAM 선정

가 인공위성 OBC 용 메모리

메모리는 컴퓨터 시스템의 제일 중요한 작업공간이다. 메모리는 CPU나 마이크로프로세서에 연결되어 데이터나 프로그램, 혹은 CPU에 즉시 접속이 가능한 처리된 정보를 저장하거나 다른 시스템 장치에 저장하는 역할을 한다. 메모리는 컴퓨터 작동의 핵심으로써 소프트웨어와 CPU의 사이를 연결시켜주는 중요한 역할을 한다. 또한 컴퓨터 메모리는 동시에 작동하는 프로그램의 수와 크기를 결정하며 매우 큰 역할을 하는 마이크로프로세서의 사용을 최적화 시켜준다.

(1) 위성에 적합한 메모리 종류

반도체 기술의 발달로 메모리 종류도 SRAM(Static RAM), DRAM(Dynamic RAM), FPRAM(Fast-Page RAM), EDORAM(Extended Data Output RAM), SDRAM(Syncronous DRAM), EDRAM(Enhanced DRAM), VRAM(Video RAM), WRAM(Window RAM), SGRAM(Synchronous Graphic RAM), RDRAM(Rambus DRAM)등 다양하다. 그중에서 SRAM 은 DRAM에 비해서 5배 더 빠르며, 가격은 2배 더 높고 크기도 2배이고, 계속적으로 리프레쉬 작업을 할 필요가 없이 DRAM

보다 안정적이다. 그러므로 안전성이 필요한 군사. 위성용 장비나 빠른 속도가 필요한 CPU내의2차 캐시 메모리에 주로 사용된다. 이러한 SRAM은 예전에는 동기식/비동기식을 사용했지만 최근에는 PBSRAM(Pipeline Burst SRAM)이라는 빠른 SRAM도 사용된다.

(2) 위성에 적합한 메모리 Package

본 프로젝트에서 사용할 프로그램 메모리 크기는 2MB 정도이다. 이는 2MB용량을 갖는 메모리 칩 하나를 사용하면 이상적이라고 할 수 있다. 하지만, 2MB 용량을 갖는 Military 등급이상의 SRAM을 찾을 수 없었으며, 이를 해결하기 위한 방법으로 작은 메모리를 여러 개를 직접 보드에 사용하는 방법과 여러 개의 칩을 Package 하여 사용하는 방법이 있다.

이를 위하여 여러 업체의 메모리 업체의 정보와 제품을 찾아보았으며, Industrial Spec 이상의 SRAM 칩 생산업체는 와 제품을 찾아보았다. Cypress,Performance,사에서 생산되는 Military SRAM 칩은 512KB로 2MB로 구현하려면 4개가 필요하며, Performance 사에서 생산되는 것은 128KB 로 16개의 칩을 필요로 하며, IDT 사에서 생산되는 칩은 32MB로 칩이 64개 필요로 하며, AeroSpace사는 비용이 비싸다. 또한 Military 이상의 메모리 팩을 생산하는 업체는 Dense Pack 으로 Dense Pac사는 2MB를 생산하고 있으며, 칩을 스택형식으로 쌓아서 만든다. 이러한 칩들 중에서 본 과제에서는 제작비용과 TMR이라는 방식의 구현에 의한 SEU의 안정도를 감안하여 보드 제작시 국내의 SRAM을 제작하는 삼성사의 메모리를 이용하기로 결정하였다.

다음 표는 현재까지 대부분의 저궤도 인공위성에서 OBC용으로 사용한 메모리를 정리하였다.

	사용 메모리	사이즈	제조사
우리별 1,2호	MSM832TL-10	SRAM 32k*8	Hybrid
	MSM8128VL-10	SARM 128k*8	Hybrid
	MS81000RKXL-10	Hybrid 1M	Hybrid
우리별 3호	SYS8512FKXL1-85 1		
	M5M5100913VP 70LL		
과학기술위성 1호	WS512K32N-70H2M	512k*32	White

3 절 소형화 OBC 보드 개발

1 보드 개발

가 8051을 이용한 데모 보드 개발

(1) System Overview

Dual CPU Test 및 TMR Memory 기능 테스트를 위한 1차 보드 제작으로서 비교적 간단한 구조의 8비트 마이크로 프로세서인 Intel 8051 CPU를 이용한 기능 검증 보드를 개발하였다. 또한 SCC와 CAN 및 기타 로직을 FPGA로 구현하고 Test하기 쉽게 ISP(In System Programming)용 Actel FPGA를 사용하였다.

FPGA로 개발하는 SCC 및 CAN을 Board 내에서 Test 할 수 있게 상용 SCC 및 CAN Controller를 부착한 테스트 시스템도 같은 Board내에 내장되어 FPGA Code 검증이 용이하다.

먼저 CPU 부분을 보면 3개의 Intel 8051 Chip이 사용되었는데 하나의 CPU는 System의 Data Bus에 직접 Access 하며 각 종 Control 신호를 내보내는 Primary CPU로 사용이 되고 나머지 2개의 CPU는 똑같이 Monitoring CPU로서 동작을 하게 된다. Primary CPU와 Monitoring CPU 2개의 Data 신호들이 FPGA에 모두 연결이 되어 있으므로 FPGA는 세 개의 CPU가 똑같이 정상 동작하는지를 확인할 수 있다. 우주 환경에서 우주방사능 등 여러 원인에 의해서 CPU가 오동작하여 특정 CPU의 특정신호 값이 다른 CPU의 신호 값과 달라졌을 경우에 FPGA가 이를 감지하여 특별한 조치를 취할 수 있다. 즉 우주환경 하에서 고 신뢰도의 시스템을 구축하기 위하여 Primary CPU외에 Monitoring CPU를 두어서 시스템 안정성을 더하는 구조이다.

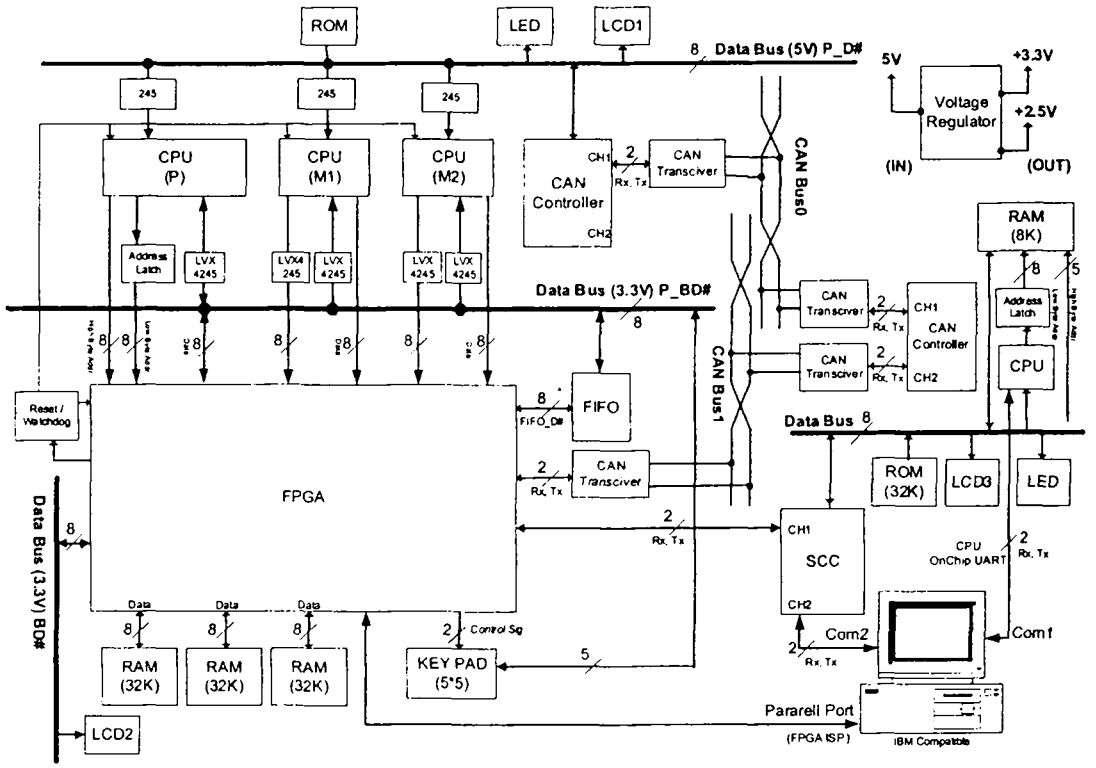
Watchdog Logic의 경우에는 FPGA가 CPU의 상태를 계속 Monitoring하여 CPU를 Reset 시켜야 할 시점을 파악하여 Primary CPU 및 Monitoring CPU 또한 FPGA를 동시에 Reset 걸 수 있다.

TMR 메모리의 경우를 살펴보면 하나의 Data를 3개의Memory에 동시에 쓰는 구조로 설계되어 있고 Data를 읽을 경우에는 3개의 Memory에서 동시에 읽어서 3개의 Data의 값을 Majority voting을 하여 그 값을 취하는 구조이다. 즉 우주환경에서 외란의 영향에 의해 동일한 Address의 3개의 Memory Data 값 중에 하나의 bit가 오류가 나서 값이 바뀌더라도(1->0 or 0->1) TMR Logic에 의해 3개의 bit 값 중 2개 이상이 나타내는 값을 취하고 오류가 예상되는 나머지 한 비트의 값을 반전시켜서 정정해주는 구조이다. 또한 주기적으로 3개의 Memory를 읽어서 TMR을 적용시켜서 Error가 난 bit를 정정해주는 기능을 수행하도록 Coding을 해야 한다.

OBC 소형화를 위해 SCC 및 CAN도 FPGA로 구현하도록 설계되었고 SCC Test를 위해 Board내에 위치한 Test 시스템의 상용 SCC와 연결이 되어 있고, CAN의 Test를 위해서 CAN Bus가 2개가 존재하여 하나의 CAN Bus는 상용 CAN Controller 끼리 연결이 되어 있고 다른 하나의 CAN Bus는 FPGA로 구현하는 CAN과 상용 CAN Controller가 서로 연결이 되어 두 개의 차이점을 분석하면서 CAN Code를 개발할 수 있도록 설계되었다.

VHDL Code 개발하면서 Debugging하기 쉽게 LED 및 LCD가 있으며 Keypad를 통한 입력 및 DIP Switch를 통한 입력이 가능하다.

(2) Block Diagram



Block Diagram을 보면 크게 두 부분으로 나누어지는데 왼쪽부분이 FPGA를 이용한 시스템 개발부분이고 오른쪽의 시스템이 SCC 및 CAN의 Test를 위한 Test System 부분이다.

먼저 FPGA를 이용한 시스템 개발부부터 보면 왼쪽 상단부분의 CPU(Primary)는 System 전체의 Data Bus를 Read/Write 하며 각종 Control 신호를 내보내는 역할을 하고 Monitoring CPU인 CPU(M1), CPU(M2)은 System Data Bus를 Read만 할 수 있고 Write는 할 수 없다. Primary 및 Monitoring CPU의 Data Bus 및 Control 신호는 모두 FPGA로 연결이 되어 있으며 이를 통하여 FPGA는 CPU의 상태를 감시할 수가 있다. System의 전원이 인가되면 Primary 및 Monitoring CPU는 프로그램 저장영역인 ROM에서 프로그램을 읽어서 실행을 하게 되는데 세 개의 CPU가 똑같이 실행이 되기 위해서는 회로 설계도 중요하지만 특히 PCB Artwork에서도 주의를 요하게 된다. 또한 CPU에 들어가는 System Clock도 하나의 Oscillator 출력

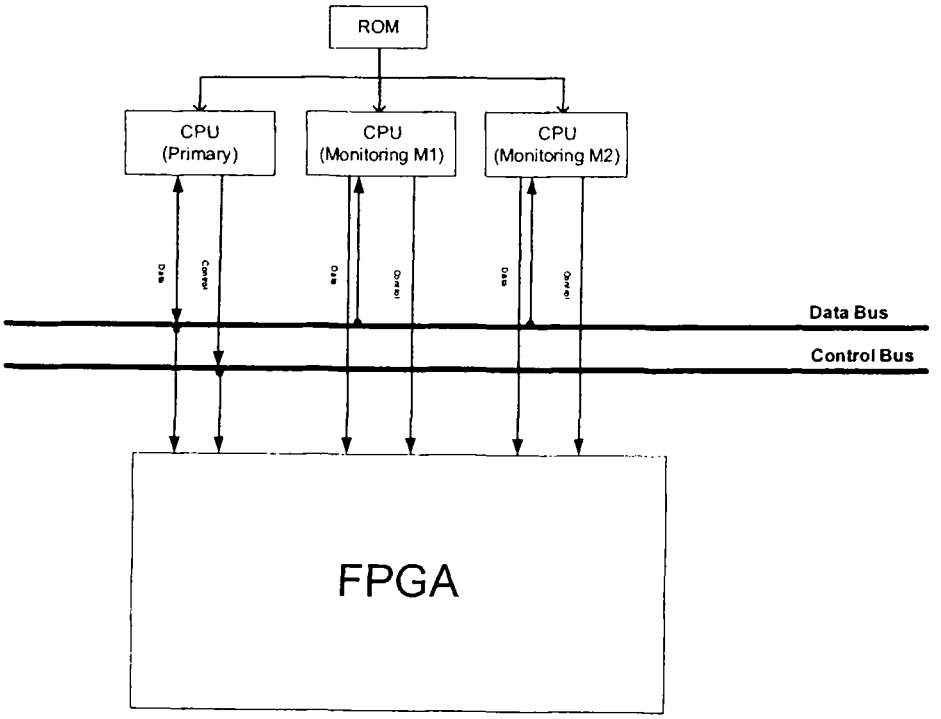
이 동시에 CPU에 인가되어야 하므로 신호가 틀어지지 않고 time delay 특성이 동일하도록 PCB Artwork의 주의를 요하게 된다.

FPGA의 밑에 부분에 보이는 외부 Memory는 TMR 구조로서 똑 같은 Size(32K)의 3개의 Memory가 있으며 동시에 쓰고 동시에 읽어서 Error 난 Bit가 발생시에는 3개의 Memory Bit값 중에서 Majority Voting Algorithm을 적용하여 Error를 Correction하게 된다.

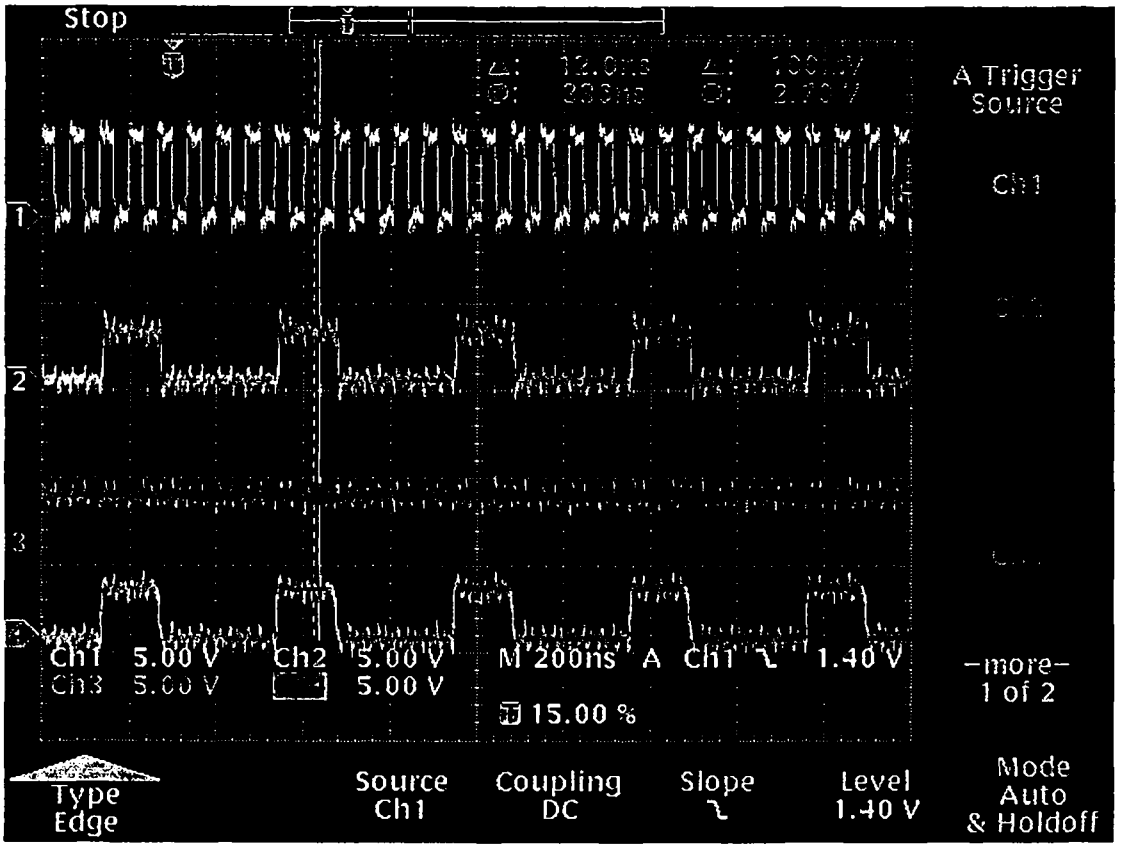
또한 FPGA가 판단하여 CPU를 Reset 시킬 필요가 있을 때에는 Watchdog Logic에 신호를 주어서 Primary CPU 및 Monitoring CPU를 동시에 Reset 시킬 수가 있으며 FPGA의 왼쪽 편에 세로로 그려져 있는 Data Bus는 각 CPU(Primary, Monitoring M1, Monitoring M2)의 Data Bus의 신호 값도 TMR을 적용시켜서 Error Correction 한 Data Bit를 내보내는 Data Bus이다. 이처럼 Test Board에서는 여러 가지 실험을 할 수 있게끔 설계가 되었으며 실험의 용이성을 위해 LED, LCD 출력, DIP Switch 입력, Keypad 입력을 줄 수가 있다.

오른쪽 편의 Test System은 FPGA로 구현한 SCC 및 CAN의 실험의 용이성을 위해 같은 Board내에 설계가 되었고 외부 ROM 32KByte, SCC, CAN, LED, LCD로 구성되어 있다.

(3) Multi CPU 구조



프로그램 메모리인 ROM에서 프로그램을 동시에 읽어서 실행이 되며 Primary CPU만이 System의 Data Bus 및 Control Bus를 장악하게 되고 Monitoring CPU M1과 M2는 Data Bus에서 읽기만 하고 쓸 수는 없다. FPGA는 3개의 CPU의 Data 신호 및 Control 신호를 Monitoring 하여 필요시에 Watchdog Reset 등을 걸 수가 있다.

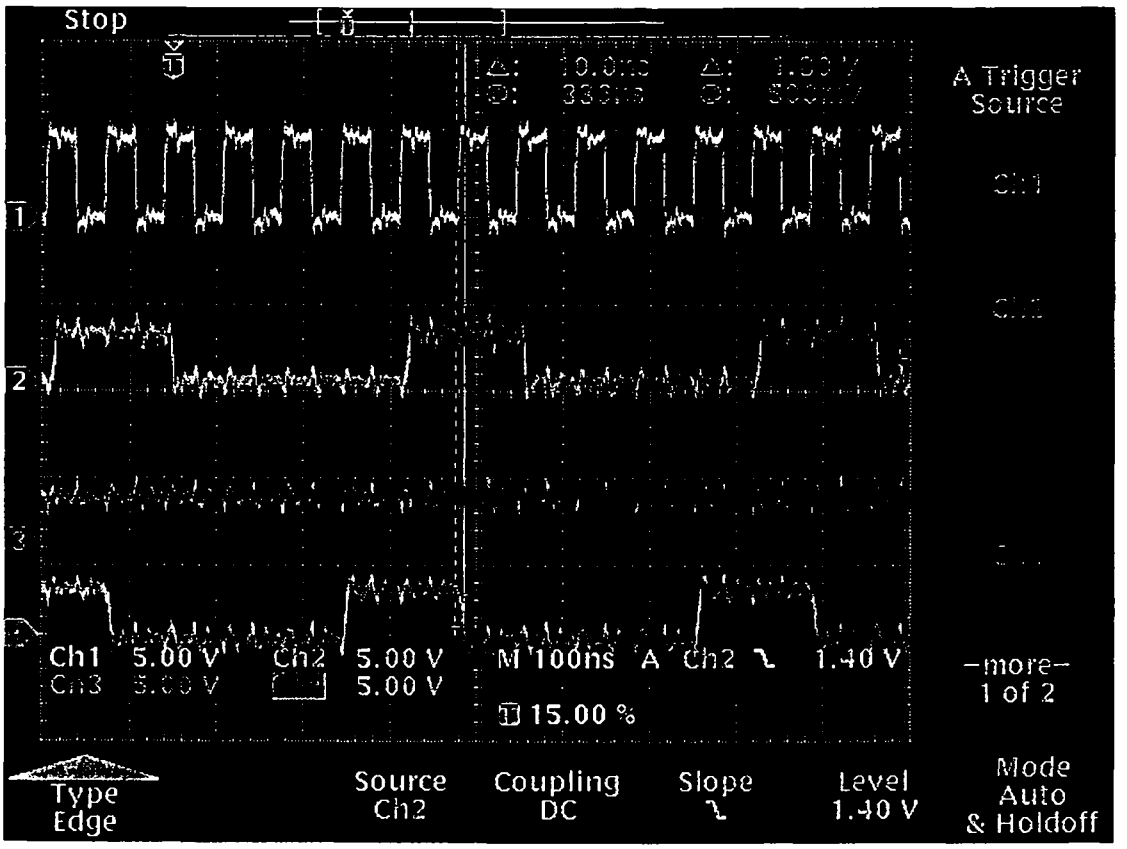


[1] CPU Clock

[2] Primary CPU의 ALE(Address Latch Enable) 신호

[4] Monitoring CPU의 ALE 신호

위의 그림에서 보면 Primary CPU의 Control 신호인 ALE와 Monitoring CPU의 ALE를 비교한 그림이다. 즉 Primary CPU와 Monitoring CPU의 Control 신호가 똑같이 나오면서 정상 동작을 하는 그림이다.



[1] CPU Clock

[2] Primary CPU의 ALE(Address Latch Enable) 신호

[4] Monitoring CPU의 ALE 신호

위의 파형에서 보면 Primary CPU의 Control 신호와 Monitoring CPU의 Control 신호가 한 Clock 차이가 나는 것을 볼 수 있다. 프로그램 메모리인 ROM에서 Code를 실행하는 시간이 서로 차이가 나는 것을 볼 수 있는데 이렇게 되면 우리가 원하는 결과를 얻을 수가 없다. Primary CPU와 Monitoring CPU가 똑같이 동작하도록 회로설계 뿐만 아니라 PCB Artwork에서도 신경을 써야 한다. 1차 실험보드에서는 100번 중에 1,2번 정도가 Primary CPU와 Monitoring CPU가 시간 차이를 두고 동작을 하였다. 그리고 CPU의 Clock을 14.7456MHz에서 동작시키는 것 보다 더 낮은 클럭인 7.3728MHz로 동작시켰을 때에는 Primary CPU와 Monitoring CPU가 시간 차이가 나지 않고 동일하게 동작을 하였다. 예상대로 CPU를 좀 더 느리게 동작시키면 보다 안정적으로 돌아간다는 것을 확인할 수 있었다. CPU를 동일한 시간에 똑같이 동작시키기 위해서는 각 CPU에 들어가는 Clock Input의 시간차이가 안 나

게 PCB Pattern을 잘 조정해야 하고 또한 각 CPU가 똑 같은 시간에 Reset이 되도록 PCB 설계에 유의해야 한다.

CPU는 Intel 8051을 사용하였으며 Spec은 아래와 같다.

8bit micro processor

32 decrete I/O pins (Port0, Port1, Port2, Port3)

Two 16 bit timer/counter

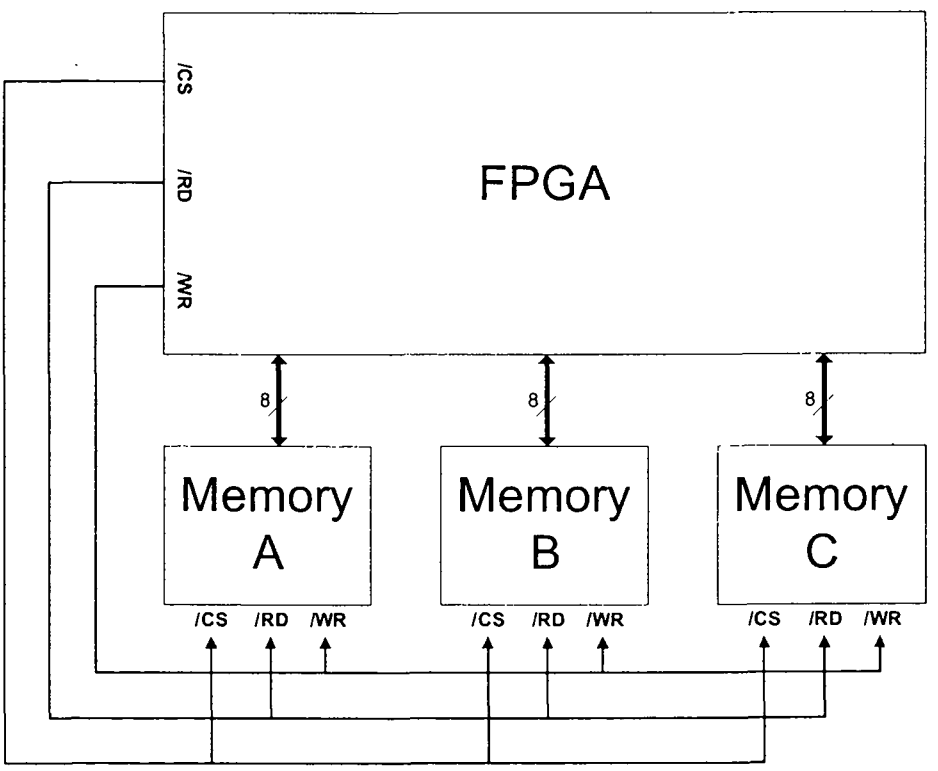
Full duplex UART

6 Interrupt Sources with 2 Priority levels

128 bytes of on board RAM

Separate 64K byte address spaces for DATA and CODE momory

(4) TMR Memory 구조



32Kbytes의 똑 같은 사이즈의 메모리를 3개 사용하며 Data를 쓸 때는 똑 같은 Address에 동일한 값을 쓰고 Data를 읽을 때에도 똑 같은 Address에서 읽어서 각각 읽은 8bit의 Data 값들을 비교하여 FPGA내에 구현한 TMR 로직에 의해 Error Correction 한 값을 사용하게 된다.

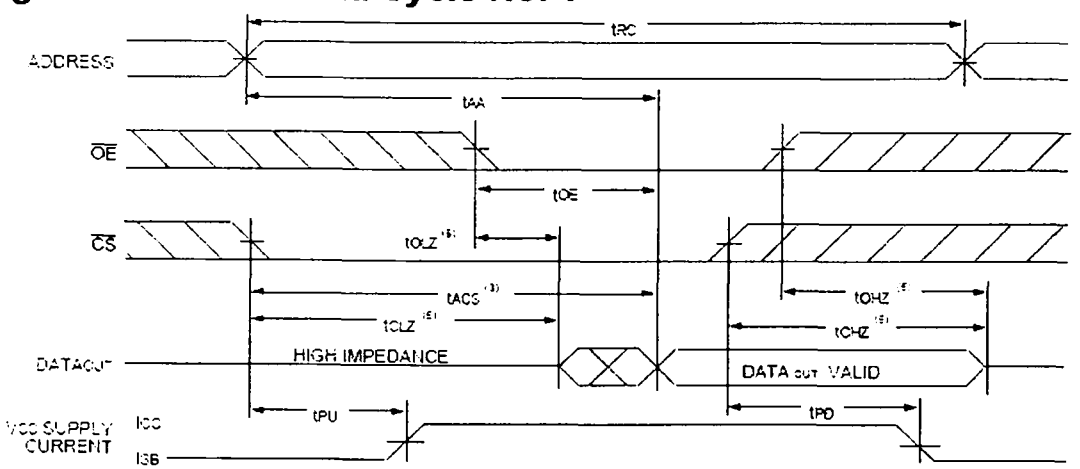
외부 Memory의 Spec은 아래와 같다.

Model : IDT71256SA

Vendor : IDT

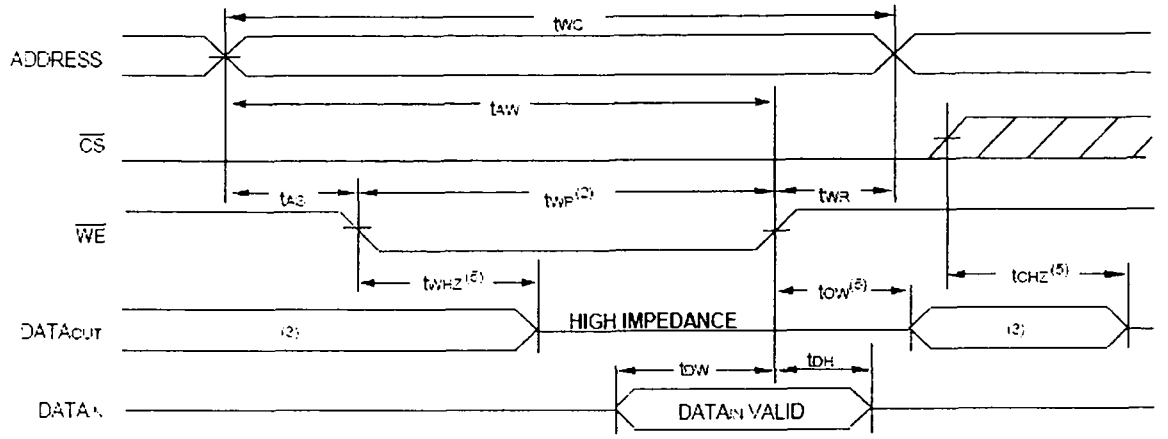
CMOS Static RAM 256K (32K*8Bit)

Timing Waveform of Read Cycle No. 1(1)



외부 메모리를 읽을 때에는 먼저 Chip Selection 신호(/CS)를 내보내고 그 다음에 Output Enable(/OE) 신호를 내면 해당되는 Address에 있는 Data의 값을 읽을 수가 있다.

Timing Waveform of Write Cycle No. 1 (WE Controlled Timing)(1,2,4)



외부 Memory에 Data를 쓸 때는 Chip Selection(/CS) 신호를 보내고 Write Enable(/WE)신호를 보내게 되면 지정한 Address에 원하는 Data 값을 쓸 수가 있다.

(5) CAN Controller

FPGA로 구현하는 CAN Logic을 Test하기 위해 Test 시스템에 상용 CAN Controller를 부착하였다. 상용 CAN Controller의 Spec은 아래와 같다.

Vendor : Philips

Model : SJA1000

CAN 2.0B Protocol Compatibility

Extended receive buffer (64byte FIFO)

Bit rates up to 1 Mbits/s

PeliCAN mode extensions

- Error counters with read/write access
- Programmable error warning limit
- Last error code register
- Error interrupt for each CAN-bus error

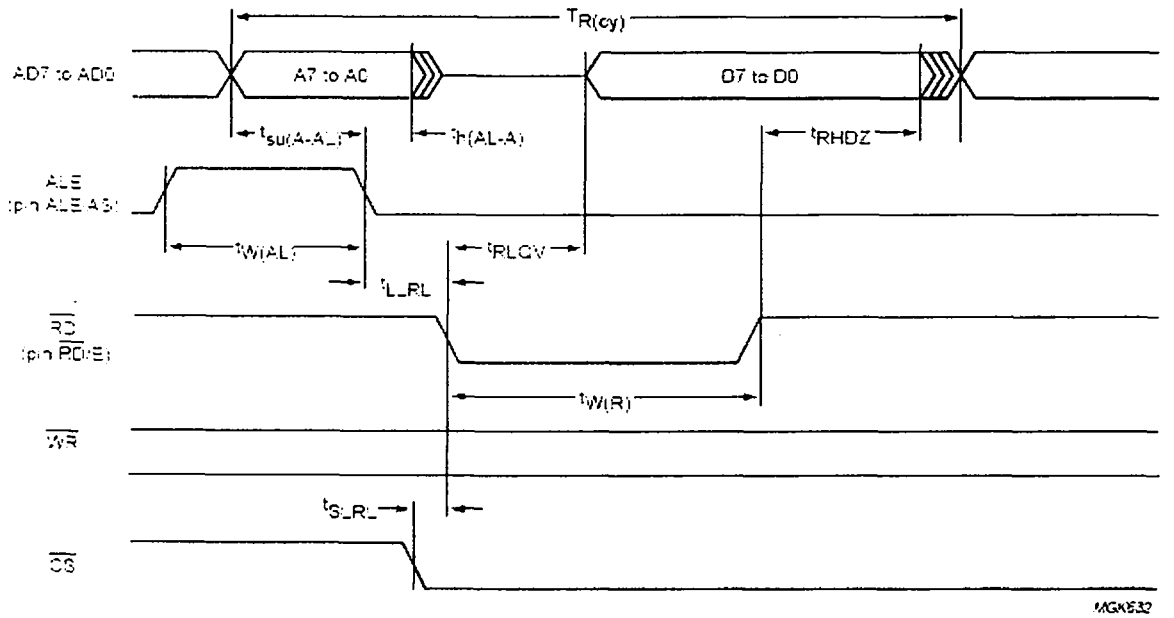
- Arbitration lost interrupt with detailed bit position
- Listen only mode (no acknowledge, no active error flags)
- Hot plugging support (software driven bit rate detection)
- Acceptance filter extension (4byte code, 4byte mask)
- Reception of own message (self reception request)

24MHz clock frequency

Interfaces to a variety of microprocessors

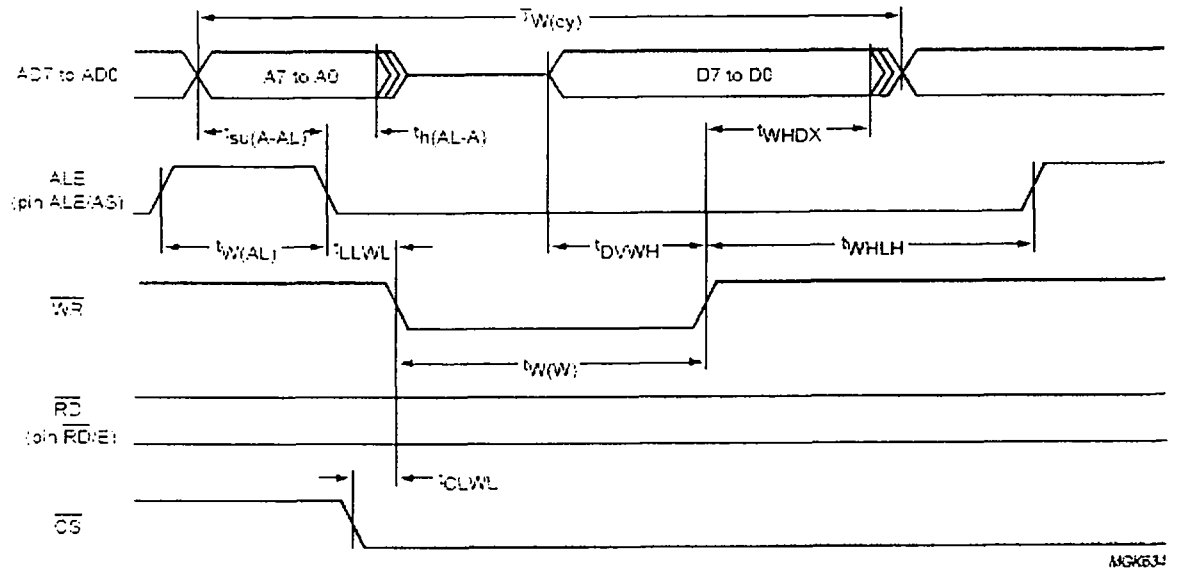
Programmable CAN output driver configuration

[Read Cycle Timing Diagram]



Read Cycle은 위와 같이 먼저 Address를 지정하고 나서 Chip Select(/CS) 신호 후에 Read Data(/RD) 신호를 보내면 지정된 Address에 저장되어 있는 Data를 읽을 수 있다.

[Write Cycle Timing Diagram]



Write Cycle도 마찬가지로 Address를 지정하고 나서 Chip Select(/CS) 후에 Write(/WR) 신호를 주면 지정된 Address에 원하는 Data 값을 쓸 수가 있다.

CAN Controller를 CAN Bus에 연결하기 위해서는 CAN Line Transceiver가 필요하며Philips사의 PCA82c250을 사용하였고 Spec은 아래와 같다.

Fully Compatible with "ISO 11898" standard

High Speed (up to 1Mbaud)

Bus lines protected against transients in an automotive environment

Slope control to reduce Radio Frequency Interface(RFI)

Different receiver with wide common mode range for high immunity against ElectroMagnetic Interface(EMI)

Thermally protected

Short circuit proof to battery and ground

Low current standby mode

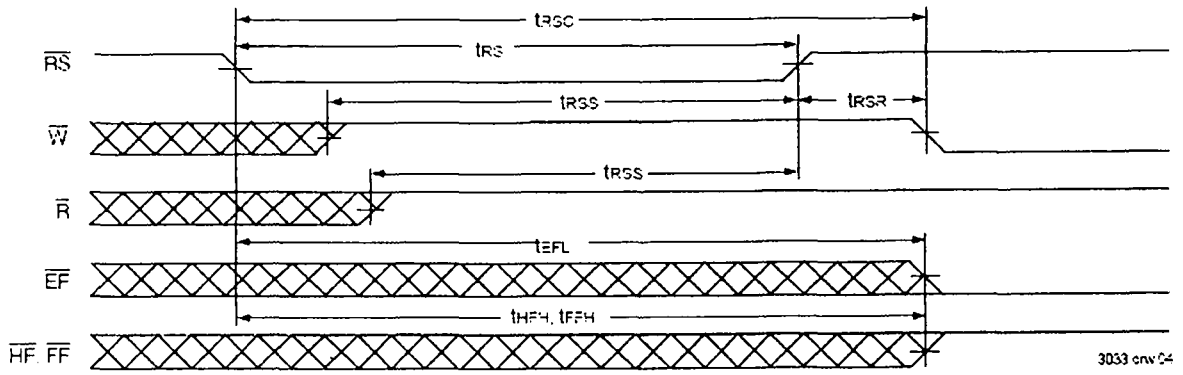
An unpowered node does not disturb the bus lines

At least 110 nodes can be connected

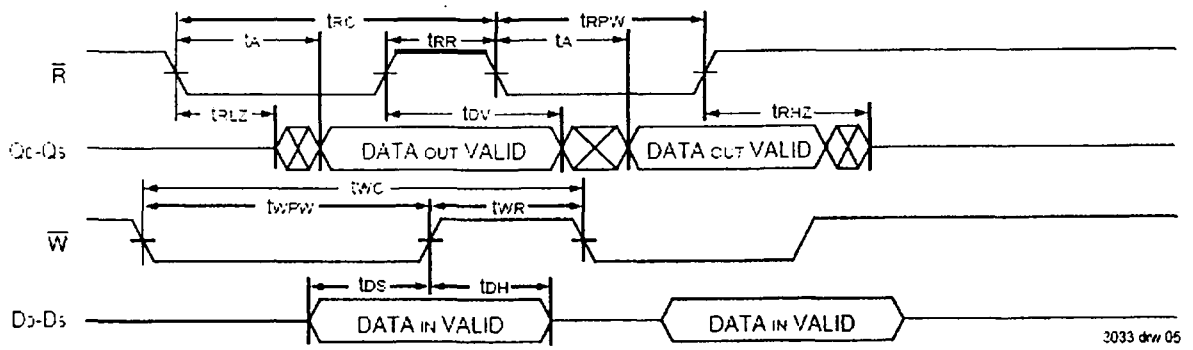
(6) FIFO

FPGA와 System Data Bus와의 효과적인 Data 전송을 위해 FIFO를 두었고 그 Spec은 아래와 같다

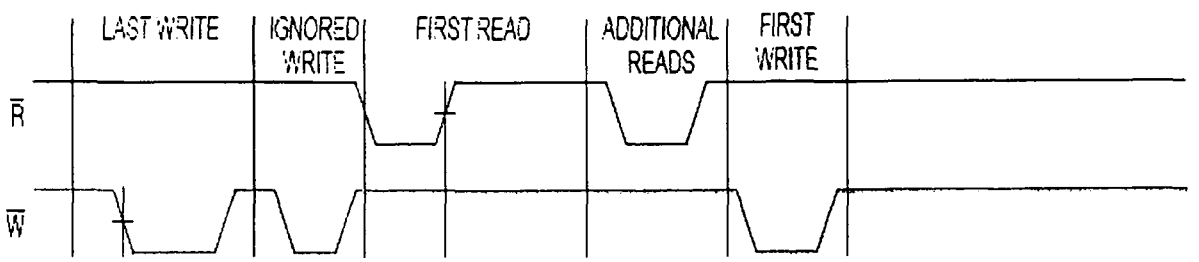
[Reset]



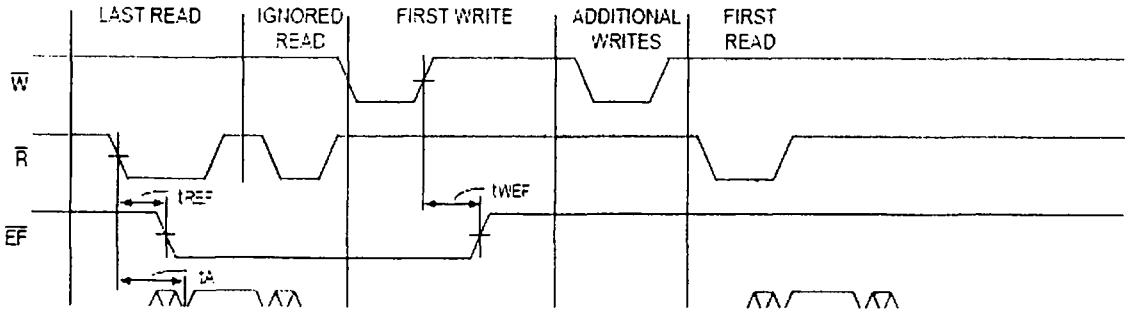
[Asynchronous Write and Read Operation]



[Full Flag from Last Write to First Read]



[Empty Flag from Last Read to First Write]



(7) LCD

Test 및 Debugging의 편리를 위해 16*2의 Text LCD를 이용하였다.

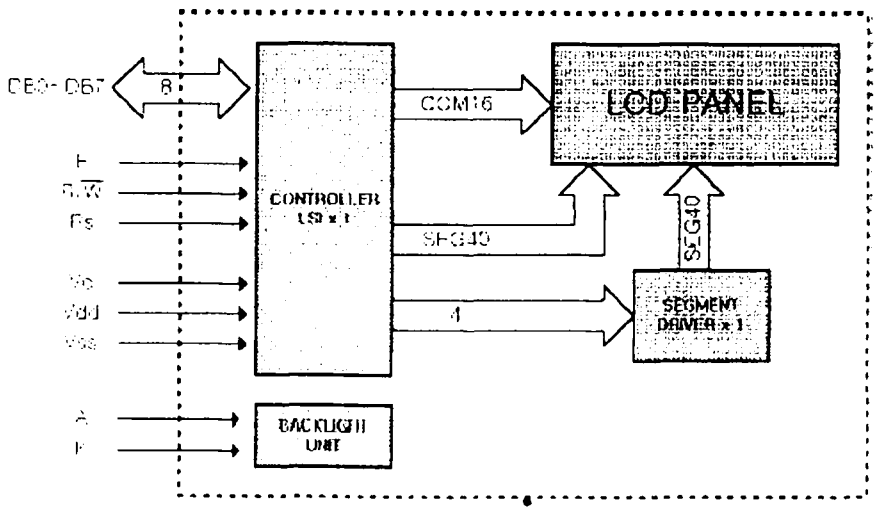
Display Format : 16 Characters * 2 Lines

Display Color : Yellow green

Input Data : 8bit parallel interface to MPU

Back Light : LED, Yellow green

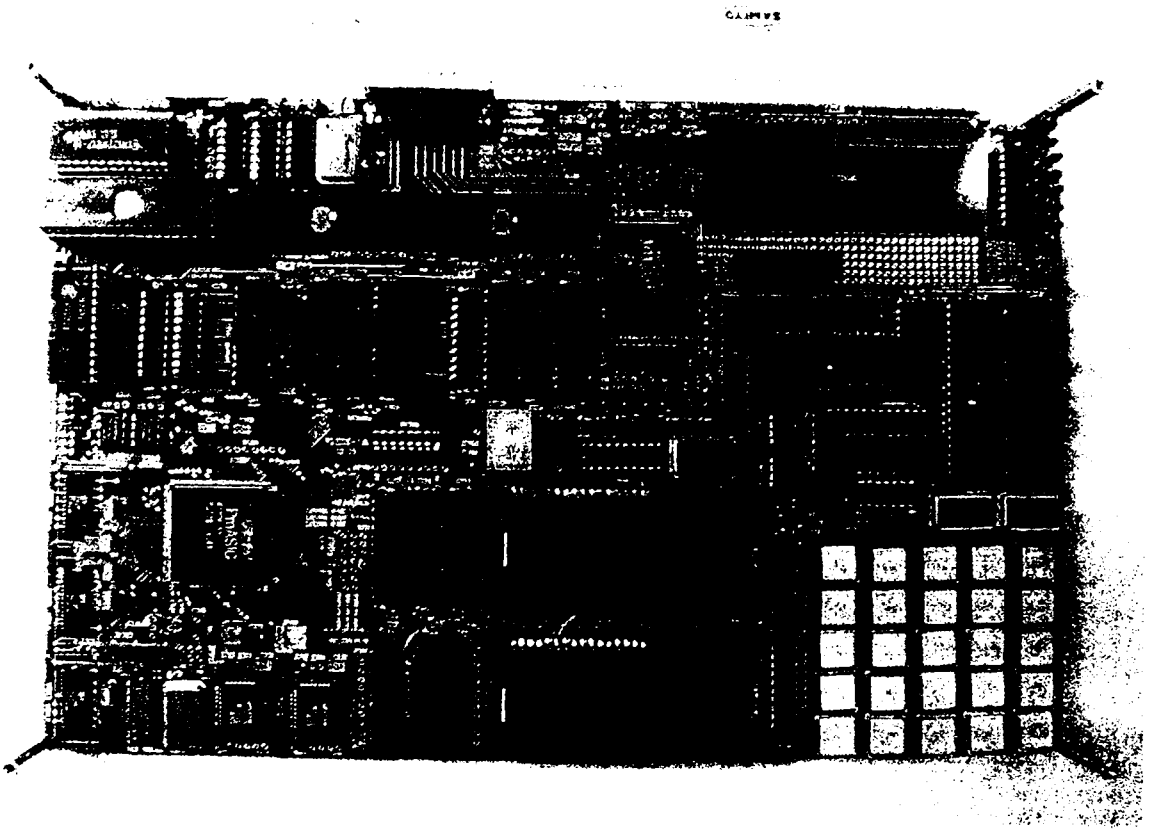
CPU와 LCD의 Interface는 아래와 같다.



Pin Configuraion은 아래와 같다.

Pin No.	Symbol	Level	Function
1	Vss	0V	Ground
2	Vdd	-5V	Logic Supply Voltage
3	Vo		LCD Driving Voltage, Operating Voltage
4	Rs	H/L	Register Select : H - Data, L - Instruction
5	R/$\overline{W}$	H/L	H - Read(LCD → MPU), L - Write(MPU → LCD)
6	E	H, H/L	Enable for read/write operation
7	DB0	H/L	Data Bit 0 (LSB)
8	DB1	H/L	Data Bit 1
9	DB2	H/L	Data Bit 2
10	DB3	H/L	Data Bit 3
11	DB4	H/L	Data Bit 4
12	DB5	H/L	Data Bit 5
13	DB6	H/L	Data Bit 6
14	DB7	H/L	Data Bit 7 (MSB)
15	A(EL+)	-	LED(+) or EL Backlight Connection
16	K(EL-)		LED(-) or EL Backlight Connection

(8) Board 사진



왼쪽 상단부분에 Intel 8051 CPU가 세 개가 보이며(Primary CPU, Monitoring CPU(M1), Monitoring CPU(M2)) 가운데 CPU 바로 위에 각 CPU에 Clock을 공급해주는 Oscillator가 보인다. 왼쪽 가운데 부분에 ISP(In System Programming) 가능한 Actel FPGA가 있으며 그 아래로 FIFO가 있다. Debugging의 편리를 위해 Keypad 및 LCD, LED가 있고 FPGA로 구현하는 CAN 및 SCC의 Test를 위해 오른쪽 Keypad 윗부분에 Intel 8051 CPU와 상용 SCC인 Z85c30(Zilog)과 상용 CAN Controller인 SJA1000(Philips)을 부착한 Test System이 내장되어 있다.

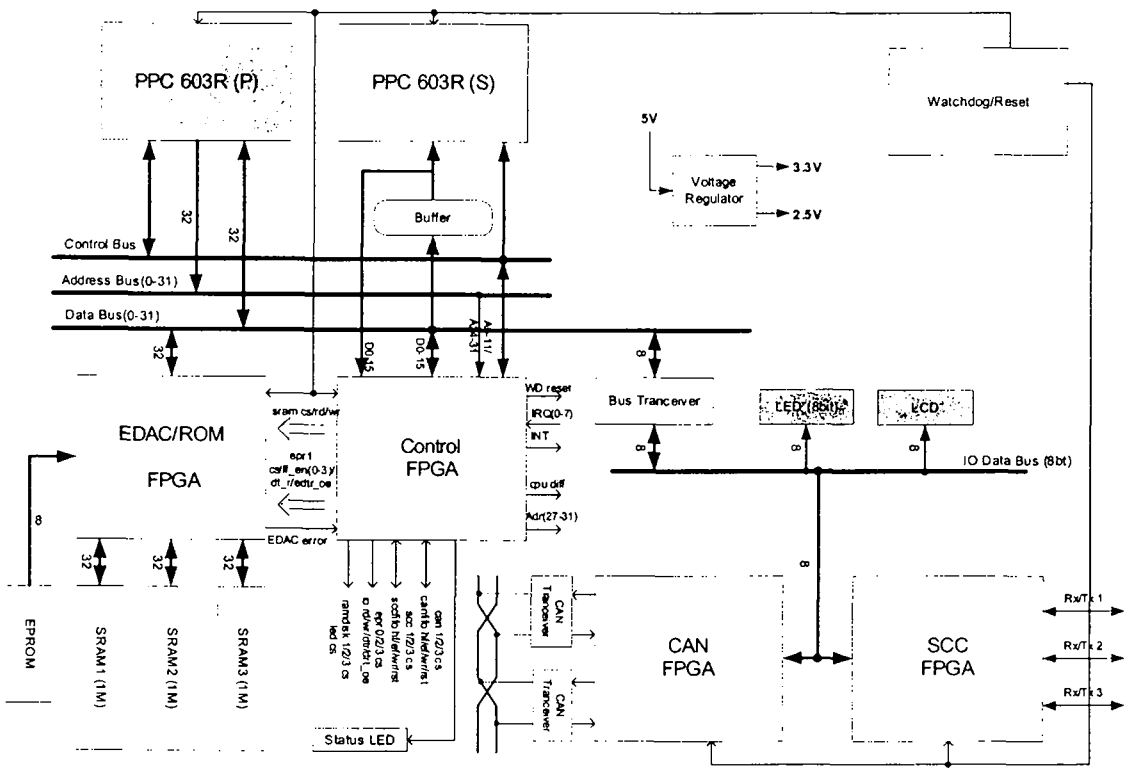
나 PowerPC 603E를 이용한 데모 보드 개발

(1) System Overview

1차 Board에서 Intel 8051 CPU를 이용하여 각종 실험을 통해 Multi CPU 및 TMR Memory 구조의 구현을 확인해 보았다. 2차 Board에서는 Power PC MPC603e를 사용한 우주용 On-Board Computer의 제작을 하였다. 1차 Board에서는 여러 가지 Test를 위해 Primary CPU 하나와 Monitoring CPU를 2개 두었었는데 2차 Board에서는 Primary CPU 하나와 Monitoring 기능을 하는 Secondary CPU 하나를 두었고 FPGA는 두 개의 CPU가 똑같이 동작하는지 감시를 하게 된다. 우주 환경 하에서 SEU와 같은 현상이 발생하여 두 개의 CPU가 똑같이 동작을 하지 않으면 Watchdog Reset 기능 등을 이용하여 시스템을 안정적으로 운용할 수 있다. 외부 메모리는 TMR 구조의 각각 1M Bytes의 SRAM을 3개 사용하였고 시스템 Main Data Bus와 I/O Bus는 구조적으로 분리시켰으며 I/O BUS에서는 LED 및 LCD등의 Debugging용 Device가 구현되어 있다. System 내에 FPGA는 총 4개가 사용이 되었으며 FPGA1은 EDAC/ROM FPGA로서 Primary CPU의 Data Bus와 연결이 되고 TMR Memory 구현을 위해 Memory Control 신호 및 3개 Memory의 Data 라인이 연결되어 있으며 프로그램 메모리가 저장되어 있는 EPROM과 연결이 되어 있다. FPGA2는 Control FPGA로서 각종 Control 신호를 내보내는 역할을 한다. TMR 기능을 사용할 지 안 할지 선택하는 신호, CPU 동작에 필요한 Control 신호, CAN과 SCC의 동작에 필요한 Control 신호, Watchdog 신호 및 LED, LCD Control 하는 데 필요한 신호, 그리고 Primary CPU Data Bus와 Secondary CPU Data Bus의 신호를 CPU가 Write하는 시점에 비교를 하여 두 개의 CPU가 똑같이 동작하는

지 비교하는 역할을 수행한다. FPGA3는 CAN을 구현하기 위한 FPGA로서 3개의 CAN을 구현할 수 있게 핀이 할당되어 있다. FPGA4는 SCC Coding용 FPGA로서 총 3개의 SCC가 구현 가능하게 핀이 할당되어 있다. FPGA3와 FPGA4의 CAN 및 SCC는 모두 I/O Bus에 물려 있어서 시스템의 Main Bus와 분리되어 있다. 또 FPGA1, FPGA2, FPGA3, FPGA4의 여유 IO 핀에 LED 및 DIP SW Input이 연결이 되어서 VHDL Coding한 후 Test에 용이하도록 하였다. System 전체의 전원부를 살펴보면 +5V, +3.3V, +2.5V가 사용이 되는데 CPU 및 FPGA의 Core 전원을 위해서 +2.5V가 사용되며 CPU와 FPGA의 IO 전원 및 기타 IC 전원을 위해 +3.3V가 사용되고 외부 ROM 및 LCD 등 기타 Device의 전원에 5V가 사용이 된다.

(2) System Block Diagram

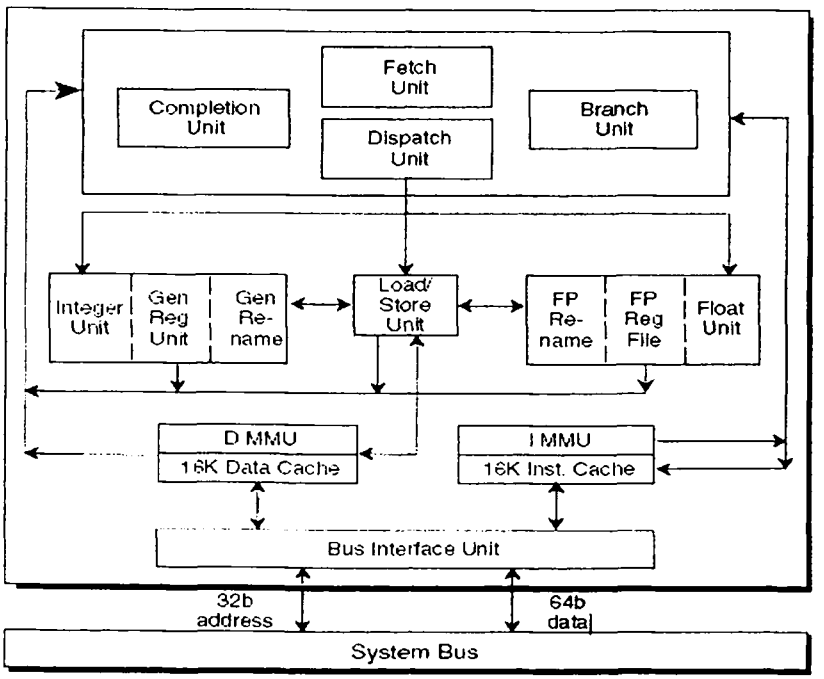


왼쪽 상단에 전체 System의 Data Bus 및 Control Signal을 관장하는 Primary CPU가 있고 또한 Secondary CPU가 존재하여 EDAC/ROM FPGA가 두 CPU의 Data Bus 및 Control Signal을 비교하여 똑같이 작동하는지 Monitoring 하게 된다. Secondary CPU는 전체 System Data Bus를 읽기만 하고 Control Signal Output

및 Data Bus Output은 FPGA로 연결이 되어 Primary CPU의 Data Bus 및 Control 신호와 비교할 수 있다. FPGA1은 EDAC/ROM FPGA로서 1M Bytes의 SRAM 3개와 연결이 되어 있어서 FPGA1 내부에 구현되는 TMR 로직에 의해 EDAC(Error Detection and Correction) 처리를 하고 EPROM1에 의해 프로그램이 실행이 된다. 또한 FPGA2(Control FPGA)에 의해 FPGA1이 TMR기능의 On/Off를 Control 할 수 있어서 성능의 비교 분석이 가능하다. FPGA2는 전체System의 모든 Control 신호를 관장한다고 보면 된다. CPU의 동작에 필요한 각종 Control 신호 및 FPGA3에 구현되는 CAN의 동작에 필요한 Control 신호, FPGA4에 구현되는 SCC의 동작에 필요한 Control 신호를 만들어 낸다. 또한 필요시에 Watchdog Reset 신호를 내보내게 되면 CPU 및 FPGA가 모두 Reset이 되게 된다. FPGA3와 FPGA4에는 각각 CAN 및 SCC를 코딩하여 구현하고 Data 입출력 Bus는 시스템의 Main Data Bus와 분리된 IO Data Bus와 연결이 되어있다. 또한 VHDL로 개발하는 FPGA Code Test의 편리를 위해 LED 및 LCD가 부착되어 있고 Test Input을 주기 위해 DIP SW Input이 가능하다.

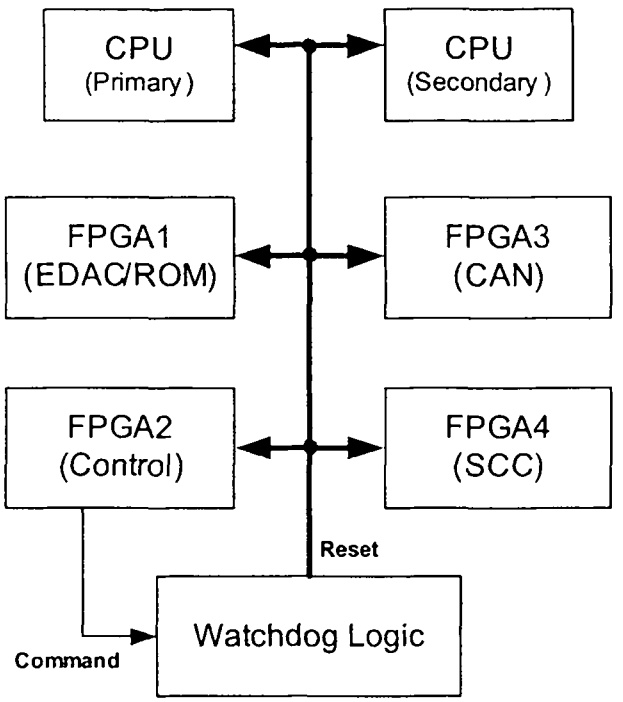
(3) CPU

Power PC 603e의 내부Block Diagram은 아래와 같다.



(4) Reset/Watchdog 부분

System에 전원이 인가되었을 때 Power On Reset 기능을 수행하며 System의 Control Signal을 관장하는 FPGA에서 Reset 명령이 떨어지게 되면 Primary CPU 및 Secondary, 또한 모는 FPGA를 Reset 하는 역할을 수행한다.



(5) TMR Memory 부분

TMR Logic은Error를 Detection하고 Correction하는 데 있어서 3개의 Data 값 중에서 Data 값이 일치하는 두 개 이상의 Data 값을 선택하는 방식이다. 즉 3개의 Data 값이 같으면 그 값을 취하고 Data 값이 두 개는 같고 하나는 틀린 경우에는 두 개가 같은 쪽의 Data 값을 취한다. VHDL Code로 보면 아래와 같다.

```
architecture BEHAVIOR OF TMR_EDAC IS
begin
    OUT_TMR <=  '1' when IN1 = '0' and IN2 = '1' and IN3 = '1' else
                '1' when IN1 = '1' and IN2 = '0' and IN3 = '1' else
                '1' when IN1 = '1' and IN2 = '1' and IN3 = '0' else
                '1' when IN1 = '1' and IN2 = '1' and IN3 = '1' else
                '0';
end BEHAVIOR;
```

그리고 On-Board Computer는 일정 주기로 외부 Memory 내용을 읽어서 TMR 처리를 하여 세 개의 Bit 중 하나의 Bit가 바뀐 부분이 있으면 다시 수정해 주는 Memory Scrubbing 기능을 수행하도록 하여야 한다. 이러한 Memory Scrubbing 주기를 너무 자주 하게 되면 Load가 많이 걸려서 다른 일을 수행하는데 지장을 초래할 수가 있고 반대로 너무 느리게 하게 되면 우주환경에서 발생될 수 있는 Single Event Upset과 같이 Memory의 특정 Bit가 반전될 수 있는 현상으로 위성 System에 큰 영향을 끼칠 수가 있다.

(6) FPGA 선택

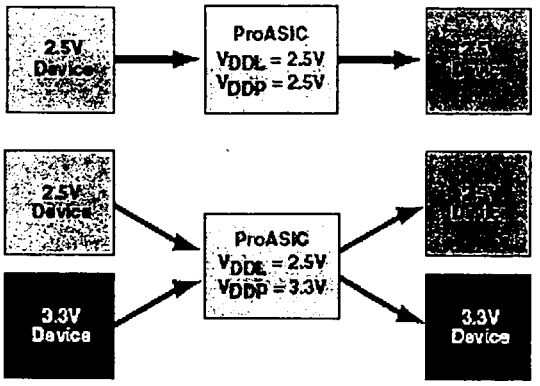
Board에 들어가는 FPGA의 선택은 Test의 용이를 위해 ISP(In System Programming) 기능이 되는 Actel사의 Flash Type FPGA를 사용을 한다.
Spec은 다음과 같다.

Internal System Performance는 250MHz이고 External System Performance는 100MHz이다. 아래의 Actel Flash FPGA Family 중에서 IO수와 Gate수를 고려하여 A500K130을 사용하였다.

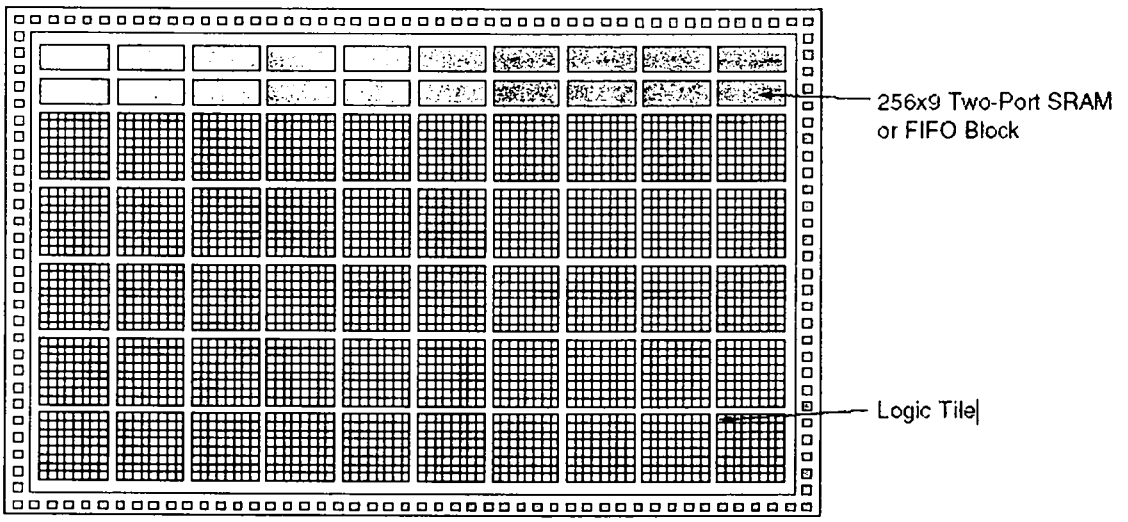
ProASIC Product Profile

Device	A500K050	A500K130	A500K180	A500K270
Maximum System Gates	100,000	290,000	370,000	475,000
Typical Gates	43,000	105,000	150,000	215,000
Maximum Flip-Flops	5,376	12,800	18,432	26,880
Embedded RAM Bits	14k	45k	54k	63k
Embedded RAM Blocks (256 X 9)	6	20	24	28
Logic Tiles	5,376	12,800	18,432	26,880
Global Routing Resources	4	4	4	4
Maximum User I/Os	204	306	362	440
JTAG	Yes	Yes	Yes	Yes
PCI	Yes	Yes	Yes	Yes
Package (by Pin Count)				
PQFP	208	208	208	208
FBGA	272	272, 456	456	456
FBGA	144	144, 256	256	256, 676

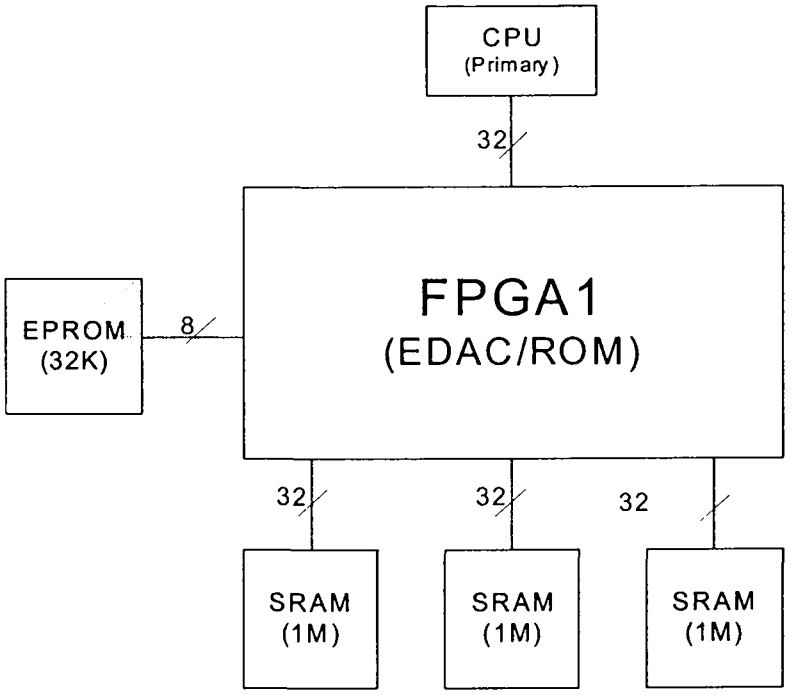
FPGA의 IO Interface는 2.5V와 3.3V 모두 지원을 한다.



ProASIC 내부의 Logic을 보면 아래와 같고 내부의 SRAM과 FIFO Block도 가지고 있다.

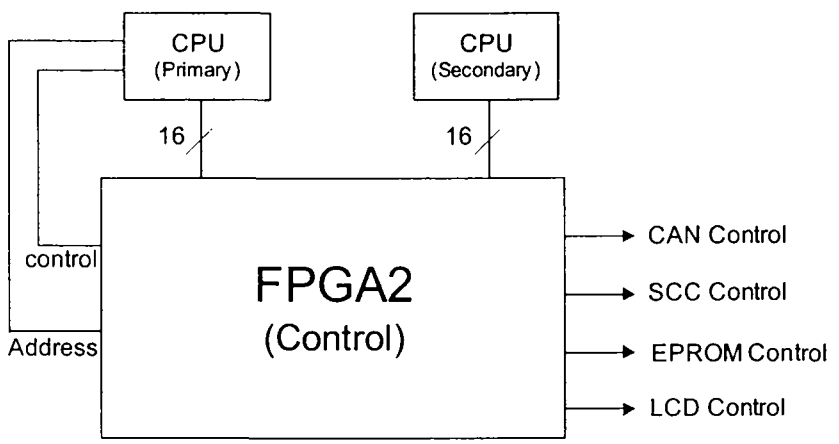


(가) FPGA1 (EDAC/ROM FPGA)



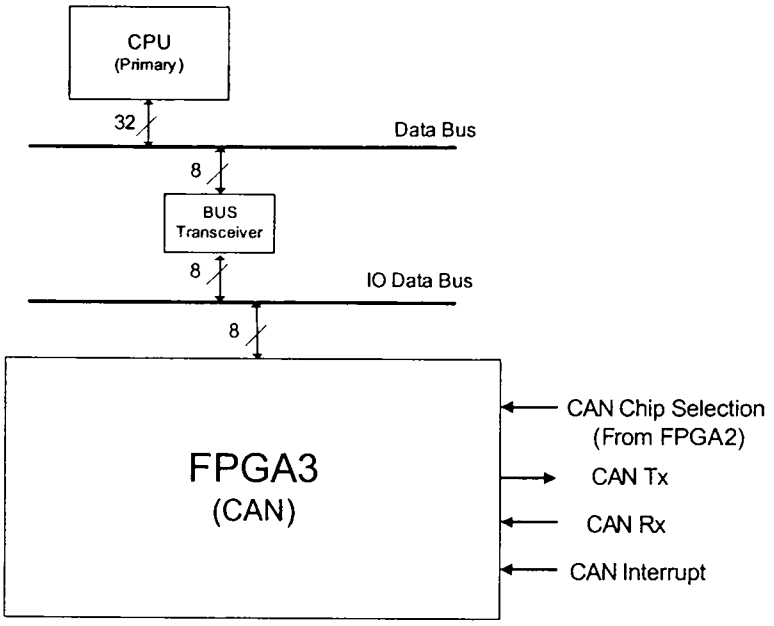
FPGA1은 Primary CPU의 32bit Data 신호와 연결되어 있고 프로그램이 저장된 32KBytes의 ROM 및 TMR 구조의 각각 1Mbyte의 외부 Memory와 연결이 되어있다.

(나) FPGA2 (Control FPGA)



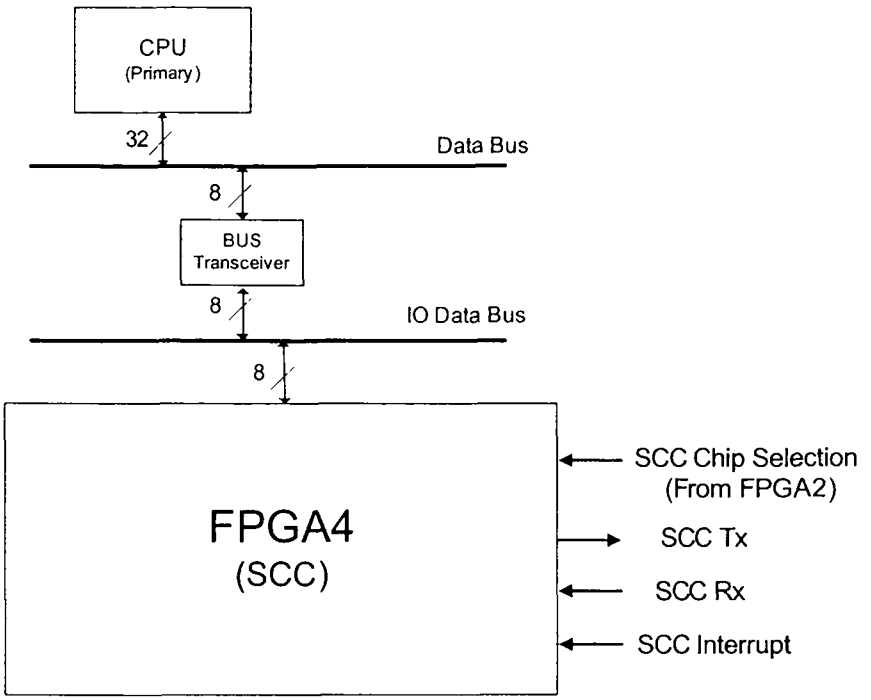
FPGA2는 System의 동작에 필요한 모든 Control 신호를 만들어 내는 FPGA이고 또한 Primary CPU Data와 Secondary CPU Data의 16bit 부분만을 비교 검색하여 두 개의 CPU가 동일하게 동작하는지 확인을 한다.

(다) FPGA3 (CAN FPGA)

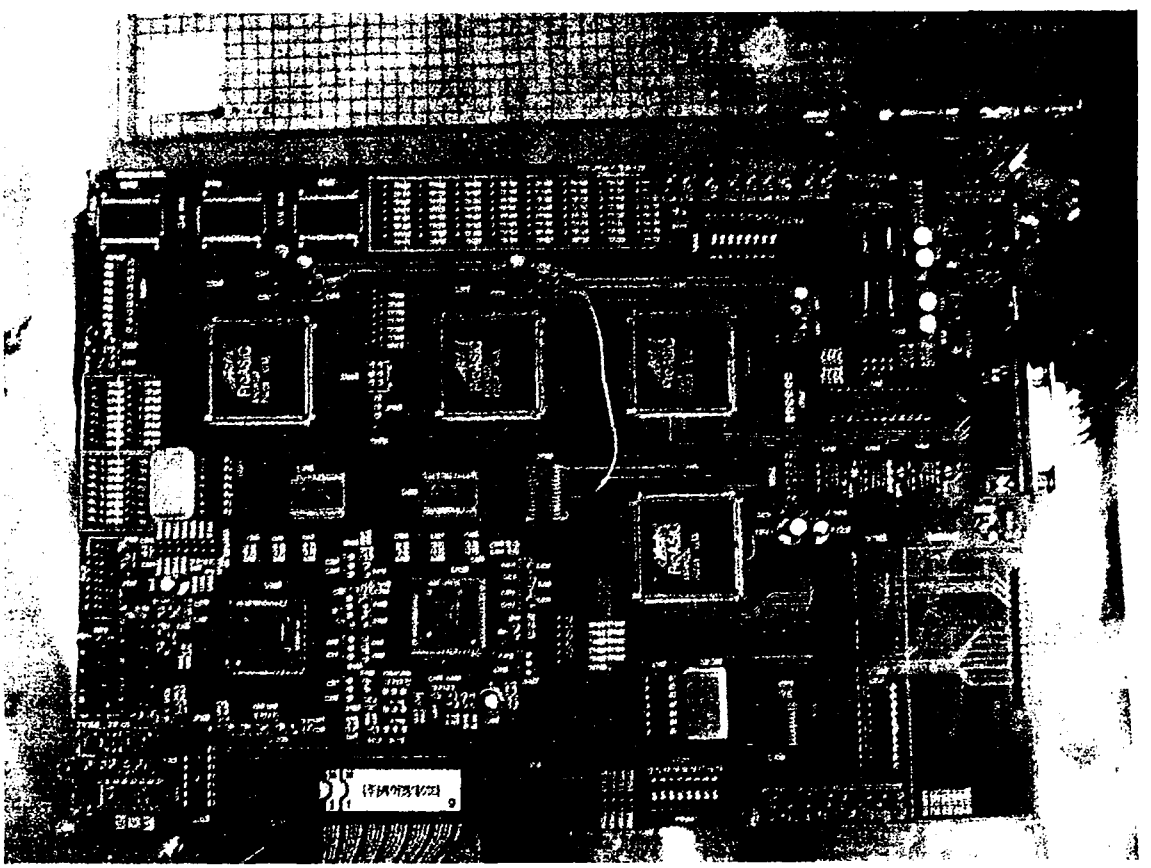


CAN FPGA는 System의 Data Bus와 구분되어 지는 I/O Data Bus와 연결이 되어 있고 FPGA 내부 Coding으로 CAN을 구현하여 테스트 할 수 있도록 되어있다.

(라) FPGA4 (SCC FPGA)



SCC FPGA는 System의 Data Bus와 구분되어 지는 I/O Data Bus와 연결이 되어 있고 FPGA 내부 Coding으로 SCC를 구현하여 테스트 할 수 있도록 되어있다.

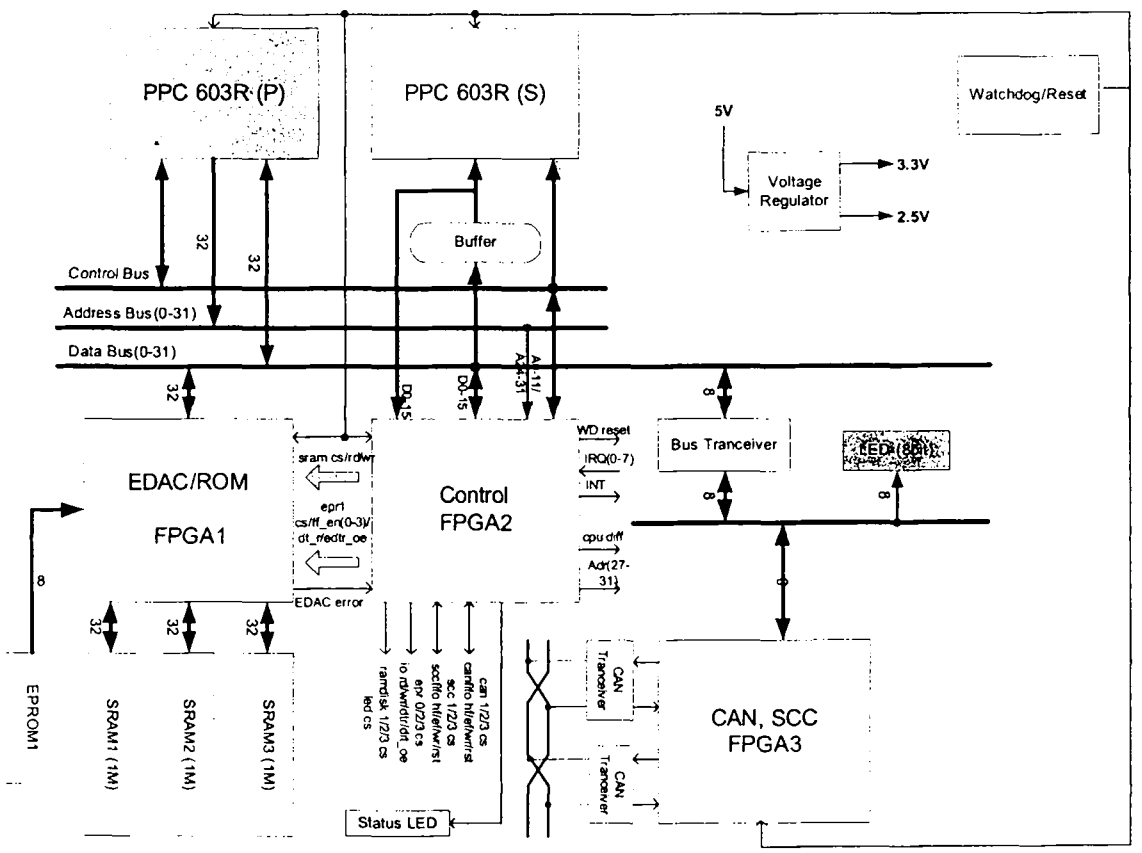


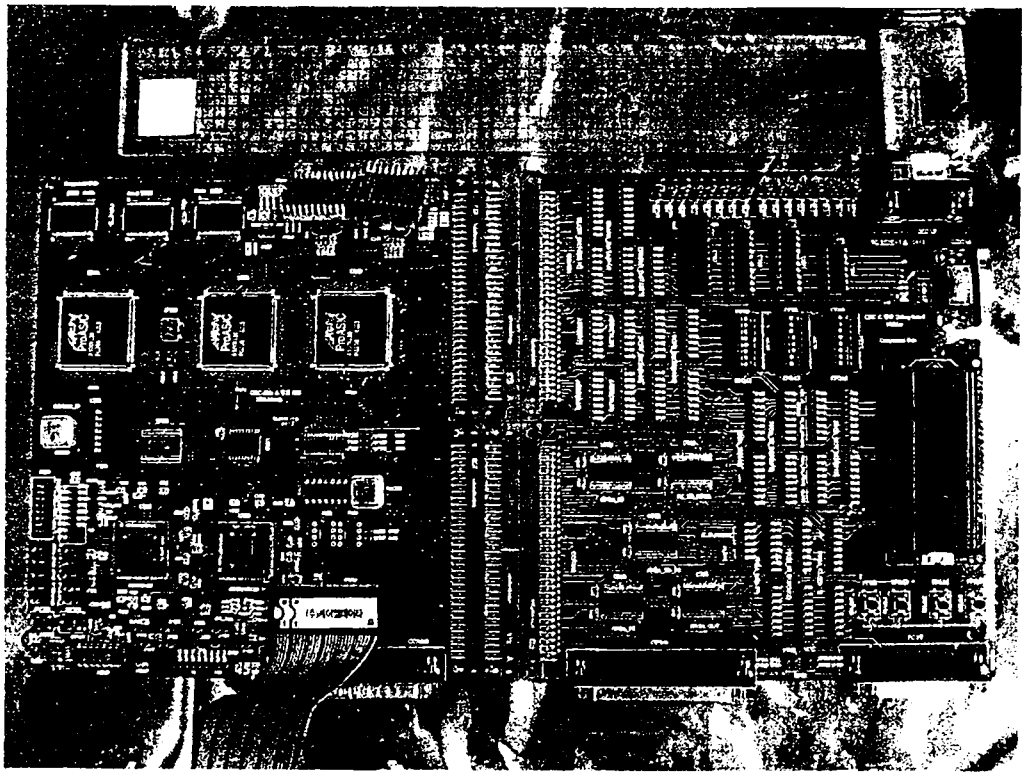
다 PowerPC603E를 이용한 소형화 보드 개발

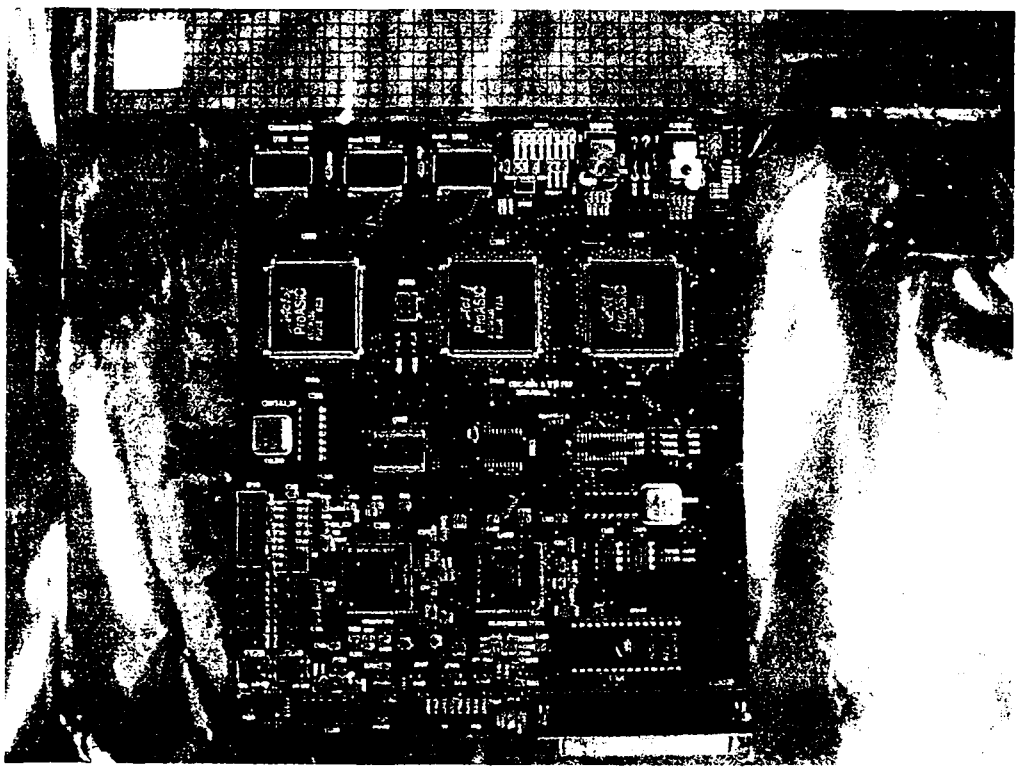
(1) System Overview

2차 Board에서 Power PC 603을 사용한 Dual CPU 및 TMR Memory와 SCC 및 CAN의 구현을 확인하였다. 최종 3차 Board에서는 성능이 입증된 Hardware를 소형화 및 경량화 시키는 데 초점을 맞추었다. 2차 Board의 FPGA3(CAN), FPGA4(SCC)를 하나의 FPGA에서 CAN 및 SCC를 구현하도록 하여 총 3개의 FPGA를 사용하였고 거의 대부분의 부품을 SMD 부품으로 사용을 하여 크기를 대폭 줄였다.

(2) System Block Diagram







라 PCB 개발

(1) 개요

본 과제에서는 우주 환경을 고려한 소형화 OBC 보드를 개발하기 위하여 CPU, 메모리, FPGA, 등이 실장 되었으며, 그 중 가장 중요한 부품 중의 하나인 인쇄회로기판(Printed Circuit Board)에 대한 연구 내용을 기술한다. 인공위성이 겪는 우주 및 발사 환경 때문에 인쇄회로기판의 패턴의 폭, 길이 및 동박 두께에 대한 고려 없이 보드를 제작하면, 보드상의 부품들이 오동작으로 인해 위성의 임무 실패를 초래할 수 있다. 그러므로 본 과제에서는 우주환경을 극복하고 정상적으로 동작하고 있는 과학기술위성1호의 Artwork기술에 대해서 기술하고, 본 과제에서 개발한 보드에 적용된 Artwork 설계 등을 기술함으로서 개발된 보드가 우주 환경에서도 임무를 안전하게 수행할 수 있는 방법에 대해 기술한다.

.

(2) 과학기술위성1호에 사용된 Artwork 설계 규격

과학기술위성1호 PCB는 STD235, IPC2221의 규격을 기준으로 Artwork 설계되었다. 이러한 설계시 다음과 같은 조건을 고려하여야 한다.

- 회로 특성을 이해할 수 있는 회로 지식
- 설계 기술, 구조 설계 기술
- PCB 고 신뢰성 기술, 전자 특성 평가 기술
- 제품 Soldering 조립 기술

특히 본 과제에서는 설계시 필요한 신호선에 대한 규격, 신호선들의 간격에 대한 규격 등에 대한 규격을 기술하고 이에 따라 PCB를 설계하며, 일반적인 PCB 설계와 비교를 위하여 괄호()를 이용하여 일반적인 PCB시 기준까지 기술한다.

(가) 신호선에 대한 규격

신호 종류	형태		비고
전원부	VCC	Plane	기타 : P5V, P3V3, P2V5 등
	GND	Plane	
	AGND	Plane	
	기타	Plane	
신호부	특정 신호	20mil (10~15mil)	CLK, 전체 RST, Tx 및 Rx 등
	일반 신호	12mil (6~8mil)	Data, Addr, 등

그림 55 신호선에 대한 규격

(나) 신호선들의 간격에 대한 규격

l 패드와 패드 간의 간격 : 12mil(6~8mil)

l 패드와 비아 간의 간격 : 15mil(6~8mil)

l 비아와 비아 간의 간격 : 12mil(7~10mil)

l 패턴과 패턴간의 간격 : 12mil(6~8mil)

l 비아와 패턴간의 간격 : 12mil(6~8mil)

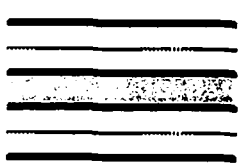
l 패턴과 패턴간의 간격 : 12~25mil(10~15mil)

여기서 패드는 부품의 접속과 취부에 사용되는 도체 패턴의 일부이며, 비아는 도체 층 상호간의 접속을 목적으로 사용하는 홀을 의미한다.

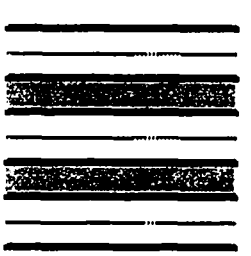
(다) PCB 층 구성

그림은 과학기술위성 1호에서 사용한 PCB 층의 구성도이다. 하지만 그라운드 층과 파워(P5V, P3V3, P2V5) 층은 환경에 따라 위 그림과 다르게 층 구성을 할 수 있으며, 위 그림과 다르게 층 구성을 하는 경우는 중요시 되는 신호 선이 파워 층과 인접해 있는 경우, 파워 층들이 인접해 있는 경우 등을 예로 들 수 있다.

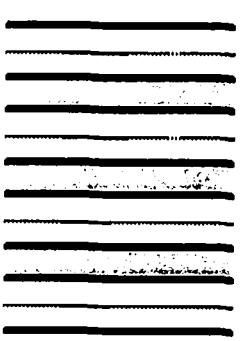
Four-Layers Stack



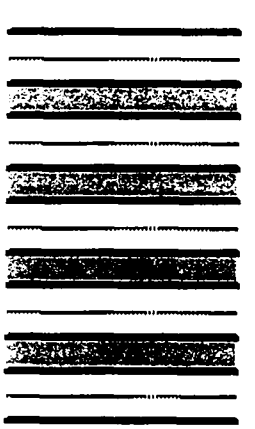
Six-Layers Stack



Eight-Layers Stack



Ten-Layers Stack



(라) PCB 제원

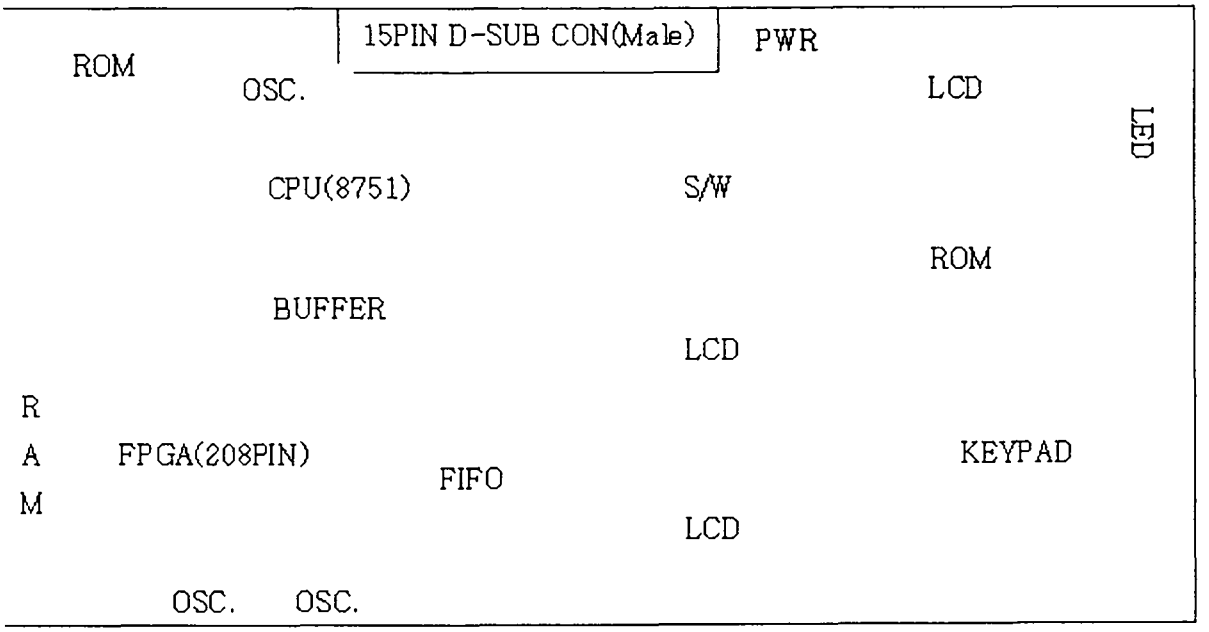
다음은 과학기술위성 1호에서 사용한 PCB 재질과 두께, 동박 두께, 패턴 재질에 관한 규격을 열거하였다.

- PCB 재질 : FR4(Flame Retardant)
- PCB 두께 : 1.6T, 2.0T
- 동박 두께 : 하프 온즈(1/2OZ), 원 온즈(1OZ)
- 패턴 재질 : 주석 납 합금(Sn-Pb), 금 도금

(3) . 회로 특성을 고려한 Artwork 설계

(가) OBC PCB 배치도(1차보드)

1차 보드의 특징은 8751 마이크로 프로세서를 이용한 테스트 보드의 제작이며, 테스트를 위한 각종 부분들이 필요하다..



위 그림은 회로 특성을 고려한 Artwork 설계 시 필요한 사항들만 배치하여 표시한 그림이다.

사용한 입력 전원은 +5V이며 전압 변환기(Regulator)를 이용하여 VCC(IC+5V), +2.5V, +3.3V 출력을 CPU, FPGA, 메모리, 버퍼 등에 입력전원으로 연결하였다. 이때 입력 +5V는 동박을 형성하여 안전한 전원이 공급할 수 있도록 하였으며 출력 전원 중 VCC(IC+5V)는 플랜(PLAN)을 형성하였고, +2.5V, +3.3V는 일부만 동박을 형성하여 설계하였다. 리셋 신호의 패턴의 폭은 15mil이며 클럭 신호는 20mil로서 패턴의 길이를 최소화 하였다. 시리얼 통신의 패턴 두께와 폭은 각각 15mil와 12mil로 설계하였다.

아래 그림들은 각 층별 Artwork 설계를 보여 주고 있다. 각각은 부품 실장층, 중요 신호층, 그라운드 층, 파워층, 일반 신호층 , 납땜이 가능한 외부층으로 이루어지며, 부품 실장시 중요한 정보를 기록하는 부품 정보에 해당하는 실크 스크린을 표시하였다.

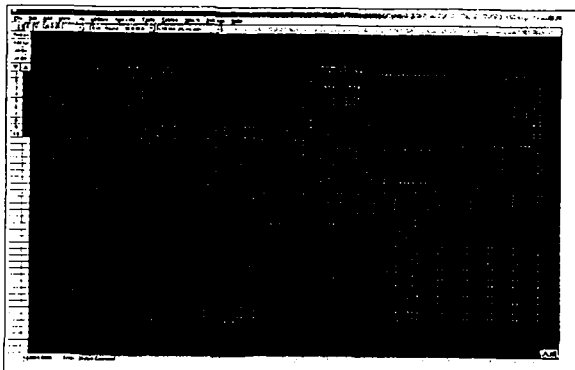


그림 58 부품 실장층
(LAYER1-Component Side)

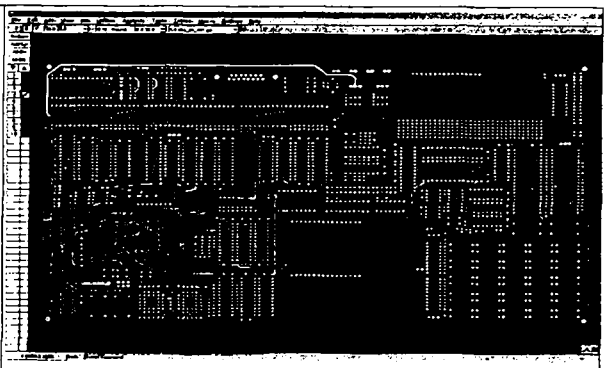


그림 59 중요 신호층
(LAYER2-Internal Side Single Layer)

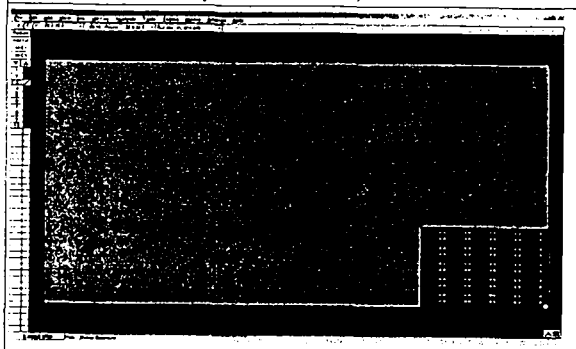


그림 60 그라운드층
(LAYER3-GND)

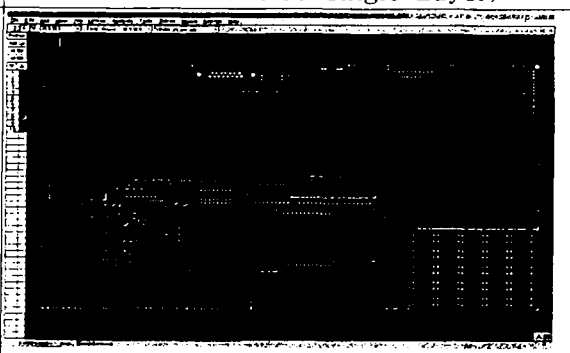


그림 61 파워층
(LAYER4- +5V, P3V3, P2V5)

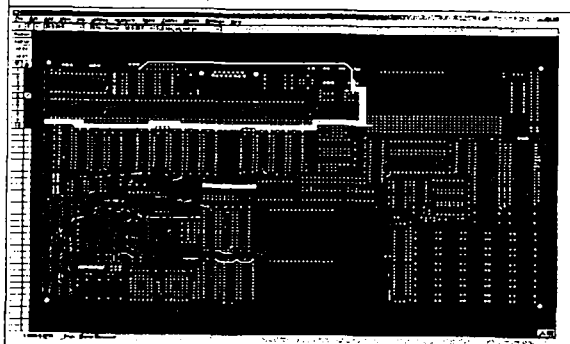


그림 62 일반적인 신호층
(LAYER5)

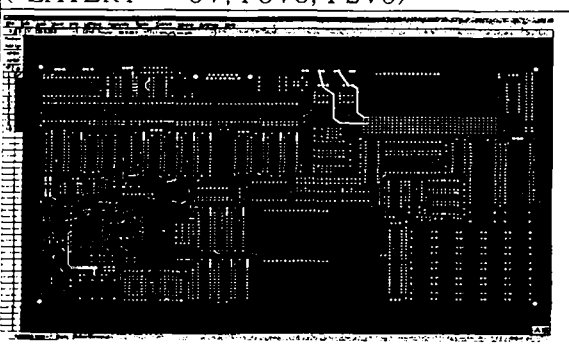


그림 63 납땜 층
(LAYER6 - Bottom)

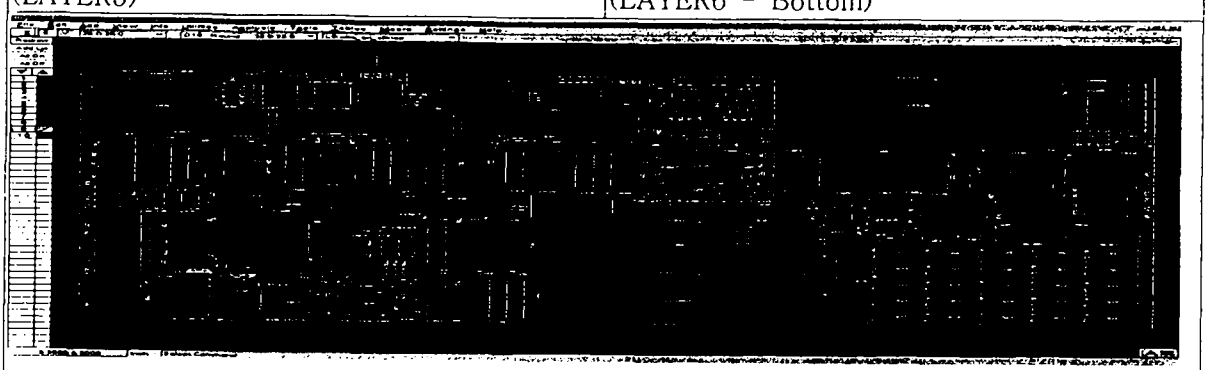
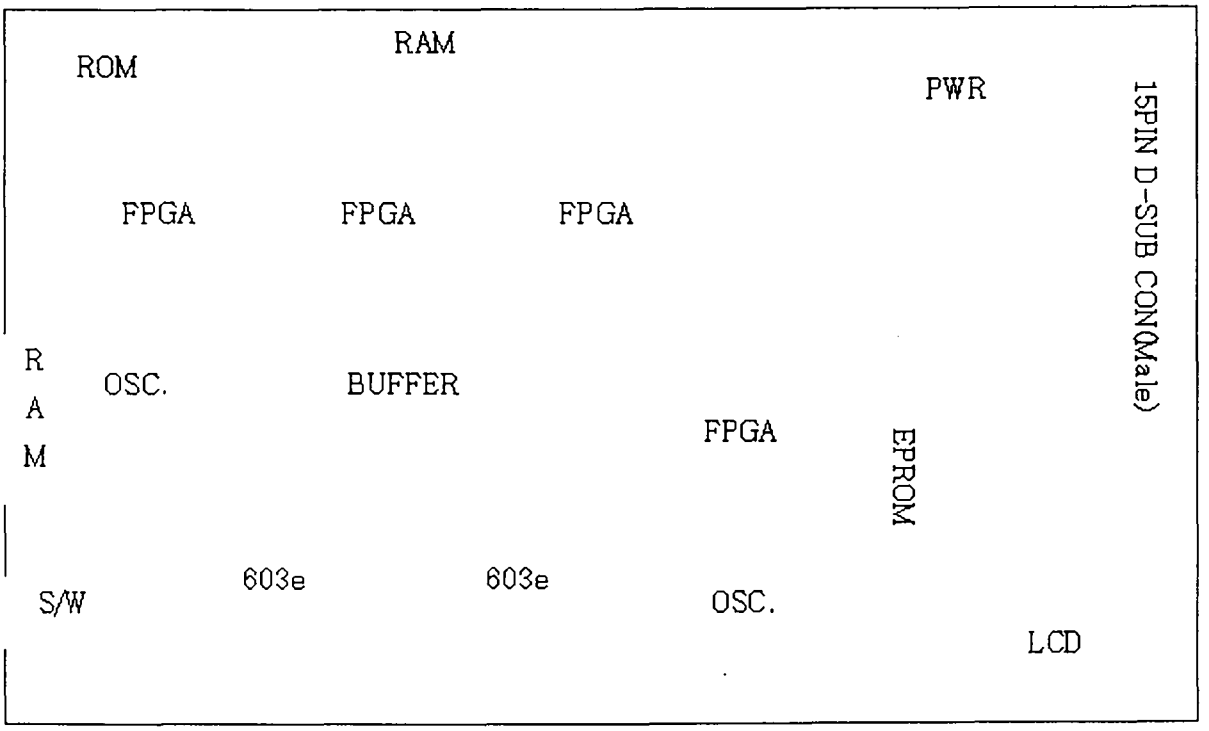


그림 64 SilkScreen

(나) OBC PCB 배치도(2차보드)

2차 보드는 603E CPU를 이용한 보드의 설계이다. 이 보드의 설계의 목적 때문에 FPGA의 개수의 제한을 두지 않았고, 그로 인해 4개의 FPGA를 사용하였다.



그림은 2차 보드의 주요 부품들을 배치한 배치도이다. 2차보드의 Artwork 설계의 가장 큰 변화는 CPU변화이다. 즉 CPU(87C51)가 DIP(Dual In Package) 타입을 BGA(603e, Ball Grid Array) 타입의 형태로 되었다. Artwork 설계 시 BGA는 패드뿐만 아니라 가상의 비아 홀이 필요하다. 가상의 비아 홀은 납이 묻히는 면적을 고루 분포하기 위해 일정한 간격으로 패드마다 비아 홀을 형성해야 한다. 또한 층 구성도 BGA 형태로 바뀜에 따라 6층에서 8층으로 2층이 더 증가하였다.

전원은 1차년도와 같이 VCC, +2.5V, +3.3V으로 각 IC에 입력 전원으로 연결된다. 1차년도에서는 +2.5V와 +3.3V의 패턴을 두껍게 하였으나 2차년도에서는 +2.5V와 +3.3V 는 플랜으로 형성하여 파워의 공급을 안정화 하는데 역점을 두었다. 클럭 신호는 안전한 패턴을 형성하기 위해 신호 층이 적은 층에 형성하였다.

다음 그림들은 각 층별 Artwork 설계를 보여 주고 있다.

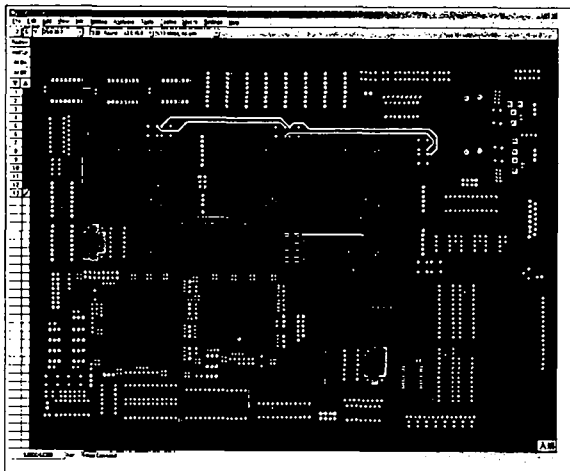


그림 66 부품 실장 층

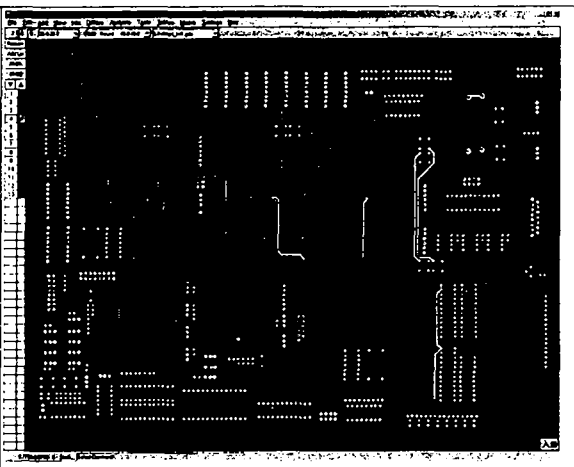


그림 67 중요 신호선 층

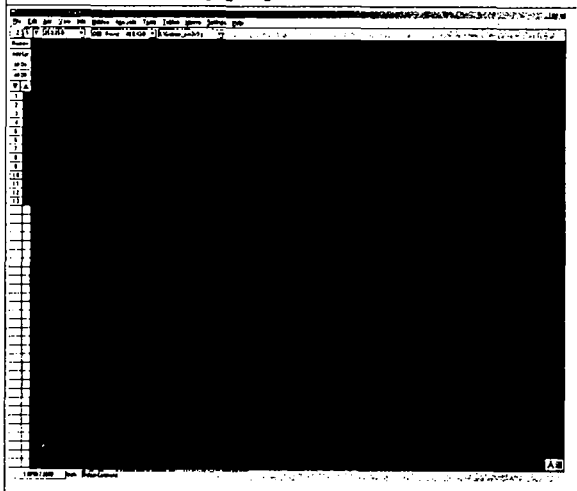


그림 68 +3.3V 동박층

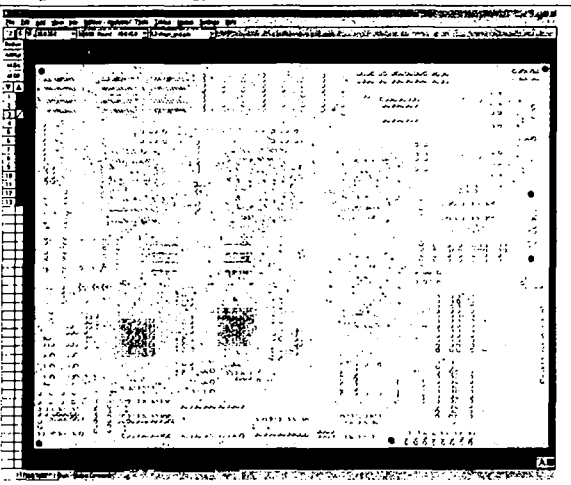


그림 69 그라운드 층

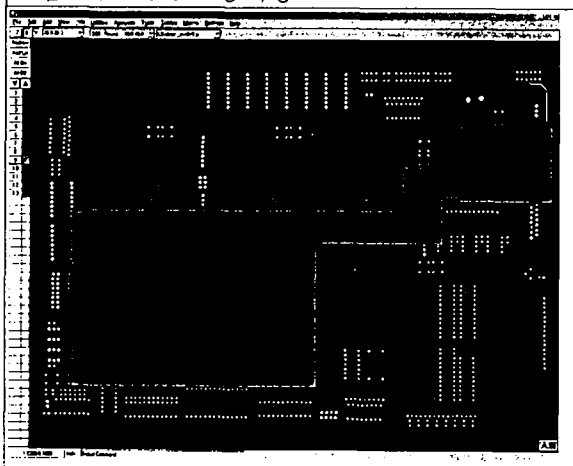


그림 70 +2.5V 동박층

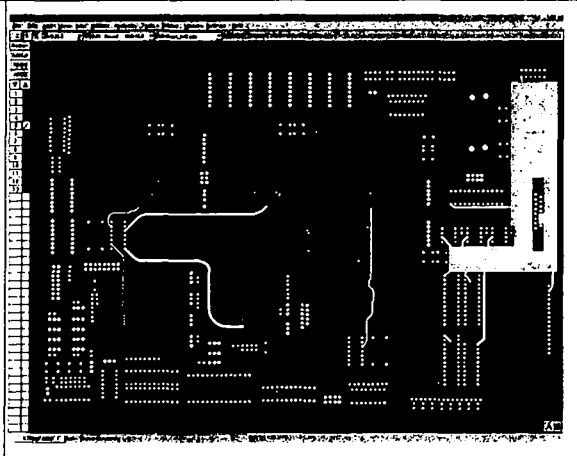


그림 71 +5V 및 클럭 보호층

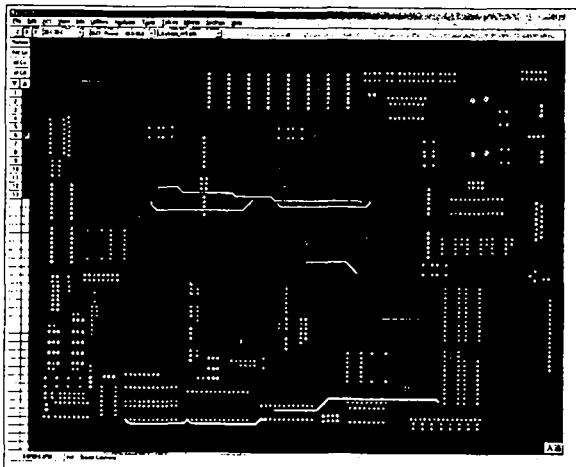


그림 72 일반 신호 층

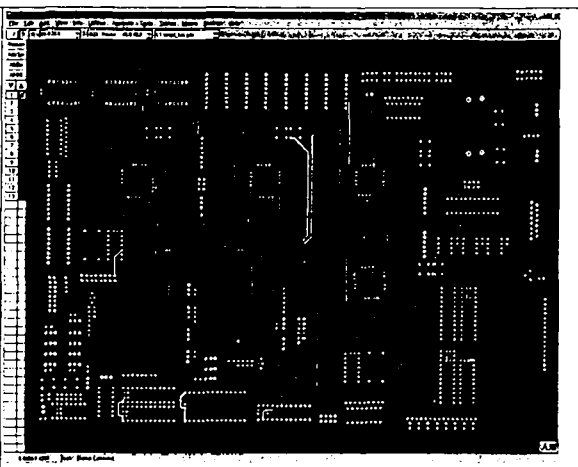


그림 73 납땜 층

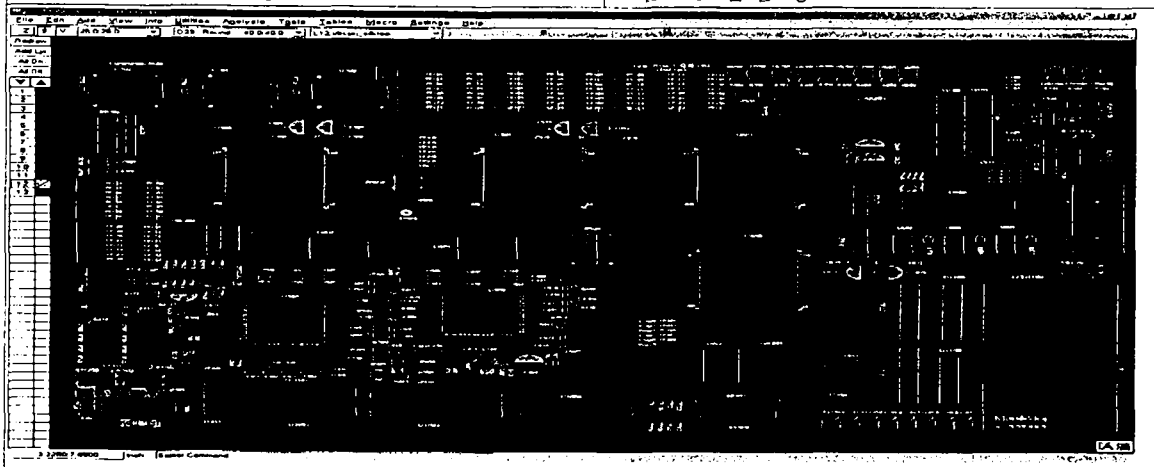
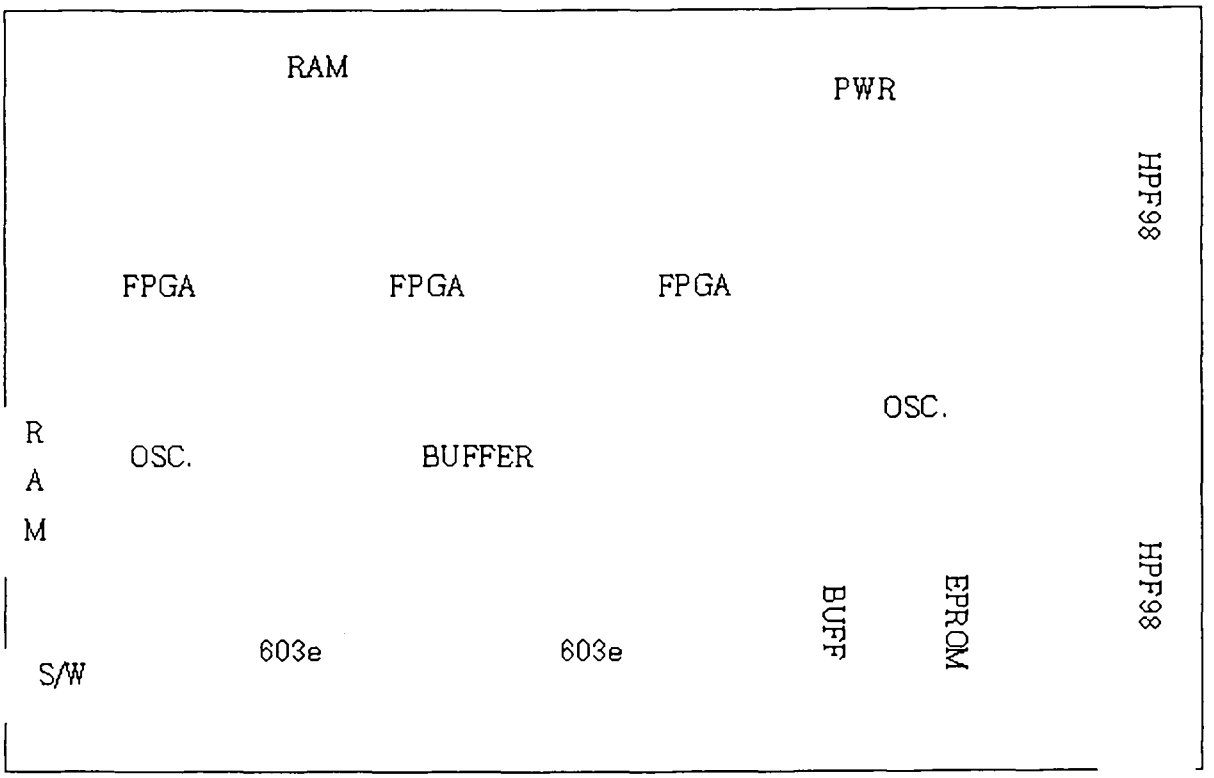


그림 74 부품 실장시 필요한 정보 표시 층 SilkScreen

3.3 OBC PCB 배치도(3차보드)

3차 보드는 개발한 2차 보드를 기반으로 소형화하였다. 이를 위해 FPGA를 4개에서 3개로 줄였으며, 각 부품들도 가능한 작은 부품을 이용하였으며, 2차보드에서 생성한 디버깅 포인트를 HPF98핀 Connector 2개를 이용하여 외부 디버깅 보드로 옮겼다.



그림은 3차보드의 주요 부품의 부품도를 나타낸다. 3차보드의 Artwork 설계는 주 목적은 PCB 크기의 소형화이다. 하지만 최소화 하지는 않고 우주 환경을 고려하여 PCB 개발에 부합된 형태로 변환하였다. 설계 결과 PCB 크기는 2차보드가 285*200mm 이었으며, 3차보드는 184*200mm이었다.

또한 2차보드에서는 FPGA의 파워인 VDDP, VDDL을 패턴으로 형성하였으나, 3차보드에서는 FPGA의 VDDP, VDDL을 동박으로 형성하여 FPGA의 파워를 최대한 안정화하였다.

다음의 그림들은 3차 보드의 설계 결과를 보인다.

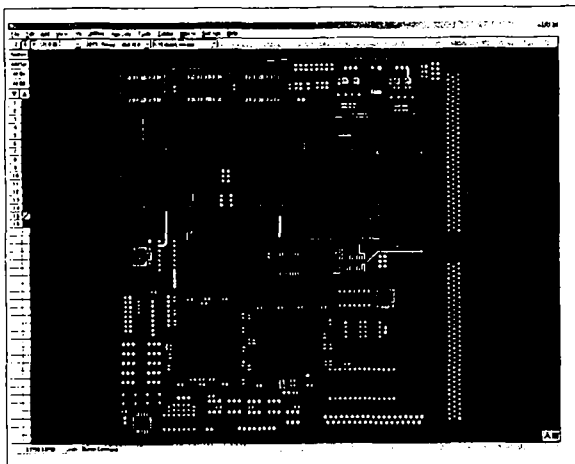


그림 76 부품 실장 층

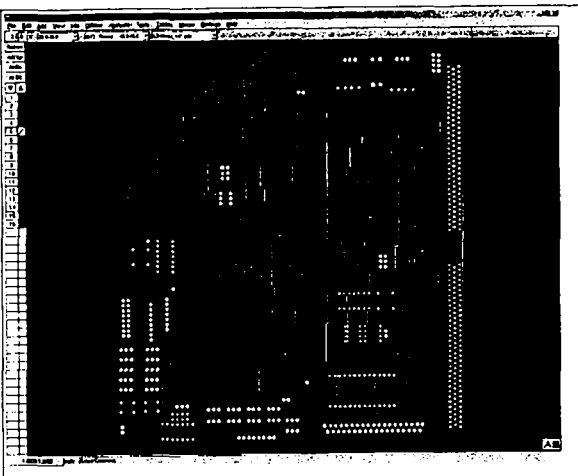


그림 77 중요 신호 층

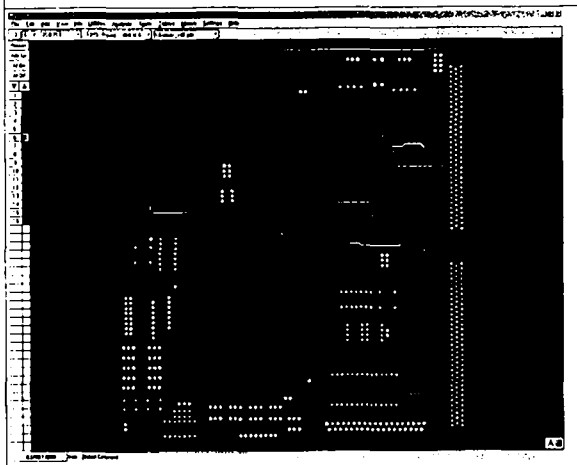


그림 78 일반 신호 층

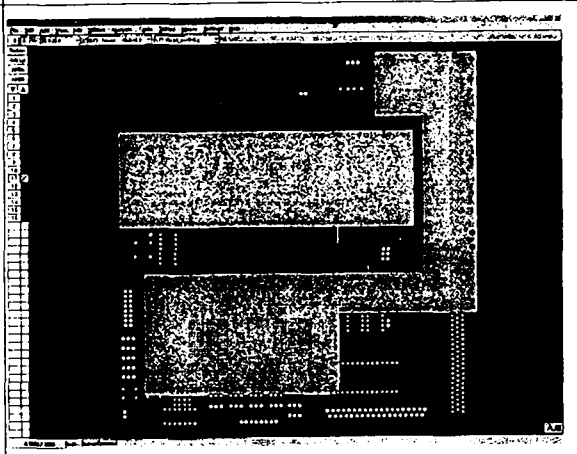


그림 79 +2.5V 동박층

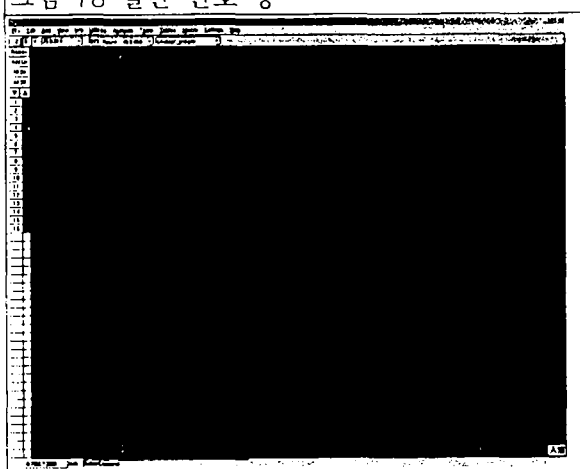


그림 80 그라운드 층

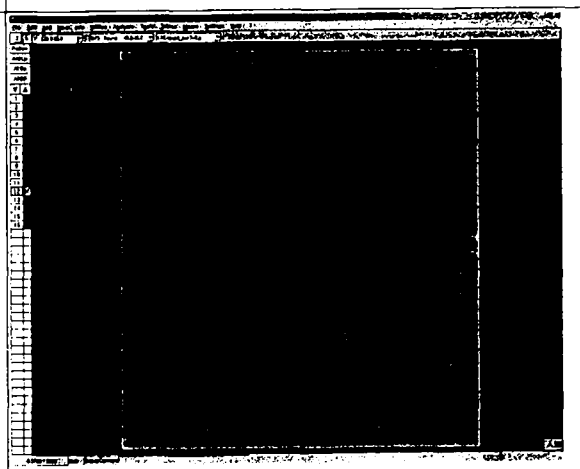


그림 81 +3.3V 동박 층

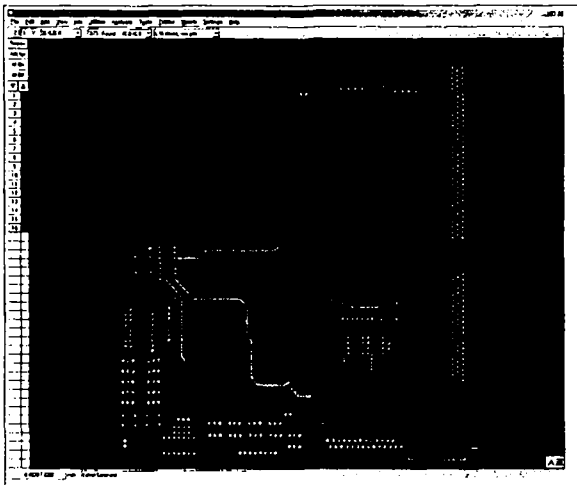


그림 82 +5V, VDDL 동박층

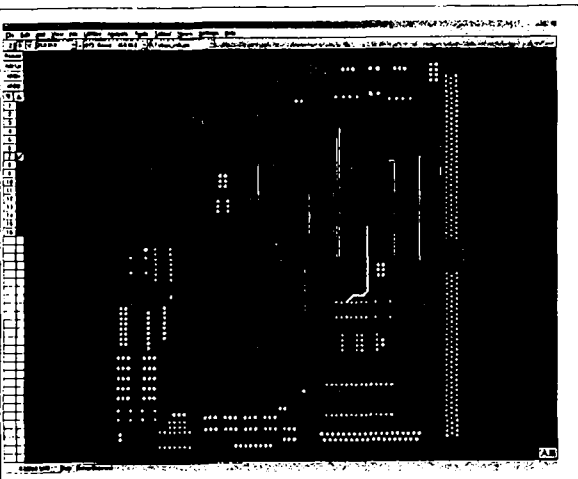


그림 83 일반 신호 층

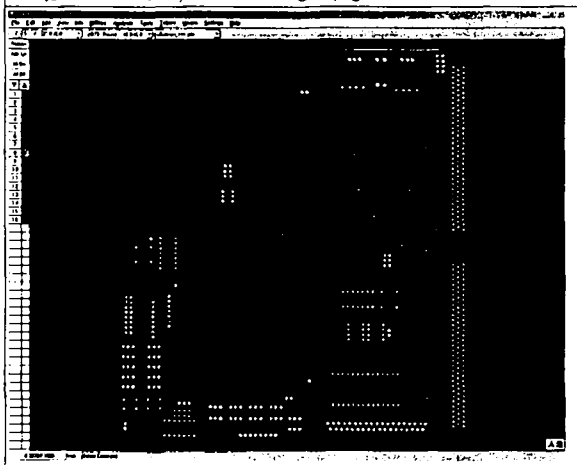


그림 84 일반 신호층

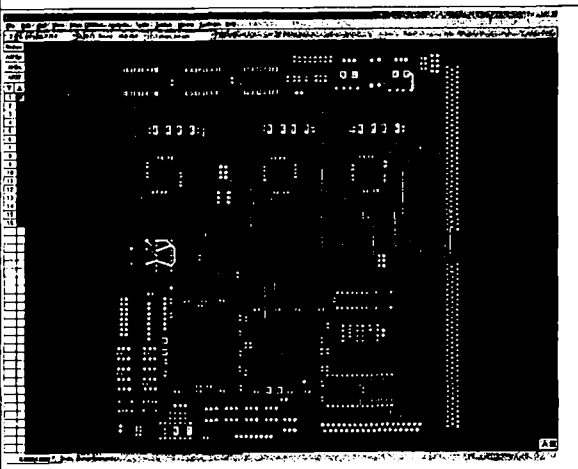


그림 85 납땜 가능한 외부 층

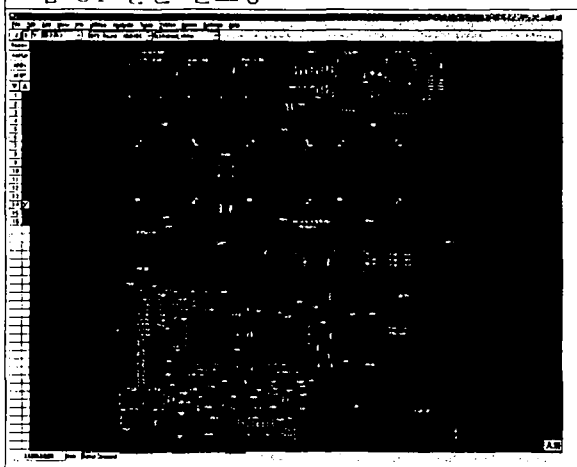


그림 86 부품 실장 정보
(SlikScreen TOP)

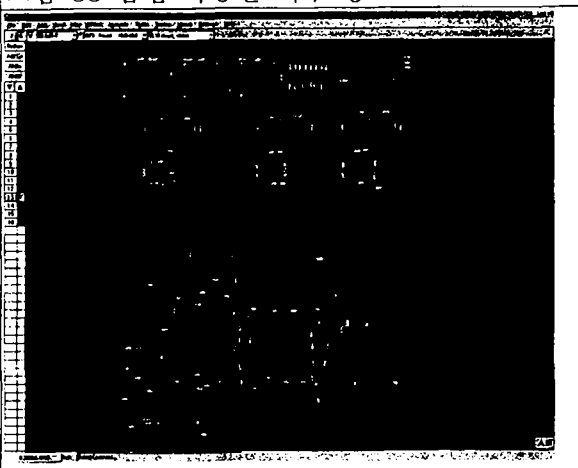


그림 87 부품 실장 정보
(SilkScreen Bottom)

2 소형화 OBC 로직 구현

가 FPGA 제어 로직 구현

(1) EDAC 로직 구현

(가) SRAM SEU 극복을 위한 TMR 구조

본 연구에서는 SEU 극복을 위하여 TMR 구조의 SRAM을 이용하였다.

이 구조는 3비트 중 한 비트에서 SEU가 발생했을 경우 극복이 가능한 방법으로 우리별, 과학위성 씨리즈에서 7비트 중에 하나의 비트에 SEU가 발생할 경우 극복 가능한 (7,4)Hamming Code 방식보다 오류 발생에 대한 극복 확률이 높다. 강한 입자에 의해 SEU가 발생시 주변 메모리 소자가 같이 영향을 받을 수 있는데, 이 경우 주변의 값을 이용하는 과학기술위성에서 사용하는 방법보다 TMR이 안전하다고 할 수 있다. 또한 TMR의 경우 판단하고 복구하는 로직자체가 간단한 로직에 의해 구현되므로 빠른 동작이 가능하다.

프로그램 메모리는 데이터를 저장하기 위한 데이터 메모리와 여분의 메모리로 구성되며, 본 연구에서는 두개의 같은 메모리를 추가적으로 사용하여 3개의 같은 데이터를 갖는다. CPU가 프로그램 메모리로부터 데이터를 읽어 올 때는 EDAC FPGA에서 3개의 데이터 메모리에서 모두 데이터를 읽어온다. 그리고 3개의 비트 중에서 많은 쪽을 대표값으로 하여 CPU로 전송한다. 메모리에 저장된 데이터의 에러에 대한 검사는 항상 CPU가 데이터를 읽을 때 이루어진다. 또한 메모리의 오류는 3비트당 1비트만 복구가 가능하기 때문에, 메모리 전체의 데이터가 항상 오류가 없도록 하기 위해서는 정기적으로 전 메모리를 읽고 써서 오류를 복구하고 다시 저장할 필요가 있다(Memory Scrubbing). 그렇지 않으면 하나 이상의 SEU에 의해 데이터 3비트중 2개 이상의 오류가 발생할수 있어 오류를 복구할 수 없는 상황이 발생할수 있기 때문이다. 과학위성1호에서는 매초마다 전 메모리 영역에 걸쳐 1Kbyte씩 프로그램 메모리를 다시 쓰는 작업을 하도록 구현되었으며, 본 연구에서는 확률이론을 배경으로 적절한 Wash 기간을 분석하였다.

본 연구에서 개발한 보드는 구조상 메모리 Wash 부분에서 CPU의 동작이 필요로 하다. 하지만, 이러한 Wash부분을 FPGA에서 DMA를 처리하는 것처럼 한다면, CPU의 Overhead는 줄어들 수 있다. Wash 부분에 대한 FPGA를 구현한다면, CPU에서 블록에 대한 Memory Wash를 요구할 경우, FPGA는 CPU의 동작을 멈추게 하고 블록의 시작 번지에서부터 끝 번지까지 4씩 더해가며 메모리를 읽고, 만일

EDAC이 발생한 메모리에 대해서만 쓰기 동작을 하면 된다. 기본적으로 CPU에서 하는 Wash에 대한 Overhead는 (명령 Fetch, 명령 분석, 메모리 읽기, 명령Fetch, 명령분석, 메모리 쓰기) 블록크기/4 만큼 하여야 하지만, 이 방식으로 할 경우 CPU에 대한 Wash Oberhead는 (명령 Fetch, 명령 분석, FPGA에 쓰기+ 메모리 읽기* (블럭크기/4) + 메모리 쓰기*EDAC 횟수) 에 해당하는 명령을 수행하면 된다. 여기서 메모리를 읽고 쓰는 Operation이 가장 많은 Operation을 이용하므로 일반적인 경우 4N 이지만 이 경우 N번 정도의 overhead 가 있어서 4배의 Overhead를 줄일 수 있게 된다.

(나) TMR을 위한 VHDL 코드

다음은 TMR을 통한 VHDL 코어를 나타내고 있다. 3개의 비트 중에서 많은 쪽을 선택하는 것이다. 또한 이들에 대한 32비트 처리는 코어를 이용하여 32개의 비트를 동시에 처리하도록 하였다.

```
OUT_TMR <=  '1' when IN1 = '0' and IN2 = '1' and IN3 = '1' else
            '1' when IN1 = '1' and IN2 = '0' and IN3 = '1' else
            '1' when IN1 = '1' and IN2 = '1' and IN3 = '0' else
            '1' when IN1 = '1' and IN2 = '1' and IN3 = '1' else
            '0';
```

```
gen_TMR_EDAC: for i in 0 to 31 generate
    u_TMR_EDAC : TMR_EDAC port map(
    IN1 => SRAM1_DATA(i),
    IN2 => SRAM2_DATA(i),
    IN3 => SRAM3_DATA(i),
    OUT_TMR => i_d_tmr(i));
    end generate gen_TMR_EDAC;
```

본 연구에서 이들에 대한 검증은 SRAM에 대한 write enable 신호를 외곡 함으로써 SRAM에 데이터를 쓸 수 없게 하여 다른 SRAM 모듈과 다른 값을 가질 수 있도록 하였다. 본 프로젝트의 결과 SRAM에 대하여 하나의 모듈의 오류 발생에 대해서도 CPU의 올바른 동작 여부를 확인 할 수 있었다.

(2) WatchDog 로직 구현

시스템이 무언가 엉뚱하게 동작을 하는데도 그 시스템 스스로 복구하지 못하는 경우가 있다. 기계장치나 운영체제에어 가장 흔한 문제로는 두 개 이상의 부품이나 프로그램이 충돌을 일으켜 더 이상 진행 할 수 없는 교착에 빠져 있는 경우나 운영체계에서 메모리 관리에 문제가 발생했을 경우 등을 들 수 있다. 가끔은 시스템 스스로가 정상적인 상태로 복구되는 경우도 있겠지만, 그것이 언제 끝날지 시간을 알 수 없으며 때로 시간이 과도하게 지연되는 수가 발생한다. 이럴 경우 WatchDog Counter를 두어 시스템을 복구할 수 있다. Watchdog Timer는 카운터의 일종으로 계속 정기적인 리셋이 입력되는 것이 정상상태이며, 비정상 상태에서 요구되는 리셋이 없어지고 카운터가 정한 값보다 올라가면 시스템을 재부팅시키는 펄스를 발생시키는 역할을 한다. 프로세스에 응용되는 프로그램을 작성하고 이식한 후 Run했을 때 동일한 루틴을 계속 반복하는 무한 루프(Endless Loop)를 돌거나 폭주 또는 프로그램 어드레스 영역 밖의 공간을 지정하게 되면 메모리에서 명령어를 가져와(patch) 실행할 수 없으므로 시스템이 정지된 상태가 된다. 이때 프로세서는 Holt, 즉 시스템의 안정성에 대해 신뢰할 수 없는 상태, 아무런 일을 할 수 없는 CPU가 죽은 때라 할 수 있다. PC도 어쩌다 발생할 경우가 있는데, 이 경우 사용자는 짜증스럽게 리셋 버튼을 눌러 빠져 나온다. 이렇게 CPU가 Holt 상태가 되었을 때 강제적으로 리셋을 눌러 초기화하는 감시용 타이머인 WDT(Watchdog Timer) 기능이 있는데 문자 그대로 집을 지키는 개를 뜻한다. 이는 주기적으로 개에게 밥을 주는데 일정시간이 지나도 먹이를 주지 않으면 주인을 물어 버리는 경우 즉, 강제적으로 리셋 또는 인터럽트(Interrupt)를 발생시킨다.

본 연구에서는 CPU에서 WatchDog Clear 신호를 줄 경우 Counter를 0으로 하고, 그렇지 않을 경우 계속 Counter를 증가 시키다가 정해진 시간(DefinedWaitingTime)이 되면 Reset신호를 LOW 신호로 만들어 SignalTime에 해당하는 시간만큼 LOW를 유지하는 방식을 취하여 VHDL 로 구현하였다.

이에 대한 검증은 간단히 reset 시 LED를 점등하므로서 확인이 가능했으며, ROM EMULATOR를 통한 ROM 프로그램 Upload 시 바뀐 ROM 프로그램으로 인해 CPU는 HOLT 상태가 되는데, 정해진 WatchDog 시간 이후 CPU RESET이 걸려 제대로 수행하는 것을 볼 수 있었다. 또한, PowerOn 시 Reset신호를 제거하고

WatchDog에 의한Reset신호를 인가하여WatchDog 설정 시간 이후 보드가 동작하는 것을 이용하여 WatchDog의 동작여부를 검증할 수 있었다.

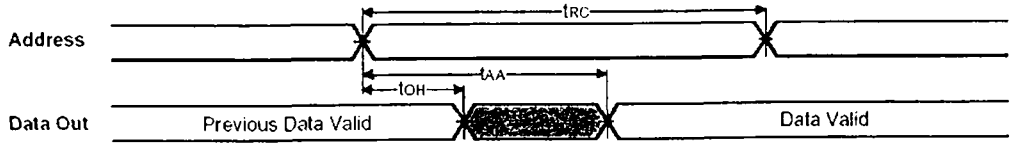
```
-- watchdog counter --
    process( CLK, WD_SEL )
    begin
        if ( WD_SEL = '1' ) then i_wd_cnt <= (others => '0' ) ;
        elsif ( CLK'event and CLK = '1' ) then
           if ( i_wd_cnt < DefinedWatingAndSignalTime ) then
                    i_wd_cnt <= i_wd_cnt + 1 ;
           else
                    i_wd_cnt <= ( others => '0')  ;
           end if ;
        end if;
    end process;

    i_wd_rst <= '1' when ( i_wd_cnt < DefinedWaitingTime ) else
                '0' ;

    WD_RST    <= i_wd_rst ;
```

(3) SRAM 제어 로직 구현

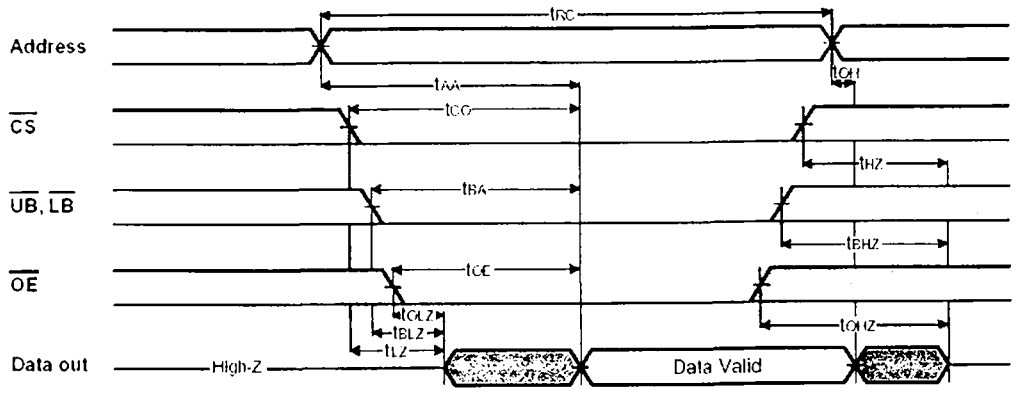
아래 그림을 삼성 SRAM을 읽거나 쓰기위한 타이밍 조건이다.

TIMING DIAGRAMS

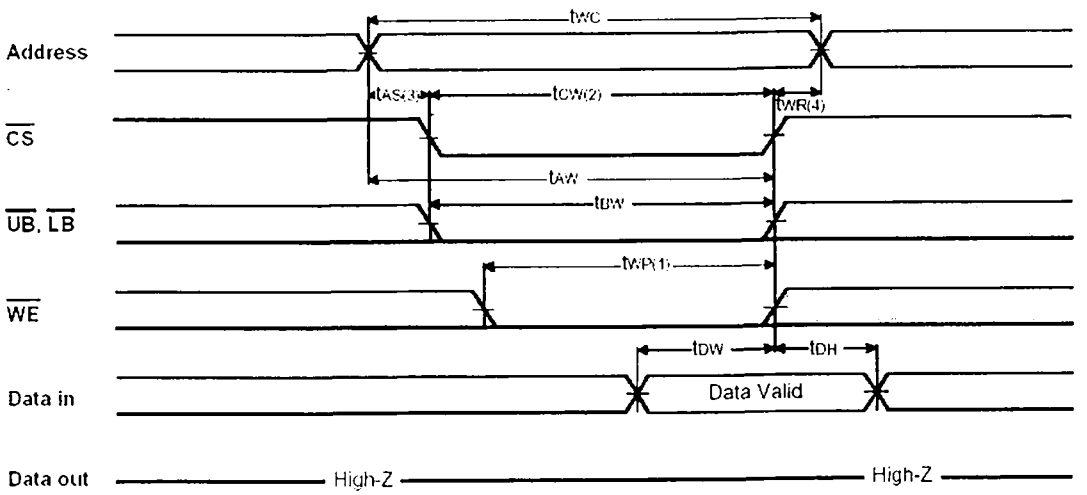
TIMING WAVEFORM OF READ CYCLE(1) (Address Controlled. $\overline{CS}$=$\overline{OE}$=VIL, $\overline{WE}$=VIH, $\overline{UB}$ or and $\overline{LB}$=VIL)



TIMING WAVEFORM OF READ CYCLE(2) ($\overline{WE}$=VIH)



TIMING WAVEFORM OF WRITE CYCLE(2) ($\overline{CS}$ Controlled)



이러한 타이밍을 조건을 맞추기 위해서 제어 FPGA2에서는 이에 해당하는 제어 신호를 생성하여야 한다. 이를 위해서 본 연구에서는 이것은 타이밍에 대한 상태 변화도를 정의하고 이를 이용하여 VHDL 코드를 생성하였다. 다음의 그림은 SRAM 제어를 위한 상태 변화 도이다. 정의한 상태는 S0,S1,S2,S3,S4,S5,S6 이렇게 7개의 상태가 있다.

S0은 준비 및 초기화 단계를 나타내는 상태로서 시스템이 초기화 되거나 SRAM

이 선택이 되지 않은 상태에 머물러 있는 단계이다. 이 단계에서 SRAM_SEL가 선택이 되면, S1으로 상태가 변화 한다.

S1은 SRAM 선택 단계로서 CLK에 의해서 S2로 변화되며 변화시 SRAM_CS_L로 SRAM 선택 신호를 생성한다.

S2는 Read/Write 신호 생성 단계로서 CLK에 의해서S3로 변화되며, 변화시 SRAM_WE_L에 의해서 Read 신호나Write신호를 생성한다.

S3는 Write Done 단계 또는 Read를 위한 시간을 기다리는 단계이다. 이는 CLK에 의해서 S4로 변화하며, SRAM 쓰기를 종료한다.

S4는 SRAM Operation 종료 신호를 생성하는 단계이다.

S5는 SRAM Operaion 종료하는 단계이고

S6는 SRAM Read 종료 단계로서 SRAM에 대한 명령을 수행하기 위한 신호 복귀를 한다.

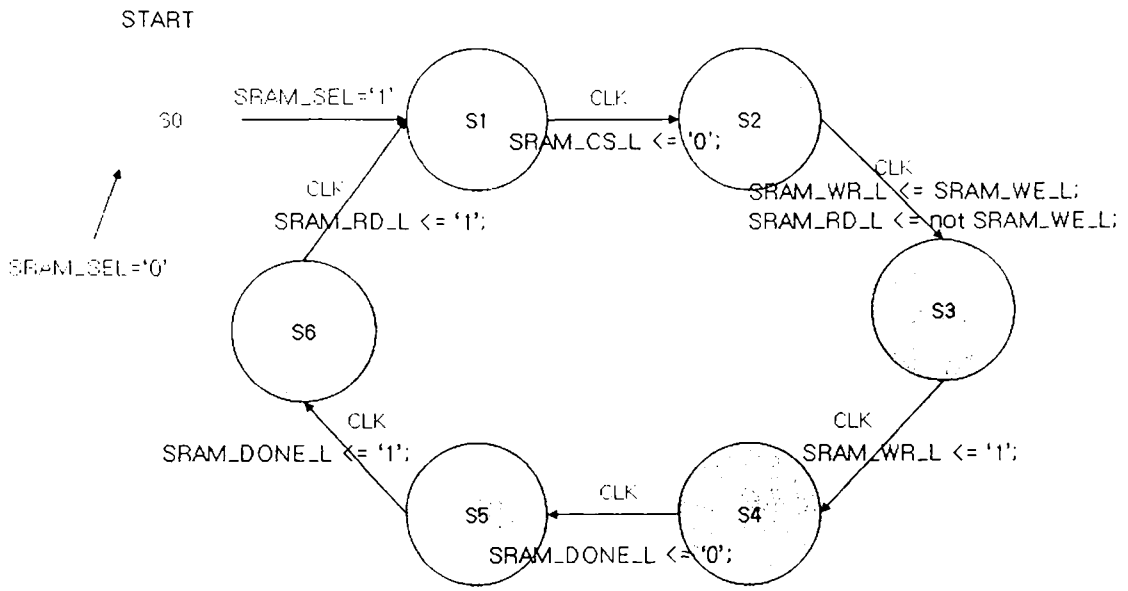


그림 90 SRAM 제어를 위한 상태도

다음은 위의 상태도에 따른 SRAM의 제어를 위한 VHDL 코드를 나타낸다.

```
entity SRAM_RD_WR is
    port( RST_L : in std_logic;                    -- Reset
          CLK : in std_logic;                      -- Clock
```

```
        TSIZ : in std_logic_vector( 0 to 2 );          -- TSIZ[0:2]: Transfer Size
        A_LO : in std_logic_vector( 30 to 31 );        -- A[30:31]: Address
        SRAM_SEL : in std_logic;                       -- SRAM Select
        SRAM_WE_L : in std_logic;                      -- SRAM Read/Write
        SRAM_CS_L : out std_logic;                     -- SRAM Select
        SRAM_RD_L : out std_logic;                     -- SRAM Read
        SRAM_WR_L : out std_logic;                     -- SRAM Write
        SRAM_BE_L : out std_logic_vector( 0 to 3 );    -- SRAM Byte Enable
        SRAM_DONE_L : out std_logic                    -- Address ACK
    );
end; --port DEFINITION AND entity
-------------------------------------------------------------------------

architecture BEHAVIOR OF SRAM_RD_WR IS

type states is (s0, s1, s2, s3, s4, s5, s6);
signal sram_state: states;

begin

    process(CLK, RST_L)
        variable v_sram_be_l: std_logic_vector( 0 to 3 );
        variable v_tsiz_a_lo: std_logic_vector( 0 to 4 );
    begin
        if (RST_L = '0') then
            v_sram_be_l := "1111";
            sram_state <= s0;
            SRAM_CS_L <= '1';
            SRAM_WR_L <= '1';
            SRAM_RD_L <= '1';
            SRAM_BE_L <= "1111";
            SRAM_DONE_L <= '1';
        elsif (CLK'EVENT and CLK = '1') then
```

```
case sram_state is
    when s1 =>
        sram_state <= s2;
        SRAM_BE_L <= v_sram_be_l;
        SRAM_CS_L <= '0';
        SRAM_WR_L <= '1';
        SRAM_RD_L <= '1';
        SRAM_DONE_L <= '1';
    when s2 =>
        sram_state <= s3;
        SRAM_BE_L <= v_sram_be_l;
        SRAM_CS_L <= '0';
        if (SRAM_WE_L = '0') then      -- Write
            SRAM_WR_L <= '0';
            SRAM_RD_L <= '1';
        else                           -- Read
            SRAM_WR_L <= '1';
            SRAM_RD_L <= '0';
        end if;
        SRAM_DONE_L <= '1';
    when s3 =>
        sram_state <= s4;
        SRAM_BE_L <= v_sram_be_l;
        SRAM_CS_L <= '0';
        if (SRAM_WE_L = '0') then      -- Write
            SRAM_WR_L <= '0';
            SRAM_RD_L <= '1';
        else                           -- Read
            SRAM_WR_L <= '1';
            SRAM_RD_L <= '0';
        end if;
        SRAM_DONE_L <= '1';
    when s4 =>
```

```
            sram_state <= s5;
            SRAM_BE_L <= v_sram_be_l;
            SRAM_CS_L <= '0';
            if (SRAM_WE_L = '0') then      -- Write
                SRAM_RD_L <= '1';
            else                           -- Read
                SRAM_RD_L <= '0';
            end if;
            SRAM_WR_L <= '1';
            SRAM_DONE_L <= '1';
        when s5 =>
            sram_state <= s6;
            SRAM_BE_L <= v_sram_be_l;
            SRAM_CS_L <= '0';
            if (SRAM_WE_L = '0') then      -- Write
                SRAM_RD_L <= '1';
            else                           -- Read
                SRAM_RD_L <= '0';
            end if;
            SRAM_WR_L <= '1';
            SRAM_DONE_L <= '0';
        when s6 =>
            sram_state <= s1;
            SRAM_BE_L <= "1111";
            SRAM_CS_L <= '1';
            SRAM_WR_L <= '1';
            SRAM_RD_L <= '1';
            SRAM_DONE_L <= '1';
        when others =>
    end case;

    if(SRAM_SEL = '1') then
        if(sram_state = s0) then
```

```
            sram_state <= s1;
            SRAM_BE_L <= "1111";
            SRAM_CS_L <= '1';
            SRAM_WR_L <= '1';
            SRAM_RD_L <= '1';
            SRAM_DONE_L <= '1';
            -- Aligned Data Transfer +
            v_tsiz_a_lo := TSIZ & A_LO;
            case v_tsiz_a_lo is
                when "00100" =>
                    v_sram_be_l := "1101
                when "00101" =>
                    v_sram_be_l := "1110
                when "00110" =>
                    v_sram_be_l := "0111
                when "00111" =>
                    v_sram_be_l := "1011
                when "01000" =>
                    v_sram_be_l := "1100
                when "01010" =>
                    v_sram_be_l := "0011
                when others =>
                    v_sram_be_l := "0000
            end case;
            -- Aligned Data Transfer -
        end if;
    else
        sram_state <= s0;
        SRAM_BE_L <= "1111";
        SRAM_CS_L <= '1';
        SRAM_WR_L <= '1';
        SRAM_RD_L <= '1';
        SRAM_DONE_L <= '1';
```

```
            end if;
        end if;
    end process;

end BEHAVIOR;
```

(4) 제어를 위한 주소 설정

CPU에서는 주소와 기타 제어 신호를 이용하여 필요한 부분에 대한 쓰거나 읽는 동작을 한다. 이러한 동작을 위해 FPGA2에서는 제어 신호를 생성하며, 제어 신호 생성시 CPU 제어 신호와 주소를 가지고 각 부분에 대한 제어 신호를 생성하게 된다. 그러므로 각 부분에 대한 주소를 정의하고 주소에 따라 제어 신호를 합성한다. 먼저 FPGA2에 있는 제어 신호 주소는 다음과 같다.

```
constant EPROM_ADDR_HI      : "111111111111";  -- FFF
constant LCD_DATA_ADDR_HI   : "111000000000";  -- E00
constant LCD_CTRL_ADDR_HI   : "111000000001";  -- E01
constant WD_ADDR_HI         : "111000000010";  -- E02
constant CAN1_ADDR_HI       : "110100000000";  -- D00
constant CAN2_ADDR_HI       : "110100000001";  -- D01
constant SCC1_ADDR_HI       : "110000000000";  -- C00
constant SCC2_ADDR_HI       : "110000000001";  -- C01
constant SCC3_ADDR_HI       : "110000000010";  -- C02
constant LED0_ADDR_HI       : "101100000000";  -- B00
constant LED1_ADDR_HI       : "101100000001";  -- B01
constant REG_ADDR_HI        : "100100000000";  -- 900
constant SRAM_ADDR_HI       : "000000000000";  -- 000
```

기본적으로 각 주소는 std_logic_vector(0 to 11) 타입이며, 12bit 만을 가지고 각 부분에 대한 제어 신호를 생성하며, 상세한 주소는 기타 나머지 주소를 이용하여 각 부분에서 제어한다.

간단한 예로서 WatchDog Clear 명령에 대해 처리하는 것을 FPGA 코드로 살펴보면, 먼저 i_clame_l 라는 신호와 WatchDog에 해당하는 주소인가를 판별하여

i_wd_sel 신호를 생성한다. 생성된 i_wd_sel 신호는 WD_TIMER 에 입력 신호를 주고 WD_TIMER에서 처리된 출력 신호를 이용한다. 해당 VHDL 코드는 (i_wd_sel <= '1' when (A_HI = WD_ADDR_HI and i_claim_L = '0') else '0' ;) 이다.

각 해당 부위에 대한 점검 및 제어를 위해서 필요한 주소를 일일이 사용하는 것도 가능하지만, 603 C 프로그램에서 이들에 대한 제어의 편의성을 위해서 다음과 같이 주소를 정의하여 사용하였다.

나 FPGA 통신 로직 구현

(1) SCC 로직 구현

본 프로젝트에서 통신 link는 FPGA로 구현하였으며, 정상 독작을 확인 하였다.
SCC 는 I0 를 전담하는 FPGA에서 구현하였으며, 데이터를 송신하는 부분과 수신하는 부분을 독립적으로 구현하였다. 또한 송신부에서는 송신 클럭을 따로 생성하였으며, 수신 부에서도 수신 클럭을 따로 생성하였다. 클럭을 따로 생성한 이유는 송신 부는 송신을 주관하기 때문에 데이터를 송신 시점을 클럭에 동기화 하여 송신하고, 수신 부는 수신 부에서 들어오는 신호를 감지하고 그때부터 동기화된 샘플링 클럭을 생성하여야 하기 때문에 두 클럭이 동기화되어 동작 작동하지 않기 때문이다.

(2) SLIP 로직 구현

SLIP PROTOCOL은 Serial Line을 통하여 프레임 단위로 명령을 주고받는 하나의 프로토콜이다. 일반적으로 SCC를 통해서 데이터를 보내는 경우, 문자 송수신시 SCC에서 발생하는 잦은 인터립트 처리를 하여야 하며, 이로서 CPU의 효용성을 떨어뜨린다. 그러므로 SLIP 단위로 데이터 열을 FPGA를 이용하여 처리함으로써 데이터 열을 송수신 과정에서 CPU를 보다 효율적으로 이용할 수 있다.
SLIP 프로토콜은 바이트 열에 대해서 특수한 시작 문자(0xC0)와 특수한 종료 문자 (0xC0) 와 이들을 변경할 특수문자(0XDB) 를 이용하여 처리한다. 그러므로 프레임의 처음에 시작 바이트 0xC0를 보낸다. 보낼 데이터에 해당하는 바이트가

0xC0 이면 0xDB, 0xDC 두 바이트를 보내며, 0xDB 인 경우 0xDB, 0xDD를 보내고 그렇지 않을 경우 해당 바이트를 전송한다. 그리고 마지막으로 0xC0를 보내서 하나의 프레임을 형성한다.

Slip Protocol에 의해서 데이터를 송신시 CPU에서는 해당 데이터 프레임에 대해서 FPGA에SLIP에게 시작을 알려주고 FPGA Q에 해당 프레임을 적고, 다시 FPGA에 SLIP 종료를 알려주면, FPGA SLIP TX 모듈에서 C0 와 해당 SLIP 데이터와 C0를 전송하므로 FIFO를 처리하면 된다. 하지만 수신 과정은 입력되는 데이터에 대해서 SLIP에 해당하는 프레임을 구성하고 절절히 구성된 프레임에 대해서 FPGA에서 수신된 SLIP프레임에 대한 크기와 해당 프레임을 저장한 후에 CPU에서 필요한 프레임 양의 데이터를 처리한다.

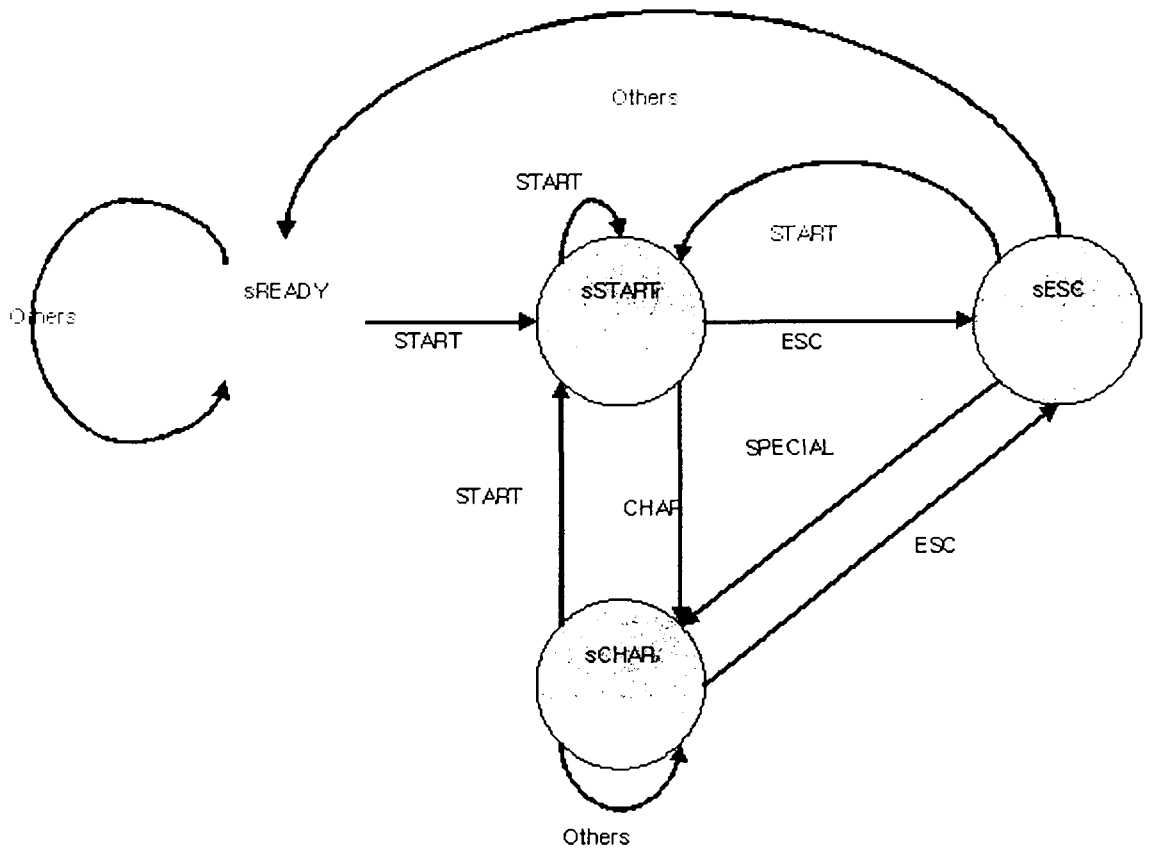


그림 91 SLIP 수신 PROTOCOL

그림은 SLIP Protocol 구현에서 하나의 프레임을 받는 과정을 기술한다.

SLIP RX를 위한 상태로는 sREADY, sSTART,sESC,sCHAR 을 정의하였다.

sREADY는 오류 상태나, 기타 초기화 상태를 의미한다. 이 상태에서 START신호인 C0 신호를 입력을 받으면, sSTART 상태가 된다. 이 상태는 시작 신호를 받은

상태로서 수신을 시작하는 상태이다. 이 상태에서 일반 문자가 오면 상태는 sCHAR 상태가 된다. 이 상태는 데이터를 받은 상태이다. 이 상태에서 ESC를 수신하거나, sSTART에서 ESC를 수신하면 상태는 sESC 상태가 된다. 이 상태는 특수 문자를 받은 상태이며, 정의된 특수 문자를 기다린다. 만일 정의되지 않은 특수 문자가 오면 오류 상태가 된다. 하지만 정의된 특수 문자가 오면 특수문자에 해당하는 문자를 수신한 것이므로 sCHAR상태가 된다. 각각의 모든 상태에서 sSTART신호가 오면 초기화 준비 상태인 sSTART 상태로 된다. 이는 만일 통신시 읽어버린 세잔 및 종료 문자에 대해서 재 준비하는 상태이다. 특히 sESC에서 오는START 문자는 오류를 나타내고 있다. 하지만, 다음의 문자를 받아들이기 위해서 sSTART 상태가 되는 것이다. 그러므로 정상적으로 SLIP 프로토콜에 의해 SLIP 프레임을 받은 경우는 sCHAR에서 START 문자를 받은 경우 밖에는 없다. 특히 만일 SLIP 프레임의 크기가 제한된 경우에는 sCHAR에서 일반 문자를 계속 받을 경우 길이와 비교해서 길이보다 클 경우 sREADY 상태로 변경해 주어야 한다.

다음은 상태도에 해당하는 상태 변환 표를 나타내고 있다. 현재 상태에 입력 문자에 따라 다음 상태를 표시하며, 해당 Operation을 표시하고 있다.

상태	입력 문자	다음 상태	비고
sREADY	C0	sSTART	Clear
sREADY	OtherChar	sREADY	[Clear]
sSTART	C0	sSTART	[Clear]
sSTART	ESC	sESC	
sSTART	OtherChar	sCHAR	Input OtherChar
sESC	C0	sSTART	Clear
sESC	SpecialChar	sCHAR	Input Responding SpecialChar
sESC	OtherChar	sREADY	Clear
sCHAR	C0	sSTART	Received One Slip Packet
sCHAR	ESC	sESC	
sCHAR	OtherChar	sCHAR	Input OtherChar

그림 92 SLIP RX Protocol 상태표

Slip Rx 에 대한 해당 State 변화 부분을 VHDL로 표시하고 있다. 해당 부분은

rxState를 case 문을 이용하여 각 상태와 입력에 따라 다음 상태를 표시하고 있다.

```
case rxState is
      when sReady => if( rx_data = START_CHAR  ) then  rxState <= sStart ;  end if;
      when sStart => if( rx_data = START_CHAR  ) then  rxState <= sStart ;
                     elsif( rx_data = ESC_CHAR    ) then  rxState <= sEsc ;
                     else                    rxState <= sChar  ;
                     end if;
     when sEsc =>  if( rx_data = START_CHAR  ) then rxState <= sStart ;
                     elsif( rx_data = S_END) then rxState <= sChar  ;
                     elsif(rx_data = S_ESC) then rxState <= sChar;
                     else                          rxState <= sReady ;
                     end if;
     when sChar  => if( rx_data = START_CHAR  ) then rxState <= sStart ;
                     elsif( rx_data = ESC_CHAR    ) then rxState <= sEsc   ;
                     else                                rxState <= sChar ;
     end if ;
end case;
```

그림 93 SLIP RX VHDL 코드 코어

(3) CAN(Controller Area Network) 로직 구현

(가) 개요

CAN(Controller Area Network)은 원래 자동차내의 각종 계측제어 장비들 간에 디지털 직렬 통신을 제공하기 위하여 1988년 Bosch와 Intel에서 개발된 차량용 네트워크 시스템으로, 1993년도에 ISO에서 국제표준 규격으로 제정되었다. CAN은 다른 자동화 통신망들에 비하여 가격 대 성능비가 우수하며, 지난 수년간 차량내의 열악한 환경에서 성공적으로 동작되어 신뢰도가 검증된 통신망이다. CAN 칩(chip)은 이미 인텔, 모토롤러, 필립스, NEC, 히타치, 시멘스 등 많은 여러 회사에서 개발했다. CAN은 마스터/슬래이브(master/slave), 다중 마스터(multiple master), 피어투 피어(peer to peeer)등을 지원하는 매우 유연성 있는 네트워크이며, 공장의 열악한 환경이나 고온, 충격이나 진동, 노이즈가 많은 환경에서도 잘 견딜 수 있다. 이러한 장점들로 인하여 최근에 와서 CAN은 공장자동화와 공정의 분산제어등의 각종 산업설비에서 제어 및 자동화 관련 장비들 간에 데이터 교환을 위한 통신망으로 널리 사용되고 있다.

① CAN의 장점

CAN 프로토콜에는 여러 장점이 있는데 첫 번째로, 표준 통신 프로토콜이므로 다양한 업체에서 제작된 서브들을 공동의 네트워크에 인터페이스 시키는 작업을 쉽고 경제적으로 수행할 수 있다.

두 번째로 CAN 프로토콜은 수백만의 메시지 확인자를 지원하고 복잡한 메시지 방식을 사용할 수 있는 유연성을 가지고 있다. 또한 에러 발견과 응답은 CAN칩 자체에서 처리되므로 그에 따른 처리가 현저히 감소된다.

세 번째로 CPU에서 주변기기로 통신작업이 이양되었기 때문에 CPU는 시스템 태스크만 전적으로 실행할 수 있다.

네 번째로 다중 채널식 통신법이기 때문에 포인터간의 와이어 작업을 줄여 와이어 크기를 대폭 줄일 수 있다. 마지막으로 표준 프로토콜이므로 시장성이 뛰어나고 이로 인해 많은 업체들이 경쟁적으로 CAN칩을 제작하고 있으며, 비용 또한 비교적 저렴하다. CAN 프로토콜은 호스트 CPU에 인터케이스된 CAN 컨트롤러 칩이나, 호스트 CPU에 장착된 CAN 주변장치에서 실행된다.

CAN은 CSMA/NBA(Carrier Sense Multiple Access with Non-destructive Bitwise Arbitration)라는 메시지 전송 메커니즘(mechanism)을 가지고 있다. CAN의 데이터 전송 메커니즘은 IEEE802.3 CSMA/CD 프로토콜과 유사하다. 즉, 각 노드는 데이터를 전송하기 이전에 버스의 상태를 감지하며, 버스의 상태가 비활성일 때 준비된 메시지를 전송한다. CSMA/CD에서는 두 개 이상의 노드가 동시에 메시지를 전송하면 메시지 충돌이 일어나서 전송된 메시지가 모두 손실된다. 그러나 CAN에서는 전송되는 메시지가 11비트의 식별자(Identifier)를 가지고 있으며, 식별자를 통하여 우선순위가 높은 메시지가 전송되도록 한다. 즉, 두 개 이상의 노드가 동시에 메시지를 전송하면 각 메시지는 서로 식별자를 1 비트씩 비교하여 제일 높은 우선순위의 메시지(즉, 가장 낮은 식별자 값을 가진 메시지)는 전송되고 낮은 우선순위의 메시지들은 전송이 중단된다. 버스에서는 '0' 비트가 '1' 비트에 대해 우세한 성질을 가지고 있다. 즉 '0' 비트는 'dominant'한 값('d' 비트)이라고 하며 '1' 은 'recessive' 한 값('r' 비트)이다. 전송하는 노드는 한 비트를 보낼 때마다 버스를 모니터링 한다. 어떤 노드가 'r' 비트를 보냈는데 버스를 모니터링한 결과 'd' 비트가 검출되었다면 이는 버스 내의 다른 노드가 자기보다 높은 우선순위의 메시지를 전송하고 있는 것이므로 그 노드는 메시지의 전송을 즉시 중단하고 수신모드로 전환한다.
전송을 중단한 노드는 버스의 상태를 계속 감지하여, 버스가 다시 비활성 상태가 되면 자동적으로 다시 메시지의 전송을 시작한다.

② CAN 데이터 프레임

CAN에 의해 데이터가 교환될 때 어떠한 스테이션도 주소화되지 않고 메시지의 내용(ex, rpm 혹은 엔진온도등)은 통신망에서 유일하게 존재하는 메시지 ID(identifier)에 의해서 정해진다. ID 는 메시지의 내용 뿐 아니라 우선순위도 결정하는데 이것은 여러 개의 스테이션이 동시에 버스를 액세스하려고 할 때 버스 할당을 위하여 중요하다.
CAN에서는 내용지향(content-oriented) 어드레싱 구조에 의해 구성의 융통성을 꾀할 수 있으며, 새로운 스테이션이 순수한 수신기라면 어떠한 하드웨어나 소프트웨어의 변경 없이 현존하는 CAN 통신망에 붙일 수 있다. 또한 데이터 전송 프로토콜이 물리적인 도착지의 주소를 필요로 하지 않기 때문에 modular electronics의 개념을 지원하고, 또한 broadcast나 multicast와 같은 다중 수신과 분산처리의 동기화를

허용하여 여러 개의 콘트롤러에서 정보로 필요로 하는 측정값들이 통신망을 통해 전송될 수 있으므로 각각의 콘트롤러가 자신만의 센서를 가지고 있을 필요가 없다. 실시간 처리에 있어서 통신망에서 교환되는 메시지의 긴급성은 메시지의 내용에 따라 매우 다를 수 있다. 예를 들어 엔진 부하와 같이 빠르게 변하는 것은 엔진 온도와 같이 상대적으로 느리게 변화하는 것보다 좀더 자주 그리고 좀더 작게 지연이 일어나도록 전송되어져야 한다.

전송되는 메시지의 우선순위는 해당 메시지의 ID에 의해 결정되어지는데, 시스템을 디자인하는 동안 결정되며 이진법에 의해 표현되고 동적으로 변할 수 없으며 가장 낮은 이진수를 갖는 ID가 가장 높은 우선순위를 갖는다.

버스엑세스 충돌은 버스레벨의 비트를 관찰하고 있는 각각의 스테이션에 의해 메시지 ID에서 bitwise arbitration을 통해 해결되는데 Dominant상태(0)가 recessive상태(1)를 덮어쓰는 "Wired and" 메커니즘과 동일한 메커니즘에 의해 버스 할당의 경쟁은 해결된다. 경쟁에서 패배한 모든 스테이션들은 가장 높은 우선순위를 갖는 메시지의 수신기가 되고 버스가 다시 사용가능할 때 까지 전송을 재시도 하지 않는다.

CAN은 분산 버스 액세스 컨트롤을 갖는 non-destructive 버스 액세스에 의해 높은 데이터 전송을 가능하게 하는 traffic-dependent 버스 할당 시스템으로 구현되고, 전송 요구가 있는 스테이션들에게만 버스가 이용되기 때문에 버스 arbitration 절차의 효율성이 향상된다. 또한 메시지 전송 요구는 시스템 전체에 대한 메시지의 중요한 순서에 따라 처리되는데, 이것은 특히 과부하 상태에서 장점을 갖는다. 그리고 버스 액세스는 메시지에 기반을 두고 우선순위가 정해지기 때문에 실시간 시스템에서 낮은 잠복시간(latency time)의 보장을 가능하게 해준다.

③ CAN 응용

CAN을 사용하는 대표적인 통신망으로는 DeviceNet, SDS, CAN Kingdom, CANopen/CAL 등이 있으며, 이들은 모두 데이터 링크 계층으로 CAN을 사용하나 응용계층은 서로 다른 프로토콜을 채택하고 있다.

DeviceNet은 Rockwell/Allen-Bradley에서 개발된 응용계층으로, 현재 폭넓은 산업 자동화 현장에서 사용되고 있으며, 기본적으로 개방 버스시스템을 채택하고 있어 모든 모듈은 같은 우선순위로 버스를 사용할 수 있으며 단 몇 개의 규정만 지키면

된다.

SDS(Smart Distributed System)는 미국 Honeywell 에서 개발하였으며, I/O 장비(on/off 스위치,

근접센서 등)와 PLC의 연결에서 사용되며, 기본적으로는 마스터(master)와 원격 I/0 사이의 일대일(point-to-point) 연결을 기반으로 하고 있다.

CAN Kingdom은 스웨덴 기업인 Kvaser AB가 제공하고 있는 응용계층을 디지털이나 아날로그를 위한 프로파일(porfile) 등을 포함하지 않고 연결되거나 제어되는 시스템을 집약하는 프로토콜을 사용하고 있다. 모든 CAN 우선순위나 ID등을 각자 모듈이 갖고 있는 것이 아니라 중심이 되는 노드 즉, King이라는 것이 갖고 있게 된다.

CANopen은 8바이트 이상의 긴 데이터(16바이트)를 전송하는데 효과적이다. 응용계층이 OSI 모델의 형태를 취하고 있으며 이로 인해 데이터 전송을 위해 표준화된 서비스, 프로토콜, 네트워크 관련 작업의 수행과 계층관리를 위한 기능 등을 제공한다.

위의 모든 응용계층들은 모두 ISO 11898 CAN 통신 프로토콜과 CAN 특성의 회로를 기반으로 하여 이루어지고 있다. 그러나 CAN Kingdom이 SDS나 DeviceNet과 가장 특이한 점은 하나의 노드가 시작할 때 시스템을 구성한다는 점이다.

최근 산업사회에서는 각종 이종의 장비, Unit 등에 정보를 서로 공유하고 신속히 전달하면서 감시하는 네트워크 시스템이 사용되면서 설비확장, 유지보수, 생산성, 배선의 절감으로 인한 경제성 등에 있어서 많은 효과를 보고 있다.

최근 일렉트로닉스의 진전에 따른 전기, 전자 부품의 증가와 자동차 전자 시스템이 기능 향상 등에 의해 와이어 하니스는 보다 복잡화 되어 중량의 증대와 볼륨(전선 다발 지름)의 비대화를 초래하고 있다. 이것은, 수mm의 자동차 스페이스, 수g의 자동차 중량을 중요시하는 자동차 설계에 있어서 심각한 문제로 대두된다. 또한 전장품의 양이 많아지게 됨에 따라 각종배선이 늘어나서 조립시 어려움이 따르고, 각제품마다 각각의 제어기가 있어 같은 기능도 중첩되는 결과가 되어 비효율적인 시스템으로 구성되어지고 있다. 그래서 전자 시스템의 제어가 보다 고도화되어 복수의 시스템이 집적화. 복합화 하여 토탈 네트워크로서 기능 이 필요하게 된다.

그리고 중소형의 작업장에서도 자동화 시스템의 설계시 Network의 효율성 및 장점을 인지함에도 불구하고 고가의 시설투자비와 복잡화되어진 Network환경으로 인해

엄두를 내 지 못하는 실정이므로 간단하면서도 가격이 저렴한 Network System의 필요성이 대두되고 있다.

특히 자동자의 경우 차량내 제어장치간 배선의 증대로 인해 네트워크 구성이 절실히 필요하게 되었고 이를 위해 IN VEHICLE NETWORK용 프로토콜을 만들게 되었다.

차량용 네트워크 프로토콜로 제안되어 요즈음 차량에 사용중인 프로토콜이 J1850과 ISO9141-2이고 이 네트워크 프로토콜은 통신 속도가 10.4K BIT/S 이다. 그리고 Network의 고속클래스로 ISO 규격인 CAN(Controller Area Network)은 자동차 내에 응용하기위해 고안된 네트워크 시스템으로써 이 네트워크 시스템을 이용하게 되면, Twist 된 2라인을 데이터 버스로 이용하기 때문에 자동차의 배선절감에 따른조립비를 감소시킬 수 있고, 커넥터, 접점이 줄어들어 시스템의 안정화를 기할 수 있으며, 여러 다른 Unit끼리 손쉬운 접속을 할 수 있어 단순화된 시스템을 만들 수 있다.

공장 자동화 분야에도 CAN을 적용하면 가격이 저렴하고 간단하게 시스템에 구현할 수 있으므로 중소형의 작업장에서도 각종 단일 Unit들 간에 네트워크화 하여 시스템의 효율화, 관리의 용이, 생산성증대에 많은 효과를 볼 수 있을 것이다.

④ 국내외 기술 현황

CAN(Controller Area Network) 시스템은 독일 Bosch사에서 처음 고안하여 자동차에 적용하기 위하여 만든 시스템으로 현재는 ISO 표준규격(ISO11898)으로 만들어져, 세계 선진국에 서는 자동차전장시스템에 적용하여 이미 ECU, ABS시스템은 CAN기본 사양으로 지원되고 있으며, 에어백, 미러 Control, 에어 컨디셔너등에도 확대 적용될 전망이다.

Intel, Motorola, Philps등 세계반도체 회사에서도 기존의 마이크로 콘트롤러에 CAN Device를 내장한 칩과 CAN Transceiver등의 칩을 개발하여 판매하고 있고, Vector Informatik회사 등 에서 CANalyzer, CANoe 등의 디자인 Tool 등을 제공하고 있다.

현재 자동화에서는 CAN을 이용한 각종 Application들이 상용화 되어지고 있는데 Sensor/Actuator Link를 위한 SDS(Honeywell), Acturator/PLC Link를 위한 Device Net, Distributed Control를 위한 CAN Kingdom(Kvasar)등이 있다.

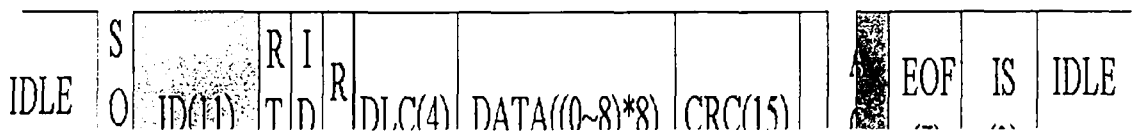
국내에는 자동화 제품으로 LG에서 미국Honywell의 SDS를 Smart Net라는 상표로 판매하고 있고, 현대 자동차에서는 1996년도 부터 CAN 개발팀이 구성되어 전장시스템을 연구 개발하고 있다.

⑤ 캔(CAN) 칩셋

CAN을 구성하기 위해서는 CAN프로토콜을 처리할 컨트롤러가 필요한데 이 기능을 하기위해 만든 것이 SJ1000이나 인텔의 82527같은 프로토콜 컨트롤러이고 이 칩들은 CPU를 통해 컨트롤 된다.
이후 프로토콜 컨트롤러와 CPU가 하나로 합쳐진 칩들이 개발되었는데 이런 것들에는 모토롤라의 68hc11 이나 인텔의 196계열, 그리고 TI의 24X계열 DSP등이 있다. 이런 CAN칩들을 INTERGRATED CAN CHIP이라 부르고 전자의 컨트롤러 CHIP은 STAND ALONE CAN CHIP이라 부른다.

(나) CAN 구현

① CAN 메시지 송수신을 위한 상태



본 연구의 Can의 로직 구현을 위한 송수신 메시지의 상태를 나타내면 그림과 같이 BUS IDLE 상태에 있다가 메시지 시작 신호를 인지하던가, 메시지가 보낼 것이 있으면 SOF(Start Of Frame)상태가 된다. 이후 ID를 조정하는 상태로 변화되며, 메시지 정보에 전송 상태를 거친 후 데이터 길이를 송신하며 최대 8바이트의 데이터를 전송하고 해당하는 CRC를 전송하며, ACK 신호를 서로 송수신하고 프레임 종료를 위한 절차를 받는 상태가 된다.
각각의 상태를 기술하면, SOF(Start of frame)는 한 개의 'd' 비트로 구성되며, 모든 노드들은 처음 전송을 시작하는 노드의 SOF에 의해 만들어지는 'leading edge'에서 동기화된다. 중재 필드(arbitration field)는 11 bits의 식별자(Identifier)와 1 bit의 RTR(Remote Transmission Request) 비트로 구성되는데, 둘 이상의 스테이션에서 동시에 메시지를 전송할 때 발생하는 메시지간의 충돌은 식별자(Identifier) 비트를 비교함으로써 해결한다. CAN 규격에는 다른 제조업체로부터 생산된 콘트롤러

사이에서의 호환성을 보장하기 위해서 CAN 시스템 구축시 11 bit Identifier의 MSB(Most Significant Bit)(ID10~ID4)가 모두 'r'인 경우, 즉 1111111XXXX(여기서 X는 '0' 또는 '1')인 경우는 사용하지 말도록 규정하고 있다. 이것은 사용자가 사용할 수 있는 식별자(Identifier)의 개수가 2,032개(=211-24)이며, 하나의 로컬 네트워크에서 생성될 수 있는 메시지의 수는 2,032개라는 것을 의미한다. RTR 비트는 데이터 프레임인지 원격 프레임인지를 구별하게 한다. 데이터 프레임이면 'd'비트를 가지고 원격 프레임이면 'r'비트를 가진다.

제어 필드(control field)는 6 bits로 구성되는데 IDE(Identifier extension) 비트는 표준 프레임

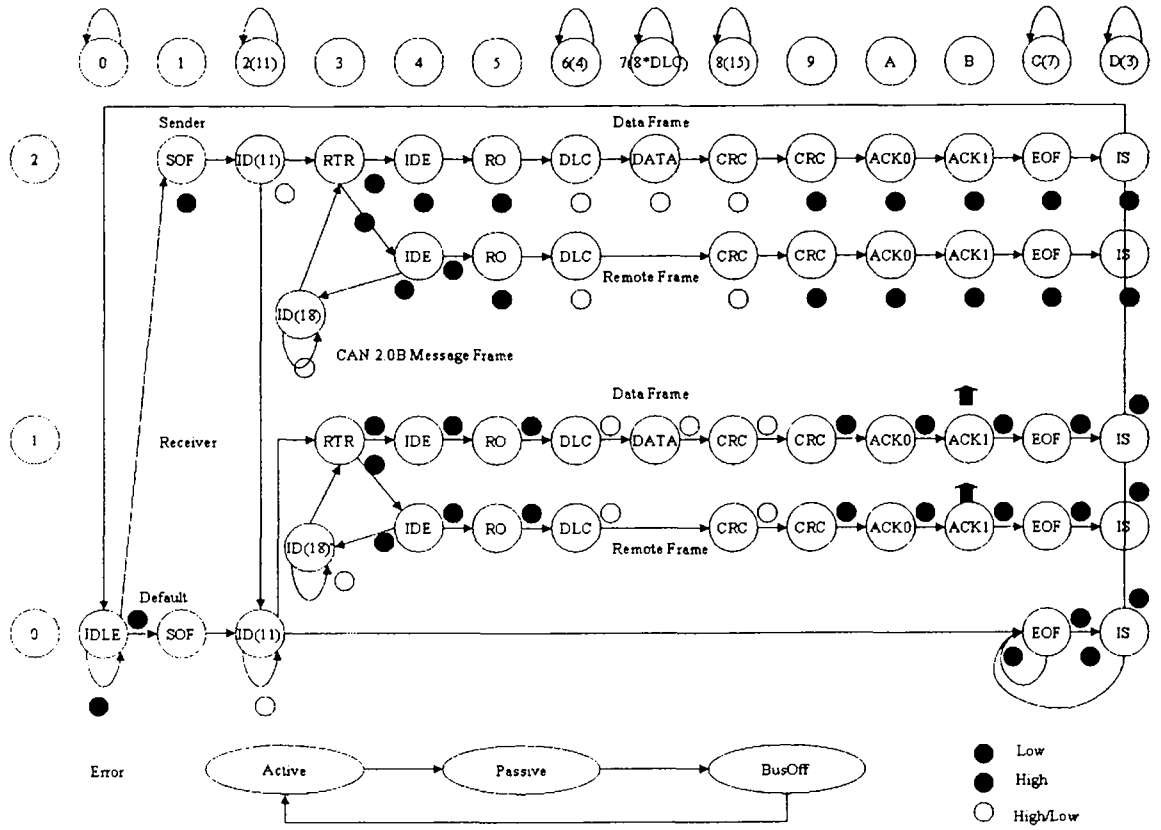
(IDE='d' 비트)과 확장 프레임(IDE='r' 비트)을 구별한다. r0 비트는 예비 비트(reserved bits)로 두고, 나머지 4 bits는 DLC(Data Length Code)로서 데이터 필드의 데이터 길이를 알려 주는 코드이다.

데이터 필드가 0 byte에서 8 bytes까지 가능하기 때문에 이진수 '0000'에서 '1000'까지의 수만 사용가능하고 그 외의 숫자는 사용할 수 없다.

CRC(cyclic redundancy check) 필드는 15 bits의 CRC와 'r' 비트로 표시되는 1 bit의 CRC 구획문자(delimiter)로 구성된다. 수신 노드에서 데이터 프레임을 수신하면 먼저 스터핑 비트(stuffingbits)를 없애고 난 뒤에 CRC를 통하여 SOF에서 데이터 필드까지의 에러 유무를 체크하게 된다. 15 bits의CRC는 127 bits보다 작은 비트카운트(bit counts)를 가진프레임에 적합한데, CAN은 최대 108 bits의 프레임이므로 에러 체크에 적당하다.

ACK 필드는 1 bit의 ACK 슬롯과 1 bit의 ACK 구획문자로 구성되는데, 메시지를 올바르게 받은 수신 스테이션이 ACK 필드를 받는 바로 그 순간에 ACK 슬롯의 비트 값을 'r'에서 'd'로 세팅하게 된다.

만약 두 비트 모두 'r' 비트가 검출되면 데이터 전송이 올바르게 이루어지지 않았음을 의미한다. 마지막으로 EOF(end of frame)는 7 bits의 'r'로 구성되고, 메시지의 전송이 끝났음을 알린다.



그림은 CAN의 로직 구현을 위해 세부 상태 변환도를 표시한다. 파란색 원은 dominant 한 값인 0 에 해당하는 비트를 송신자는 송신하고 수신자는 수신한 것을 의미하며, 빨간색 원은 recessive한 값인 1 에 해당하는 비트는 송신하거나 수신한 경우를 표시하며, 노란색은 각 상태에 따라 결정된 dominant나 recessive 비트를 송신이나 수신하는 것을 표시한다. 또한 파란색 화살표는 수신 측에서 해당 ACK를 송신하는 것을 표시한다.

그림에서 보면 크게 5가지 수평 라인을 통한 메인 상태가 존재한다. 그것은 송신노드 와 수신노드, 방관노드에 해당하며, 송신 노드와 수신 노드는 각각 데이터를 주고받는지 또는 Remote Frame 을 주고받는 하는 모드에 해당한다.

전체의 흐름도를 기술하면, 우선 모두 IDLE 상태에 있게 된다. 이때 송신할 메시지가 발생하면 송신자가 되어 버스에 SOF 상태가 되면서 버스에 'd' 값을 출력한다. 이후 자신의 메시지 ID 11비트를 송출한다. 이때 만일 송신한 id에 해당하는 비트와 수신한 id 비트가 다르지 않을 경우 계속 송신 노드로 있게 되고, 그렇지 않을 경우

수신노드가 되어 해당 메시지를 수신하게 된다. 송신 노드로 있을 경우 해당 메시지의 종류에 따라 RTR 값이 'd'에 해당하면 데이터 프레임에 해당하는 과정 상태도 변화를 추구하며, 그렇지 않을 경우 다음의 IDE 신호에 따라 Remote 프레임이나, CAN2.0B의 메시지 프레임이 된다. 데이터 프레임일 경우 IDE와 RO 에 해당하는 신호를 보내고 4비트에 해당하는 메시지 바이트 수에 해당하는 길이를 전송한다. 이후 길이에 해당하는 메시지 데이터를 전송한다. 이후 15비트에 해당하는 CRC를 전송하고, CRC 종료에 해당하는 비트를 전송한다. 이후 1비트의 ACK 신호를 전송하고, 수신 노드로부터의 1비트 ACK를 기다려 체크한다. 이후 7비트의 EOF에 해당하는 비트를 전송하고 다음 메시지를 위해 3비트의 메시지를 쉰다.

그림에서 아래에 있는 타원은 CAN 제어기의 오류 대한 상태를 표시하고 있다.

CAN의 오류 상태는 3가지가 있으며 Error Active, Error Passive, BUS OFF 상태이다.

Error Active 상태는 정상적으로 메시지를 송수신할 수 있는 경우로 tx 오류 개수가 128보다 작고, rx 오류 개수가 128 보다 작은 경우에 해당한다. 이 상태에서는 오류 발생시 active error flag를 전송하여 메시지 상에 오류가 발생했음을 알린다.

Error Passive 상태는 에러가 종종 발생한 경우로 정상적으로 메시지를 송수신하는 상태로서 ErrorActive가 아니면서 tx 오류 갯수가 256 보다 작은 경우로 passive error flag를 전송하는 한다. BUS OFF 상태는 심각한 오류가 발생한 경우로, Reset 되기 전까지 메시지 송수신을 하지 않는다. 이 경우는 tx 오류 발생수가 256 개 이상인 경우에 해당한다.

CAN 메시지 전송시 발생하는 오류는 BIT 오류, 메시지 오류, ACK 오류가 있다. BIT 오류는 비트 Stuffing 오류와 Bit 오류가 있다. Bit Stuffing 오류는 메시지 전송시 11111 다음에 0 가 안 올 경우에 해당하며, Bit 오류는 보낸 신호와 수신된 신호가 다를 경우에 해당한다. 메시지 오류의 경우는 CRC 오류와 프레임 오류가 있으며, CRC 오류는 해당하는 CRC가 맞지 않을 경우이며, 프레임 오류는 프레임에서 정해진 값 이외의 Bit 값을 수신 했을 경우에 해당한다. 그리고 ACK 오류는 정해진 ACK 신호를 수신 하지 못하였을 경우에 해당한다.

이러한 오류의 발견은 송신 노드와 수신 노드 모두에서 발견할 수 있으며, 오류 발견시 오류 프레임을 즉시 전송하고 이러한 오류 프레임을 받으면 모드 노드에서 전송을 취소하며, 제어기 상태를 재설정하며, 이에 따라 메시지 재 전송을 한다.

현재상태		입력	샘플링	입력에 의한 판단	내부값판단	==>	바뀐상태		출력			
-Reset							-Reset		H			
Status	Send/Recv	Sub0	Sub1	Sub2	기타		NextStatus	Send	Sub0	Sub1	Sub2	설정
IDLE		L					SOF	R				
IDLE					Data		SOF	S	L			
SOF				L			ID10					
SOF				H			IDLE					ERROR_SOF
ID1i	S				I < 10		ID1I	S	Di			
ID1I	S			X==DI			ID1I+1	S				
ID1i	S			X!=DI			ID1I+1	R				
ID1I	R			DI			ID1I+1	R				IDI=DI
ID1i					I == 10		RTR					
RTR	S				RTR==L		RTR		L			
RTR	S				RTR==H		RTR		H			
RTR	R			L			IDE					RTR=L
RTR	R			H			IDE1					RTR=H
IDE	S						IDE	S	L			
IDE				L			RO					
IDE				H			ERROR_IDE					ERROR_IDE
IDE1	S				IDE1==L		IDE1	S	L			
IDE1	S				IDE1==H		IDE1	S	H			
IDE1	S			X != d			ERROR_IDE1					ERROR_IDE1
IDE1				L			RO					
IDE1				H	Ver2.0B		ID20					
ID2i	S				I < 17		ID2i	S	Di			
ID2i	S			X==Di			ID2i+1	S				
ID2i	S			X!=Di			ID2i+1	R				
ID2i	R			Di			ID2i+1	R				ID1i=Di
ID2i					I == 17		RTR1					
RTR1	S				RTR1==L		RTR1		L			
RTR1	S				RTR1==H		RTR1		H			
RTR1	R			L			IDE					RTR1=L
RTR1	R			H			IDE2					RTR1=H
IDE2	S				IDE2==L		IDE2	S	L			
IDE2				L			RO					
IDE2				H			ERROR_IDE2					ERROR_IDE2
RO	S						RO	S	L			
RO				L			DLC0					
RO				H			ERROR_RO					ERROR_RO
DLCi	S				I<3		DLCi	S	di			
DLCi	S			X==di			DLCi+1	S				
DLCi	S			X!=di			ERROR_DLC0					
DLCi	R			X			DLCi+1	R				DLCi=X
DLCi					I==3		Data0					
DLCi					RTR==H || RTR1==H							
Datai	S				I < DLC*8-1		Datai	S	di			
Datai	S			X==di			Datai	S				
Datai	S			X!=di			ERROR_DATAi					
Datai	R			X = di			Datai					Datai=X
Datai					I == DLC*8-1		CRC0					
CRCi	S				I < 14		CRCi	S	ci			
CRCi	S			X==ci			CRCi+1	S				
CRCi	S			X!=ci			ERROR_CRCi					
CRCi	R			X==CALC_CRCi			CRCi+1	R				
CRCi	R			X!=CALC_CRCi			CRCi+1	R				ERROR_CRCi
CRC14	S						CRC14	S	H			
CRC14	S			X==H			ACK0	S				
CRC14	S			X!=H			ERROR_CRC15					
CRC14	R			X==H			ACK0	R				
CRC14	R			X!=H			ACK0	R				ERROR_CRC1
ACK0	S						ACK0	S	H			
ACK0	S			X==H			ACK1	S				
ACK0	S			X!=H			ERROR_ACK0	S				
ACK0	R			X==H			ACK1	R				
ACK0	R			X!=H			ERROR_ACK0	R				
ACK1	R						ACK1	R	L			
ACK1	S			X==L			EOF0	S				
ACK1	S			X!=L			ERROR_ACK1					
ACK1	R			X==L			EOF0	R				
ACK1	R			X!=L			ERROR_ACK1					
EOFi	S				I < 6		EOFi	S	H			
EOFi				X==H			EOFi+1					
EOFi				X!=H			EOFi+1					ERROR_EOFi
EOFi					I == 6		ISO					
ISi	S				I < 2		ISi	S	H			
ISI				X==H			ISi+1					
ISi				X!=H			ISi+1					ERROR_ISi
ISi					I == 2		IDLE					

그림은 해당 세부 상태로들 상태표로 만든 것이다.

각각의 필드는 현재 상태, 송수신 모드, 시작시 버스 값, Sampling 시 버스값, 판단 조건에 따른 다음 상태, 송수신 모드, 시작시 버스 출력 값, 해당 설정값을 표시한

다.

예를 들어 제일 윗단의 Reset상태는 계속 Reset이며, 버스 값은 High 또는 TriState를 나타낸다.

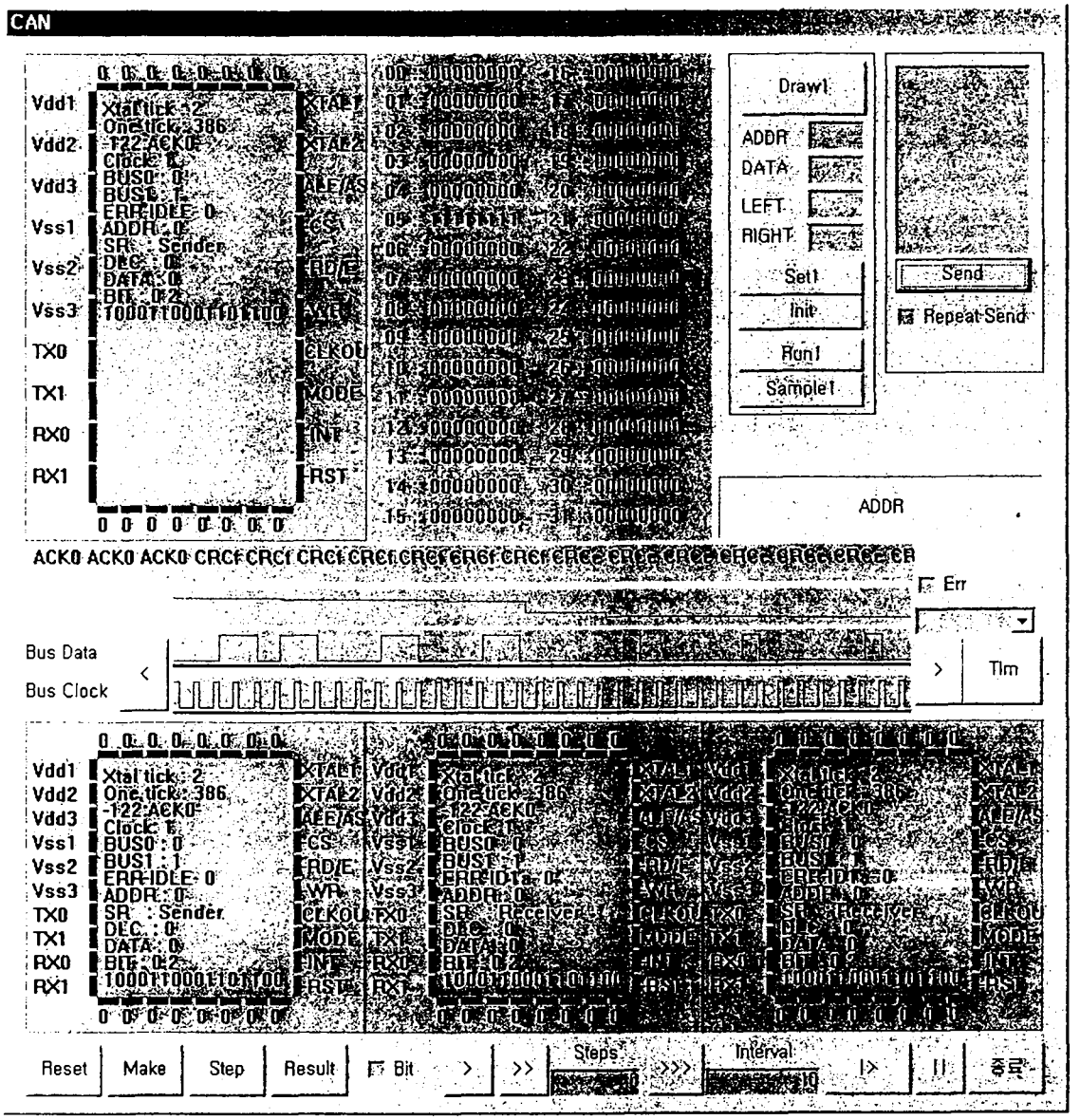
앞의 상태도에서는 송신 위주의 상태 변환을 살펴 보았고 이번에는 상태 변환 표에서 수신 노드 위주로 기술한다.

실제 변환에서 IDLE 상태에서는 입력값이 최초 Sampling 값이 'd' 인 경우 바로 수신노드가 되며 상태는 SOF 가 되어 수신데이터를 기다린다. 이후 버스 Rx 값에 해당하는 Sub2 상태에서 수신 값이 'd'인 경우에 ID10 상태가 된다. 각 ID10 ~ ID19 에 해당하는 ID1i 에서는 Rx Sampling 시간에 해당하는 시점의 버스 Rx를 얻고 ID1(i+1) 상태로 이동하며, 이때 Rx Bit 주소에 해당 비트 값을 적어둔다. 이후 ID1a 인 경우 다음 상태는 RTR 상태가 된다. 여기서 단순히 수신 노드만을 살펴보았는데, 만일 송신 노드일 경우라면, 송신한 비트 값과 수신한 비트 값이 'd'에 의해서 다른 경우 이 값에 따라 송신 노드는 수신 노드로 송수신 모드를 바꾸게 된다.

계속해서 수신 노드에서 RTR, IDE 와 경우에 따른 CAN 2.0 B 프레임에 따라서 각각의 비트에 따른 처리를 한 후 DLC0 의 상태가 되며, 이때 해당 프레임의 데이터 크기를 얻는다. 데이터 크기에 해당하는 4비트의 DLC를 얻을 후에 DLC3에서 데이터를 받은 이후 DATA0 상태가 되며, 이때부터 DATA(DLC*8) 만큼의 데이터를 얻는다. 이후 15비트의 CRCi 의 해당 과정을 거치고 ACK0, ACK1을 거친 후 7비트의 EOF와3비트의 IS 상태를 거친다. 이러한 과정에서 각각의 DATA, CRC, EOF, IS 능 비슷하게 counter의 값을 이용하게 된다. 특히 ACK1에서 보면 수신 노드에서 Sub0 시간에 'd'을 송신하고, Sub2 시간에 버스의 값을 읽어서 그 값에 따라 오류가 발생시 오류에 해당하는 신호를 처리한다.

② CAN 세부 상태에 의한 Simulation Code

본 연구에서 정의한 상태도의 로직 점검을 위하여 C 코드를 이용하여 CLOCK LEVEL Simulation을 하여 정의한 CAN의 상태의 정확성을 점검 하였다.



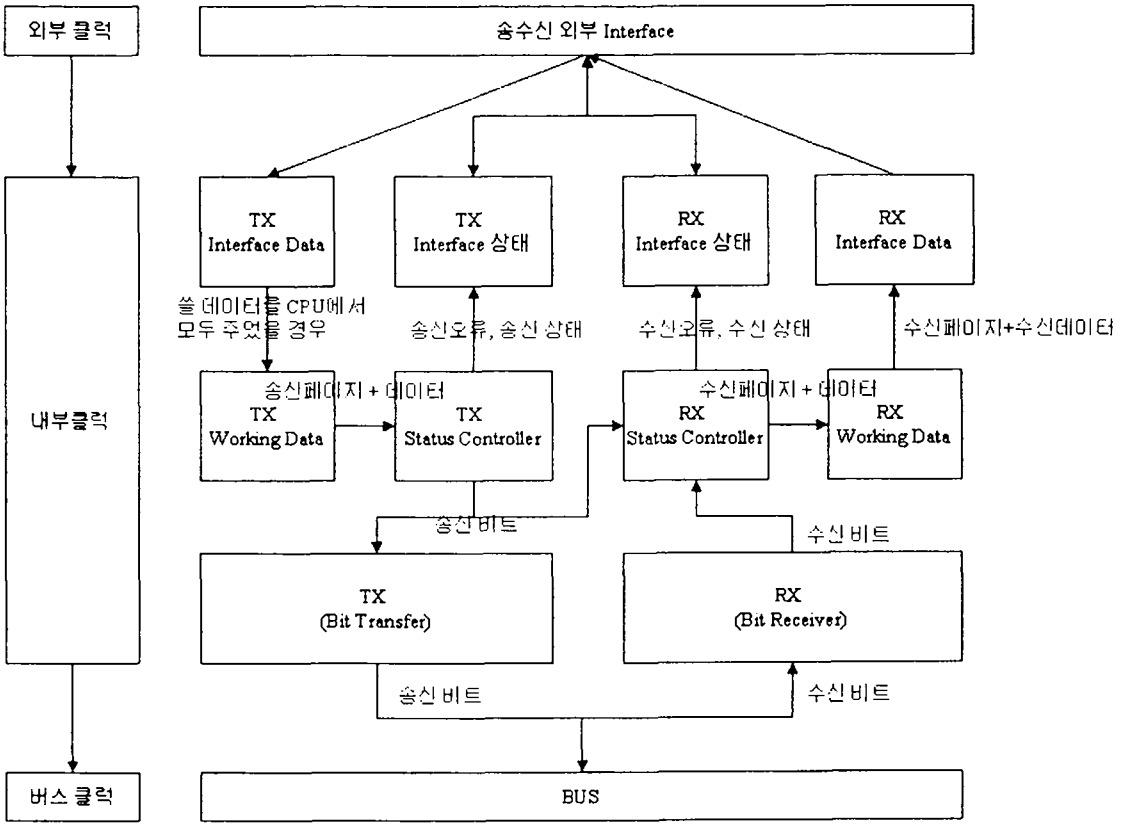
그림은 본 연구에서 CAN 로직 검증을 위한 테스트 프로그램을 표시한다. 테스트 프로그램은 각각 클럭 단위의 Simulation이 가능하도록 설계하였으며, CAN 로직을 점검한다. 그림 윗부분에 CAN의 칩 상태와 내부 변수의 값을 표시하고 있으며, 윗쪽 오른쪽에 칩에 대해서 설정이 가능한 명령 버튼을 두어서 CPU에서 오는 신호에 대비 하였다. CAN 칩은 전원부와 송수신부, 클럭 수신부, 칩에 대한 제어부와 입출력 주소부와 입출력 데이터부로 구성되어 있다. 전원부는 VDD와 VSS로 되어 있으며, VDD는 HIGH 값을 VSS는 GND를 나타낸다. 송수신부는 TX와 RX 부분으로 나누어져 있다. TX1은 클럭 수신부에서 수신한 클럭 신호와 상태에 따른 클럭을 이용하여 버스 클럭을 표시하고 있으며, TX0는 해당 시점에서 송신하는 버스의 데

이터를 표시하고 있다. TX1에서 송신한 버스 클럭을 RX1에 의해서 동기화 되며, TX0에서 생성한 데이터는 Rx Data Sampling 시점에서 Sampling 된다. 칩에 대한 제어부는 칩을 제어하는 외부에서의 신호를 표시한다. 이는 주로 CPU에서 생성된 칩 제어 신호와 주소 정보를 이용하여 설정하게 된다. 또한 여기서는 8비트로 구성된 CAN 내부 Register 주소 정보와 8비트로 구성된 데이터 버스를 이용하여 CAN 칩을 다룬다.

아래의 그림의 3개의 CAN 칩을 표시하고 있다. 기본적으로 CAN은 네트워크를 이용하기 때문에 A 노드에서 B노드로 노드간의 프레임이 제대로 움직이는가를 점검하기 위해서 3개의 칩을 Simulation했으며, 각각의 내부 상태가 그림의 위의 노드 표시부에 나타나게 하였다.

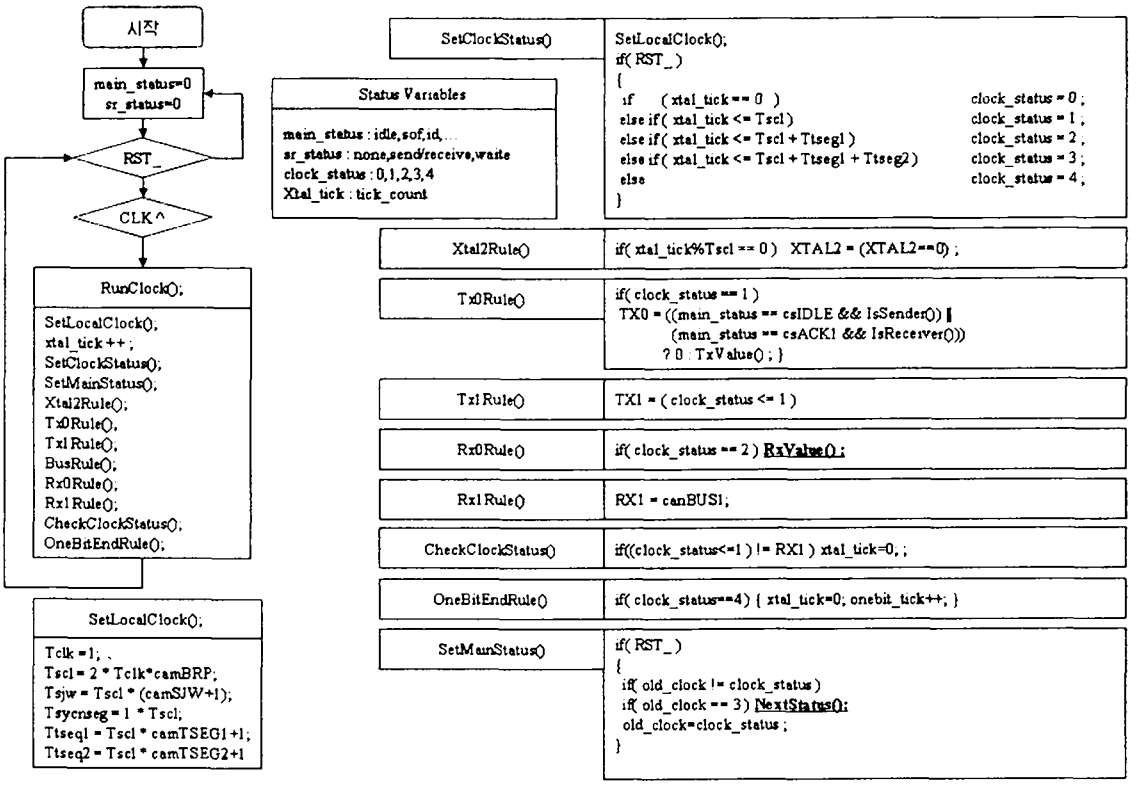
그리고 칩 내부의 상태를 표시하기 위하여 각 칩의 클럭 상태, 현재 상태, 버스 값, 오류가 나왔던 정소, 송수신 모드, 데이터 프레임 값, 데이터 값, CRC를 표시하도록 하여 CAN의 CORE 부분의 상태도가 제대로 이동하는 것을 알 수 있었다.

전체적인 버스의 값을 표시하기 위하여 중간에 버스 클럭과 버스 데이터와 현재 상태를 표시하였고, 이들을 통해 버스의 상태 진행 여부를 파악할 수 있었다.

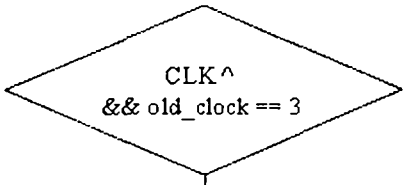


그림은 구현한 CAN의 내부 상태를 표시한다. 그림의 왼쪽 파트는 외부 클럭을 이용하여 내부 클럭과 버스 클럭을 생성하는 부분을 표시하고 있다. 이 내부 클럭 부분에서는 sampling에 필요한 신호를 생성하거나, 버스로 나갈 클럭을 생성한다. 오른쪽부분은 송수신 인터페이스에서 버스까지의 신호 흐름을 표시하고 있다. 외부에서는 송수신 상태나 송수신 오류 등의 내부 정보를 요청 할 수 있으며, 송신 데이터를 출력 할 수 있고, 수신 데이터를 읽어 갈 수 있다. 송수신 데이터의 흐름을 살펴보면, CPU에서 생성된 데이터들은 TX working data 영역에 쓰여지며, 이를 바탕으로 Tx 상태가 변경된다. 필요한 Tx 데이터가 있게 되면, Tx 상태 흐름도에 따라서 Tx 데이터 영역에 있는 데이터를 비트화 해서 CAN 버스로 출력하게 된다. 버스로 출력된 클럭 버스와 데이터 비트에 따라서 수신된 비트는 Rx 상태 처리기와 Rx 데이터 버퍼로 데이터가 이동되며, 하나의 데이터 프레임 수신이 완료되면 CPU는 Rx 상태를 읽고 해당 데이터 프레임을 읽는다.

<< Can Chip 상태 흐름도 >>



그림은 CAN을 위한 코드의 흐름도를 나타내고 있다. 최초 CAN이 초기화 될 경우 main_status와sr_status가 초기화 된다. 이것은 RESET 신호가 LOW인 경우 계속해서 초기화된 상태를 유지하게 된다. 만일 RESET 신호가 1인 경우에 외부 클럭에 의한 상태 변화가 일어난다. 클럭에 해당하는 상태 변화는 클럭 tick에 해당하는 xtal_tick을 하나 증가 시키며, 해당 clock 상태에 대한 Rule을 적용하여 Tx 비트를 전송 시점과, 버스의 Rx 신호를 수신한 시점을 결정하며, 버스 클럭에 대한 시점을 결정한다. 또한 한 비트가 끝나는 시점도 결정한다.



NextStatus();	
IsStatus(csIDLE)	NextLowHighStatus(csSOF ,csIDLE);
IsStatus(csSOF)	NextLowHighStatus(csID10,csERROR);
IsStatusRange(csID10,csID19)	NextArbitrationStatus();
IsStatus(csID1a)	NextArbitrationEndStatus();
IsStatusRange(csID20,csID2i)	NextArbitrationStatus();
IsStatus(csRTR0)	NextLowHighStatus(csIDE0,csIDE1);
IsStatus(csIDE0)	NextLowHighStatus(csRO ,csERROR);
IsStatus(csRTR1)	NextLowHighStatus(csIDE0,csIDE2);
IsStatus(csIDE1)	NextLowHighStatus(csRO ,csID20);
IsStatus(csIDE2)	NextLowHighStatus(csRO ,csERROR);
IsStatus(csRO)	NextLowHighStatus(csDLC0,csERROR);
IsStatusRange(csDLC0,csDLC2)	NextProgressStatus();
IsStatusRange(csDLC3,csDLC3+8*DLC()-1)	NextProgressStatus();
IsStatus(csDLC3+8*DLC())	NextLowHighStatus(csCRC0,csCRC0);
IsStatusRange(csCRC0,csCRCf)	NextProgressStatus();
IsStatus(csACK0)	NextProgressStatus();
IsStatus(csACK1)	NextProgressStatus();
IsStatusRange(csEOF0,csEOF5)	NextProgressStatus();
IsStatus(csEOF6)	if(IsSender()) is_sending_data=FALSE;
IsStatusRange(csIS0,csIS1)	NextProgressStatus();
IsStatus(csIS2)	NextLowHighStatus(csERROR,csIDLE);
IsStatus(csERROR)	NextStartStatus();
IsStatusRange(csWAIT0,csWAIT6)	NextLowHighStatus(csWAIT0,main_status+1);
IsStatus(csWAIT7)	NextLowHighStatus(csWAIT0,csIDLE);
CheckAndSetStatus(csIDLE);	

그림은 다음 상태를 변환하기 위한 블록을 나타낸다. 우선 기본적으로 버스의 데이터 비트 값을 얻은 후 다음 상태를 결정하게 되는데, 이에 해당하는 것을 표시한다. 즉 위에서 보면 IsStatus(csIDLE) NextLowHighStatus(csSOF,csIDLE)의 부분은 보면 만일 현재 상태가 IDLE 상태일 경우 현재 수신되는 신호가 LOW이면 SOF 상태로, 수신되는 신호가 HIGH 이면 IDLE상태를 계속 유지하라는 것을 나타낸다.

NextLowHighStatus(l,h)	{ if(RX0 == 0){ CheckAndSetStatus(ilow),} else { CheckAndSetStatus(ihigh), } return ,}
NextArbitrationStatus()	{ if(!(IsSender() && (TX0==0)==(RX0==0))){ sr_status=2; error_status=main_status;} main_status++ ; return ,}
NextArbitrationEndStatus()	{ if(!(IsSender() && (TX0==0)==(RX0==0))) { sr_status=2; error_status=main_status;} main_status = (IsReceiver() && !IsAcceptable()) ? csWAIT0 : main_status+1 ; return ; }

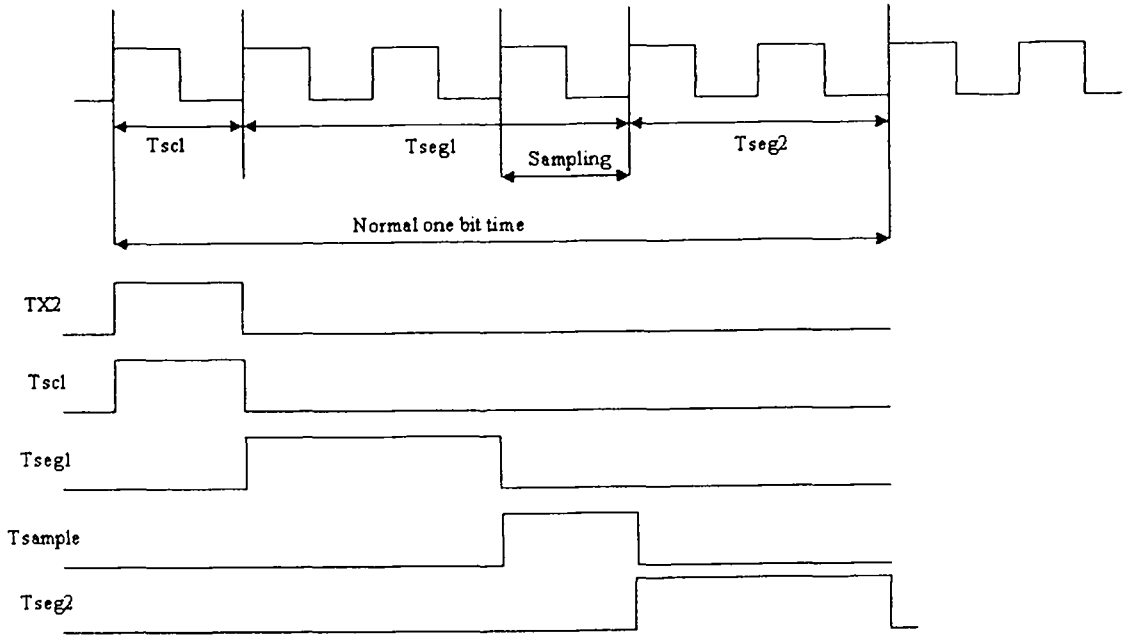
NextProgressStatus()	{ main_status++ ; return ,}
NextStartStatus()	{ main_status=0 ; sr_status=0 ; return;}
CheckAndSetStatus()	{ CheckAndErrorStatus(istatus); CheckAndStartStatus(istatus); }
CheckAndErrStatus()	{ if(istatus == csERROR) error_status=main_status; main_status=istatus; }

CheckAndStartStatus()	{ if(istatus == csIDLE)
	{ CrcInit();
	if(!is_sending_data && if_tx_q_has_data())
	{ TxQueueToRegister(); is_sending_data=TRUE; }
	sr_status=is_sending_data;
	}
	main_status=istatus;
	}

IsSender()	sr_status==1
IsReceiver()	sr_status==2
IsStatus(istatus)	if(main_status == istatus)
IsStatusRange(ist0,ist1)	if(ist0 <= main_status && main_status <= ist1)
IsAcceptable()	return (((canReceiveId & canAcceptCode) | canAcceptMask) == 0xFF);
DLC()	(IsSender() ? (CanAddrMap[11] & 0x0F) : (CanAddrMap[21] &0x0F));

그림은 다음 상태에 대한 마크로 정의와 상태 점검을 위한 상태를 표시하고 있다. 즉 예를 든 NextLowHighStatus(l,h)정의는 Rx0의 값에 따라 오류에 대한 상태 점검을 한 후에 상태를 설정한다.

이는 NextLowHighStatus(l,h) == if(RX == 0) {CheckAndSetStatus(l) … 이고, CheckAndSetStatus()는 오류를 점검를 점검하는 것을 체크하며, 현재 상태가 시작 상태인가를 점검하게 되는데, VHDL에서는 이를 동시에 표기가 가능하므로 CheckAndSetStatus()에 해당하는 것을 하면 된다.



그림은 기본적인 클럭 제어 부분에 해당하는 시간 제어 부분을 나타내고 있다. 이 부분은 SetClockStatus() 블록에 해당하는 것을 표시하고 있으며, 클럭을 조건에 따라 분기하여 각각 필요한 타이밍 신호를 생성하고 이를 사용한다.

③ Can VHDL 코드

CAN구현의 전체적인 흐름은 C를 통한 CAN core Simulation과 이를 VHDL로 변환하는 과정을 거쳐서 Can의 core 부분을 구현하였다. 특히 가장 복잡하며 상태 변화의 핵심은 Rx0 Rule이다. 이는 Sampleing 된 신호를 기본으로 다음의 상태로 변환하는 것을 포함한다. 즉clock 이 상승하고 Tsample='1' 인 경우 각각의 상태에 따라서 다음 상태를 결정하는 것이다.

각 상태에 따른 변화에서 일반적으로 계속 진행하는 경우

IsStatusRange(csERROR,csERROR7) NextProgressStatus();

위와 같이 오류 상태에 따른 다음 상태를 정의하는 문장은 다음과 같은 VHDL 코드로 변형된다.

```
    case main_status is
            when csERROR => next_status <= csERROR1 ;
            when csERROR1=> next_status <= csERROR2 ;
    ......
            when csERROR7=> next_status <= csWAIT  ;
```

```
            ......
        end case;
```

반면, 경우에 따라서 출력 값을 바꾸며 다음 상태로 변화하는 경우

IsStatus(csEOF6){ if(IsSender()) is_sending_data=FALSE;NextProgressStatus(); }

앞에서의 상태 변화 부분은 그대로 쓰고 신호값을 출력하는 부분을 생성한다.

```
case main_status is
        when csEOF6 => next_status <= csIS0 ;
        ....
end case;

is_sending_data = '0' when main_status = csEOF6 and next_status = csIS0 ; else
                  ......
```

또한 Arbitration 의 경우

IsStatusRange(csID10,csID19) NextArbitrationStatus();

메인 상태 변화는 일반적인 경우 NextStatus와 같다. 또한 조정 상태인지에 따라서 하지만, sr_status를 조건에 따라 변화 시켜야 한다.

```
case main_status is
        when csID10 => next_status <= csID11 ; -- 각각의 상태를 다음 상태로
        ....
end case;

--- 점검용 비트를 이용한다. --
is_arbitration_state <= '1' when main_status = csID10 and ··· = csIO11 ; else '0'
sr_status <= srsReceiver when is_arbitration_state = '1' and TX='0' and RX='1' ;
```

또한 가장 복잡한 부분이 RX 값이 LOW, HIGH 상태에 따라 바뀔 경우이며 이 경우 필요에 따라 해당 값이 오류가 발생하였는가에 대한 check도 필요하다.

IsStatus(csWAIT6) NextLowHighStatus(csWAIT0,csIDLE);

이 경우는 RX 가 LOW인 경우 다시 7개의 'r'을 기다리며, 그렇지 않을 경우 초기화하는 것을 나타낸다. 여기서 중요한 부분은 초기화 부분이며, 이 부분에서 초기화

시에 해당하는 값을 처리하는 부분이 있어야 한다. 특히 다음 상태가 오류 상태인 경우 오류 처리를 한다던가, 또는 다음 상태가 초기 상태일 경우 초기화가 필요하다.

```
-- RX에 따라서 상태를 변화시킨다 --
case main_status is
        when csWAIT6 => if ( RX = '0' ) then
next_status <= csWAIT0 ;
else
        next_status <= csIDLE ;
end if ;
.....
end case;
.....
-- 오류 상태나 초기 상태에 따른 초기화를 수행한다 --
case next_status is
when csERROR =>       if( main_status = csERROR ) then
                              err_cnt <= err_cnt + 1 ;
                      end if ;
       when csIDLE  => if( not(main_status = csIDLE) )  then
                          -- initialize variable --
                      end if ;
end case ;
```

각각의 Simulation 한 블록은 위와 같이 VHDL코드로 변경하여 CAN core 부분만을 구현하였다. 또한 RX에 의한 전체적인 코드 부분은 clk 이 변하고 Tsample이 변하는 시점에서 일반적인 다음 상태에 대한 정의를 하며, 이렇게 변한 부분은 한 Cycle이 끝나는 시점에서 다음 상태로 전이한다.

```
process( clk, Tsample, RX )
If( clk'event and clk='1' and Tsample='1' ) then
case main_status is
                when csIDLE => ..... -- 해당 상태 변경 --
```

```
        end case;
end if;
end process ;

process( main_status, next_status , clk, Tseg2 )
```

기본적인 클럭 제어 부분에 해당하는 VHDL 코드는 다음과 같이 해당 xtal_tick은 클럭이 발생할 때 마다 증가하고, 이에 따라서 clock_status를 생성한다. 그리고 이것에 따라 클럭 제어 상태에 해당하는 신호가 생성된다.

```
If  RST_L ='1' then
        If      ( xtal_tick = 0     )                   then clock_status <= 0 ;
        Elsif   (xtal_tick <= Tscl )                    then clock_status <= 1 ;
        Elsif   (xtal_tick <= TsclAndTtseg1 )           then clock_status <= 2 ;
        Elsif   (xtal_tick <= TsclAndSeg1AndSeg1) then clock_staus <= 3 ;
        Else                                                clock_staus <= 4 ;
        End if;
End if;

TX2   <= '1' when clock_status = 0 else '0' ;
Tseg1 <= '1' when clock_status = 1 else '0' ;
Tsample<='1' when clock_stauts = 2 else '0' ;
Tseg2  <='1' when clock_status = 3 else '0' ;
```

다 보드 검증을 위한 테스트 환경 및 개발

(1) 보드 검증 환경

보드의 전체적인 테스트 환경에 대해서 기술한다. 보드의 완성까지는 보드 회로 설계 과정, 보드 제작 및 점검 과정, FPGA 코딩 및 버그 수정 과정, 기본 프로그램 코딩 및 버그 수정 과정을 갖는다.

본 연구에서 사용한 툴들은 보드 제작과 FPGA 제작 603 코딩 및 버그 점검 등을 위해 기 제품을 이용하거나 개발하여 사용하였다.

보드 회로 설계 과정에서는 Veribest을 사용하였고, 보드 제작 및 점검은 Logic Analizer 와 Oscilloscope 을 사용하였으며, FPGA 구현 및 버그 수정 과정은 Actel 의 Libero 환경을 사용하였으며, 603 프로그램 코딩 및 점검 과정은 603 Diab Compiler를 사용하였다.

특히 본 프로젝트는 기존 상용의 프로그램 외에 테스트와 로직 점검을 위한 프로그램을 설계하고 제작하였다. CAN의 로직을 점검하기 위하여 National Instrument 사의 PCI CAN Interface를 이용하였으며, 전체적인 보드 상에서 CPU의 상태 점검을 위한 S/W를 구현하였다.

(가) FPGA 로직 구현 툴

보드 제작 과정에서 VHDL을 통해 FPGA를 작성하는 과정에서 본 프로젝트에서는 Actel의 리베로를 사용하였다. 그림은 리베로의 전반적인 그림을 표시하고 있다.

기본적으로 VHDL로 코딩된 FPGA 프로그램은 Synthesis 에 의해서 합성되며, 합성된 파일을 기본으로 FPGA 라우팅을 결정하여 파일로 생성하며, 이를 기반으로 FPGA 로직을 구현 한다.

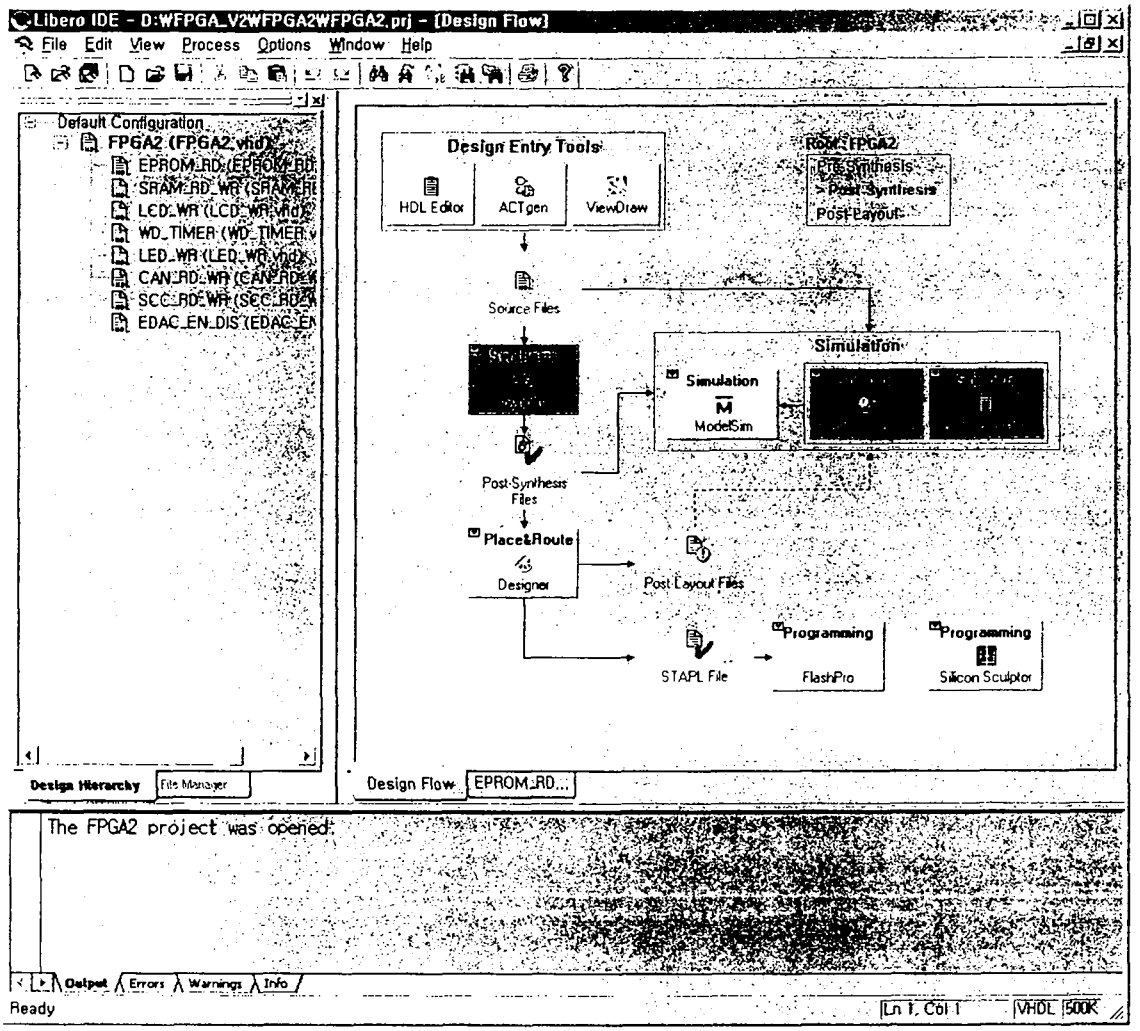


그림 103 리베로 작업 환경

그림은 Actel의 Flash Pro를 이용하여 FPGA를 로직 구현 과정을 표시한다.

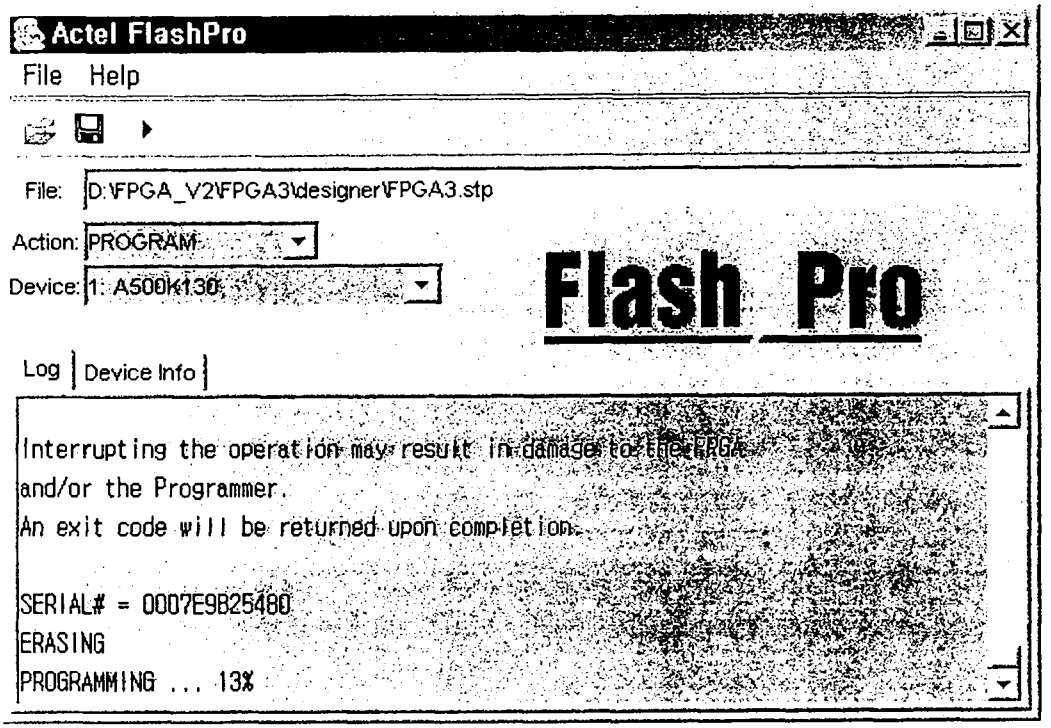


그림 104 FlashPro 작업 환경

(나) ROM Emulation 툴

본 과제에서 개발한 보드에서 CPU를 테스트하기 위한 코딩 과정은 다음과 같다. 우선 테스트를 위한 603CPU의 코드는 C 로 작성한다. C 코드를 603 컴파일러로 컴파일 한다. 컴파일된 코드를 ROM에 굽는다. 하지만, ROM에 굽는 작업 시간이 많이 소요되는 작업이고, 또한 ROM을 탑재하거나, 제거 하는데 작업이 필요하므로, 본 프로젝트에서는 점검 코드의 효율적인 사용을 위하여 ROM Emulator를 이용하여 테스트를 진행하였다.

그림은 ROM Emulator를 통하여 RomEmulator에 RomCode를 설정한 후의 그림을 나타낸다.

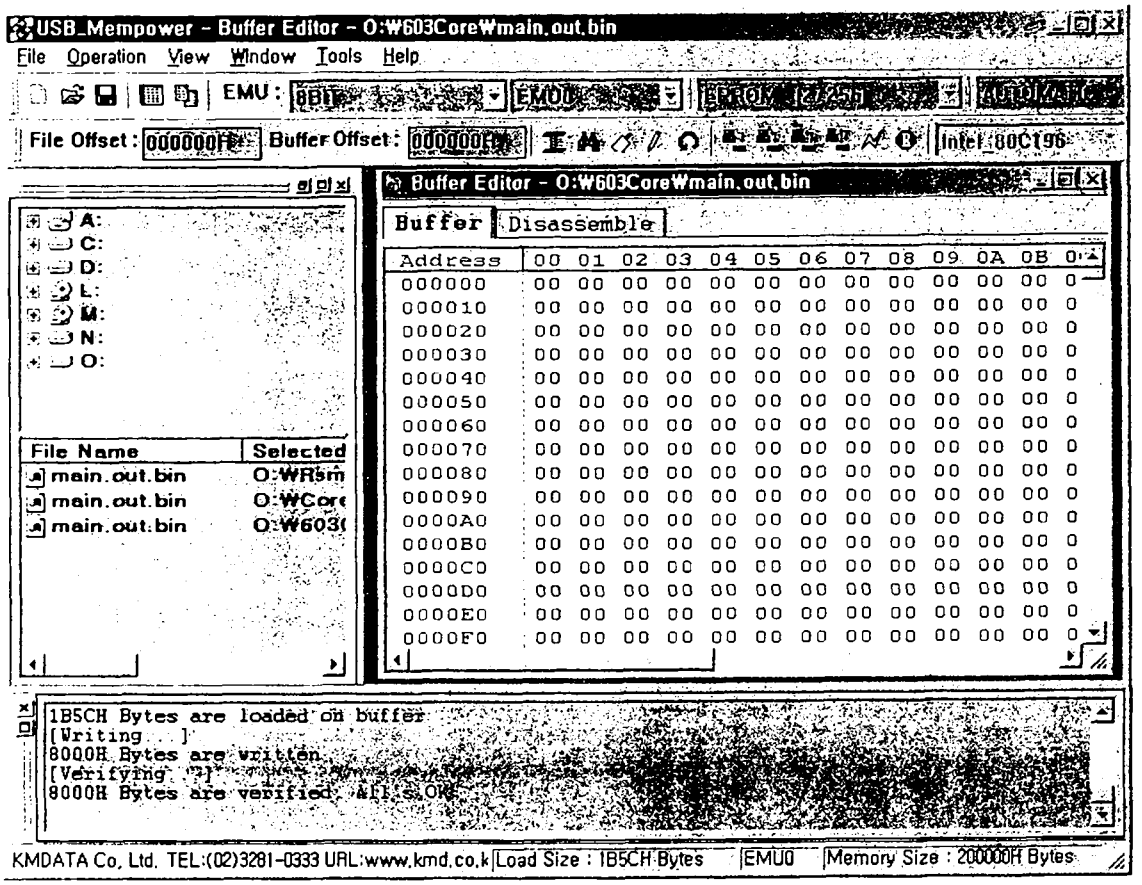


그림 105 롬 에뮬레이터를 통한 ROM Emulation 환경

(2) 보드 검증 환경 개발

(가) LED를 통한 H/W 검증

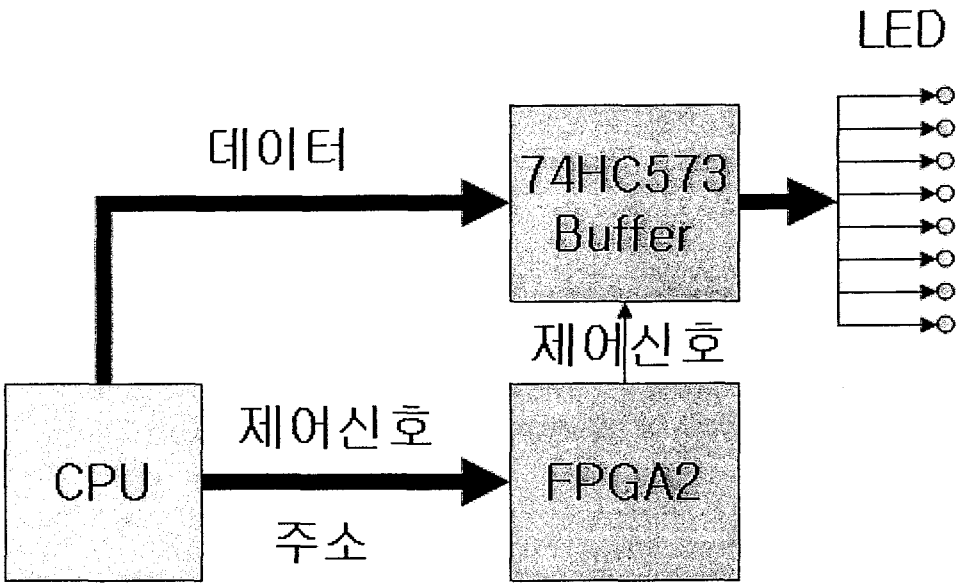


그림 106 LED 제어도

본 연구에서 개발한 보드에는 주 보드에는 IO를 위한 FPGA인 FPGA3에 의해 제어되는 1Byte에 해당하는 LED를 가지고 있으며, 이것은 DIP 스위치에 의해서 FPGA3의 자체 상태를 비롯하여 여러 상태를 가지고 있다. 또한 테스트를 위한 보조 보드에는 2 바이트에 해당하는 LED를 가지고 있다. 이는 직접CPU 버스에 연결되어 있으며, FPGA2 제어 신호에 의해서 데이터를 버퍼링 하며, 각각 LED1, LED2라 하겠다. 이를 위해 2개의 74HC573P 칩을 두개를 사용하여 LED를 버퍼링을 한다. 즉 만일 LED1에 출력을 원한다면, CPU 에서는 LED1에 해당하는 주소와 필요한 데이터를 출력하고, 이는 FPGA2의 제어 신호에 의해서 CPU 데이터 버스의 한 바이트 데이터가 LED1 출력 신호를 관장하는 74HC573에 데이터를 쓴다. 그리고 74HC573의 출력 신호가 LED1의 출력으로 표시된다. 이는 마치 CPU의 데이터를 다른 레지스터에 쓰는 동작과 같다. 이는 FPGA2의 제어 신호를 생성하는 과정에서 해당 주소에 쓰는 동작을 하는 것과 같다.

```
process(CLK, RST_L, LED_SEL)
   begin
      if(RST_L = '0') then
         LED_DONE_L <= '1';
      elsif (CLK'EVENT and CLK = '1') then
         if (LED_SEL = '1') then
            LED_DONE_L <= '0';
         else
            LED_DONE_L <= '1';
         end if;
```

```
    end if;
  end process;
```

(나) LCD를 통한 H/W 검증

LCD 모듈은 주로 마이크로 프로세서와 결합하여 문자와 화상을 표시하는데 사용되거나, 간단한 계측 장비에 상용된다. LCD제어 장치는 크게 디스플레이 부와 컨트롤부의 두 부분으로 나뉘는데, 본 프로젝트에서는 디버깅을 위하여 제어부와 디스플레이 부가 하나로 되어 있는 16*2 LCD 모듈을 사용하였다. LCD는 고유의 특성 때문에 고속으로 동작 시키면 정상적인 Data를 읽거나 쓸 수 없으므로, LCD 사양서에 나와 있는 타이밍 조건을 만족시켜야 한다. 본 연구에서는 LCD 는 디버깅 용으로 사용하므로 쓰기 동작만을 이용한다.

LCD의 타이밍 조건은 다음과 같다.

항목(시간)	기호	최소시간	최대시간	단위
EnableCycle	tCYC	500	-	(ns)
EnablePulse	PW	220	-	(ns)
AddressSetup	tAS	140	-	(ns)
DataDelay	tDDR	-	320	(ns)
DataSetup	tDSW	195	-	(ns)
DataHold	tH	10	-	(ns)
AddressHold	tAH	20	-	(ns)

CPU 버스를 이용하여 LCD에 데이터를 표시하기 위해 본 연구에서는 CPU에서 LCD에 데이터나 제어 데이터에 대해 출력 신호를 생성하면, 제어 FPGA인 FPGA2에서 LCD 타이밍 조건을 맞추기 위한 동작을 하게 된다. 각각의 시간은 25Mhz에 해당하는 1클럭의 기간은 40ns 이므로 각각 필요한 클럭 만큼에 해당하는 Delay를 한 후 LCD 출력 명령을 완료한 상태를 출력하도록 하였다.

다음 FPGA 코드는 FPGA2에서 LCD 제어를 위해 사용한 코드이다. 우선 칩 선택 신호를 5 클럭 시간동안 출력하며, 방향은 항상 LCD로의 출력을 가리키며, 단순 데이터인가 제어 데이터인가는 선택 신호에 따라 바뀌며, 8 클럭 Delay를 한 후 LCD로의 데이터 출력을 종료한다.

```
-- state transition
    process( CLK, RST_L, LCD_SEL )
    begin
        if( RST_L = '0' ) then
                lcd_state <= sReady ;
        elsif( CLK'EVENT and CLK = '1' ) then
            if ( lcd_state = sReady and ( LCD_SEL(0) = '1' or LCD_SEL(1) = '1') )
then lcd_state <= sData0  ;
            elsif( lcd_state = sData0 )                then lcd_state <= sData1  ;
            elsif( lcd_state = sData1 )                then lcd_state <= sData2  ;
            elsif( lcd_state = sData2 )                then lcd_state <= sData3  ;
            elsif( lcd_state = sData3 )                then lcd_state <= sData4  ;
            elsif( lcd_state = sData4 )                then lcd_state <= sData5  ;
            elsif( lcd_state = sData5 )                then lcd_state <= sData6  ;
            elsif( lcd_state = sData6 )                then lcd_state <= sData7  ;
            elsif( lcd_state = sData7 )                then lcd_state <= sEnd    ;
            elsif( lcd_state = sEnd   )                then lcd_state <= sReady  ;
            else                                       lcd_state <= sReady  ;
            end if;
        end if;
    end process;

-- 상태에 따른 제어 LCD 제어신호 생성 --
LCD_CS      <= '1' when lcd_state = sData0 else
               '1' when lcd_state = sData1 else
               '1' when lcd_state = sData2 else
               '1' when lcd_state = sData3 else
               '1' when lcd_state = sData4 else
               '0' ;
    -- RS --
    LCD_RD_WR  <= '0' when LCD_SEL(1) = '1' else -- CTRL
                  '1' ;                          -- DATA
```

```
-- R/W --
LCD_DIR     <= '0' ;                              -- WRITE
LCD_DONE_L  <= '0' when lcd_state = sEnd else '1' ;
```

LCD로 적절한 상태의 데이터를 출력하기 위해서는 적절한 제어 신호와 절적한 데이터 신호를 출력할 필요가 있다. 본 연구에서 사용한 명령은 LcdOutCtrl()함수 작성을 통해 LCD 초기화, CLS, HOME, GOTO 00 등의 제어 신호를 주고, LcdOutStr(str)함수 작성을 통해 문자열을 출력할 수 있도록 하였다.
예를 들어 첫 번째 줄에 "START" 라는 문자열을 출력을 한다면, LcdOutCtrl(0x80) 명령을 통하여 첫 번째 줄로 이동하고, LcdOutStr("START") 를 통하여 "START"라는 문자열을 출력한다.

(다) DIP 스위치를 이용한 FPGA 개발 로직 검증

```
SRAM_WR_L(0) <= i_sram_wr_l or FPGA2_DIP(0);  --
SRAM_WR_L(1) <= i_sram_wr_l or FPGA2_DIP(1);  --
SRAM_WR_L(2) <= i_sram_wr_l or FPGA2_DIP(2);  --
WD_RST       <= i_wd_rst    when FPGA2_DIP(3) = '1' else
          '0'    when FPGA2_DIP(4) = '1' and CPU_DATA_DIFF = '1' else
          '1'  ;
```

본 프로젝트는 DIP스위치를 이용하여 간단한 보드의 동작 여부를 결정한다. 기본적으로 FPGA2는 제어 신호를 생성하는 FPGA이며, 이에 대해 DIP switch를 이용하여 5가지 제어가 가능하게 하였다. 먼저 DIP0에서 DIP2 번은 SRAM TMR 제어를 위하여 사용하였다. DIP switch 가 1 인 경우 해당하는 SRAM에 대한 Write 신호가 생성하지 않게 하여 각각의 메모리 영역의 값을 다르게 할 수 있다. 또한 이것은 JUMPER Setting에 의한 EDAC logic 동작에 따라 오류를 극복하는 것을 알 수 있다.
DIP3은 WatchDog제어에 대한 동작 여부를 결정한다. 이것이 설정되어 있으면, WatchDog이 동작한다. 그러므로 CPU 동작의 오류나 프로그램의 오류가 발생하였을 경우 WatchDog에 의한 CPU Reset에 대한 시험시 사용하게 된다.

DIP4는 메인 CPU와 보조 CPU 간의 서로 다른 값을 가질 경우 이에 대해 CPU의 동작을 멈추게 한다. 이를 통하여 시험시 서로 CPU의 출력 값이 서로 다른지에 대한 간단한 판단이 가능하다.

① DIP Switch를 이용한 FPGA2 상태 표시

```
LED_SMD_A(0) <= '1'                  when FPGA2_DIP(5 to 7 ) = "000" else
                PC_TO_FPGA(1) when FPGA2_DIP(5 to 7 ) = "001" else
                PC_TO_FPGA(2) when FPGA2_DIP(5 to 7 ) = "010" else
                '0'    ;

LED_SMD_A(1) <= i_eprom_sel   when FPGA2_DIP(5 to 7) = "000" else
                i_sram_sel    when FPGA2_DIP(5 to 7) = "001" else
                i_led_sel(0)  when FPGA2_DIP(5 to 7) = "010" else
                i_lcd_sel(1)  when FPGA2_DIP(5 to 7) = "011" else
                i_wd_sel      when FPGA2_DIP(5 to 7) = "100" else
                i_scc_sel     when FPGA2_DIP(5 to 7) = "101" else
                i_can_sel     when FPGA2_DIP(5 to 7) = "110" else
                i_reg_sel     when FPGA2_DIP(5 to 7) = "111" else
                                        '0' ;

LED_SMD_A(2) <= i_eprom1_cs_l when FPGA2_DIP(5 to 7) = "000" else
                i_eprom1_rd_l when FPGA2_DIP(5 to 7) = "001" else
                i_sram_cs_l   when FPGA2_DIP(5 to 7) = "010" else
                i_sram_rd_l   when FPGA2_DIP(5 to 7) = "011" else
                i_wd_rst      when FPGA2_DIP(5 to 7) = "100" else
                i_wd_sel      when FPGA2_DIP(5 to 7) = "101" else
                i_wd_done_l   when FPGA2_DIP(5 to 7) = "110" else
                '0' ;
```

제어용 FPGA2에서 일반적인 상태를 표시하기 위한 LED는 3개다. 이를 통하여 표시하기 위해 DIP5~7 의 3개를 이용하여 8개의 가지수를 만들어서 각 LED에 표시하게 하였다. LED 0 는 PC에서 들어오는 신호를 확인하거나, FPGA의 전원 상태를 확인하는데 사용하였다.
LED1은 각 부분의 선택 제어 신호가 생성되는지에 대해 간단히 확인하기 위해 사용하였으며, LED2는 각 부분의 읽기 쓰기 등의 제어 신호를 확인 하기 위해 사용하였다.

② DIP Switch를 이용한 FPGA3 상태 표시

```
LED_SMD_A <=  i_shift_reg              when FPGA3_DIP(0) = '1' else
              i_scc1_tx_reg            when FPGA3_DIP(1) = '1' else
              i_scc3_tx_reg            when FPGA3_DIP(2) = '1' else
              i_scc1_rx_reg            when FPGA3_DIP(3) = '1' else
```

```
i_scc3_rx_reg                when FPGA3_DIP(4) = '1' else
i_scc_clk_cnt(15 downto 8 )  when FPGA3_DIP(5) = '1' else
i_can1_tx_reg                when FPGA3_DIP(6) = '1' else
i_can1_rx_reg                when FPGA3_DIP(7) = '1' else
i_cnt_reg( 31 downto 24 ) ;
```

IO 제어용 FPGA3에는 상태 표시를 위해 8개의LED를 사용할 수 있다. 이를 통하여 FPGA3의 각각의 상태를 표시할 수 있도록 DIP스위치를 이용하여 표시하였다. 8가지 상태를 사용하였다. 각각은 FPGA3의 기초 동작 여부 파악을 위한 상태 표시, IO register에 대한 기초 동작 여부 파악을 위한 상태 표시, SCC 클럭 표시 상태, 내부 계수기 상태 표시를 하고 있으며, 이를 통하여 FPGA3의 전반적인 동작여부를 확인할 수 있다.

또한 FPGA 코드 개발 과정에서 필요한 사항은 적절히 DIP Switch에 대한 제어를 조절하므로써 가능하다. 하지만, LED는 기초적인 FPGA3의 상태를 파악하고, 전체적인 동작은에 대한 상태 점검은 603 프로그램을 통하여 LCD에 문자를 표시하거나, 통신 채널을 통한 상태 전송을 통하여 확인한다.

(라) 두 개의 CPU 에 대한 동기화 결과

본 프로젝트에서는CPU를 두개를 사용하였다. 하나는 메인 CPU 이고 다른 하나는 메인 CPU를 감시하는 보조 CPU이다. 본 프로젝트에서는 이 두 프로세서를 통해서 CPU가 같이 동작하여, SEU등의 열악한 환경에서 CPU 오동작을 감시하여, CPU를 RESET하여 재시동 함으로써 위성 등에서 사용시 빠른 복구가 이루어지도록 하였다.

본 프로젝트에서는 이러한 보조 프로세서에 대해서 활용 가능한지에 대한 의문이 있었으며, 프로젝트의 실험 결과 어느 정도까지 실험 가능하다는 것을 알 수 있었다.

이것에 대한 검증 과정은 FPGA에서 두개의 CPU에서 나오는 데이터 신호를 가지고 검증을 하였다. 특히 문제가 되는 시점은 데이터를 쓰는 시점에서 올바른 데이터를 써야 하므로, 이 시점에서 두 개의 CPU 신호가 같은가? 에 대한 검증을 하였다.

신호가 같은가에 대한 검증을 위해서 보조 CPU에서 FPGA로 들어가는 신호에 대해서 잘못된 정보를 추가 할 수 있게 하였다. 또한 CPU에서는 COUNT를 세어 LCD에 표시하는 프로그램을 수행을 하고, CPU가 서로 다른 신호를 내 보낼 경우, FPGA에서 CPU의 프로그램을 멈추게 하여 LCD에 COUT 정보가 표시되도록 하였다.

실험 결과 제작된 보드 상에서 일반적으로 두 CPU의 상태는 같았다. 하지만, 잘못된 정보를 추가적으로 주었을 경우 CPU가 멈추는 현상이 나타났으므로, FPGA에서의 CPU의 서로 다른 값에 대해 동작이 제대로 됨을 알 수 있었다.

하지만, 여러 가지 실험을 해 본 결과 CPU가 다른 데이터를 내 놓는 현상을 얻어낼 수 있었다. 실험은 FPGA에 CPU 데이터 입력시 Jumper를 통하여 오류 정보나 CPU 데이터 정보를 선택할 수 있게 하였는데, 이때 CPU에서 FPGA로 들어가는 점퍼의 길이가 3mm 정도의 짧은 점퍼를 대신하여 점퍼 사이의 전선의 길이를 늘려 사용해 보았다. 그 결과 주변 환경에 따라 다른 현상을 보이지만, 약 60센티미터 정도의 선에서 환경에 따라 두 CPU가 서로 다른 값을 출력하는 경우가 발생하였다.

이는 이러한 용도의 보드 개발시 배선에 따라서 두 CPU가 서로 동기화 되지 않은 데이터를 출력할 수 있음을 보이고 있다.

즉, 본 프로젝트에서 하는 방법을 우주환경과 같은 열악한 환경에서 사용하려면, 보드 설계에서부터 치밀한 계획과 많은 테스트를 하여야 함을 나타낸다.

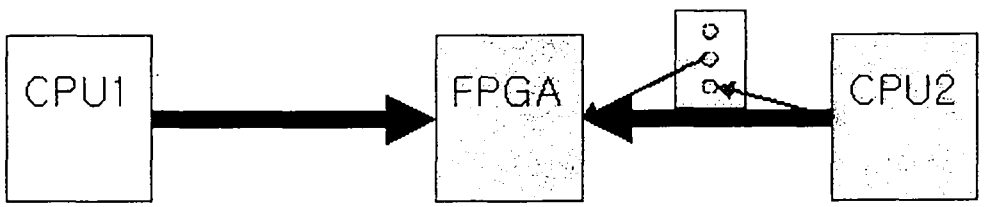
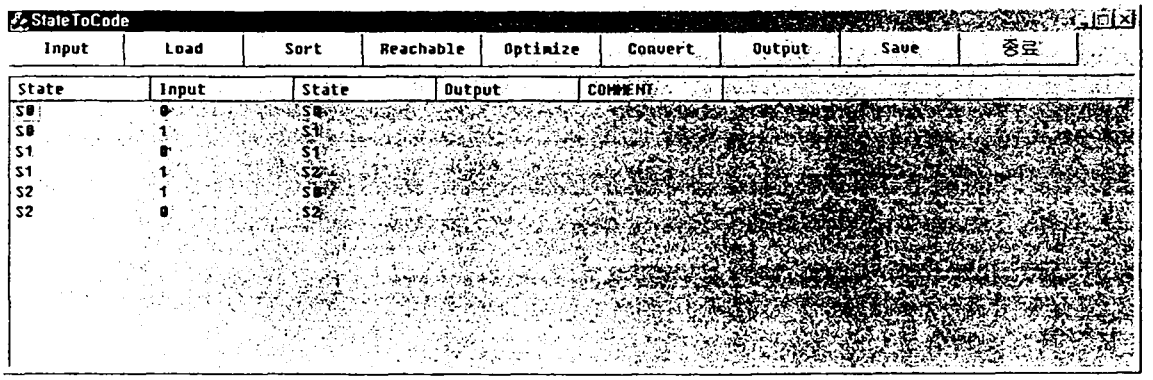


그림 107 두 개의 CPU 동작 검증을 위한 설정

(마) StateToC S/W 코드 개발을 통한 로직 개발

본 연구는 많은 VHDL 코딩 작업을 필요로 한다. 그리고 대부분의 코딩 작업에는 상태를 정의하고 상태에 따른 출력을 정의하며, 입력에 따른 상태 변화를 필요로 한다. 그러므로 본 연구에서는 이러한 상태 변화도에 따른 VHDL 코딩의 효율적인 작업을 위하여 간단히 상태를 VHDL 코드로 변환하는 툴을 개발하였다.

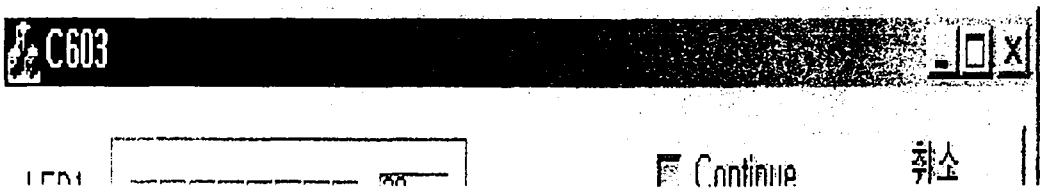
개발된 툴은 현재 상태에서 입력에 따른 상태 변화와 출력을 콤마로 분리한 상태 변환 정의 파일을 입력으로 한다. 그림은 입력 파일을 로드한 상태를 표시하고 있다.



로드한 정보를 VHDL 코드로 변환과정은 로드한 정보를 현재 상태에 따라 정렬하고 불필요한 상태를 제거하기 위해서 초기 상태에서 도달 가능한 상태만을 고른다. 이후 현재 상태에서 입력 범위에 따라 상태 변화 규칙을 줄이고, 이 규칙을 기반으로 VHDL 코드로 변환한다. 그림은 상태를 VHDL로 변환후의 결과를 표시한다. 또한 필요에 따라 VHDL 코드로 저장한다. 이렇게 생성된 VHDL 코드는 필요에 따라 수정하여 검증된 FPGA코드를 생성한다.

(바) C 코드 테스트 환경 개발

그림은 603 코드 테스트를 위해서 Window를 이용하여 생성한 테스트 S/W 이다. 이것을 통하여 C 코드를 우선 Window 환경에서 Simulation한다. 그리고 이 코드를 603 코드에 적용하여 초기 603 코드의 수행에 관한 검증된 코드를 이용하여 제작된 보드에서 C 코드에서 컴파일된 바이너리 파일을 이용하여 롬을 생성하여 초기 보드 테스트 수행에 이용하였다.



이러한 환경을 제공하기 위해서 Dialog 화면에 해당하는 class를 생성하였으며, 다음과 같은 내부 변수와 인터페이스 함수를 위한 메소드 함수들을 정의 하였다. 클래스 한 부분으로서 LED 에 해당하는 변수 값을 가진 클래스 부분만을 나타내고 있다. 변수로서는 Led1, Led2를 가지고 있으며, 메소드로서는 dSetLed1 … 등등의 설정 등의 메소드를 표시하고 있다.

```
class CC603Dlg : public CDialog
{
// Construction
public:
        CC603Dlg(CWnd* pParent = NULL);       // standard constructor
public: // 603 interface value //
        BYTE Led1,Led2        ;
public:  // 603 interface //
        void dSetLed1(BYTE iv) { Led1 = iv ; dSetLed1UI(Led1); }
}
```

생성한 클래스를 이용하기 위해서 C 의 프리프로세스를 이용하여 적용한다. 예를 들어 윈도우 환경에서 프로그램이 돌때는 인터페이스 정의SetLed1(macV)는 다이얼로그의 LED를 설정하는 것으로 하고, 그렇지 않을 경우 SetLed1(macV)는 직접 LED1 주소에 값을 쓰는 작업을 하게 정의한다.

```
#ifdef WIN_ENV

extern  CC603Dlg* pDlg ;
#define SetLed1(macV)           pDlg->dSetLed1(macV)
#else
#define LED1_ADDR               (0xB0000000)
#define SetLed1(macV)           { *(uint*)LED1_ADDR = (macV) ;}
#endif
```

또한 603 코어 프로그램에 대해서는 Core603이라는 함수를 선언하여 프로그램을 수행한다.

즉 윈도우 환경에서는 DWORD Core603(void)으로 선언하고, 수행 버튼에 의해서 Core603을 쓰레드를 이용하여 수행한다. 반면 실제 603 환경에서는 void Core603(void)을 선언하고, main 함수에서 Core603()을 호출하여 수행한다. 추가적으로 윈도우 환경에서는 필요에 따라 쓰레드를 종료할 필요가 있으므로 주 메인 루

프 부분에 약간의 수정을 가하여 쓰레드의 수행에 관한 제어가 가능하게 하였다

```
// main function 정의
#ifdef WIN_ENV
DWORD Core603(void)
#else
void Core603(void)
#endif

// 주 메인 루프 정의 //
#ifdef WIN_ENV
        while( pDlg && pDlg->mb_ContinueRun )
#else
        while(1)          // when
#endif
```

(사) 603 SW 를 통한 보드 검증

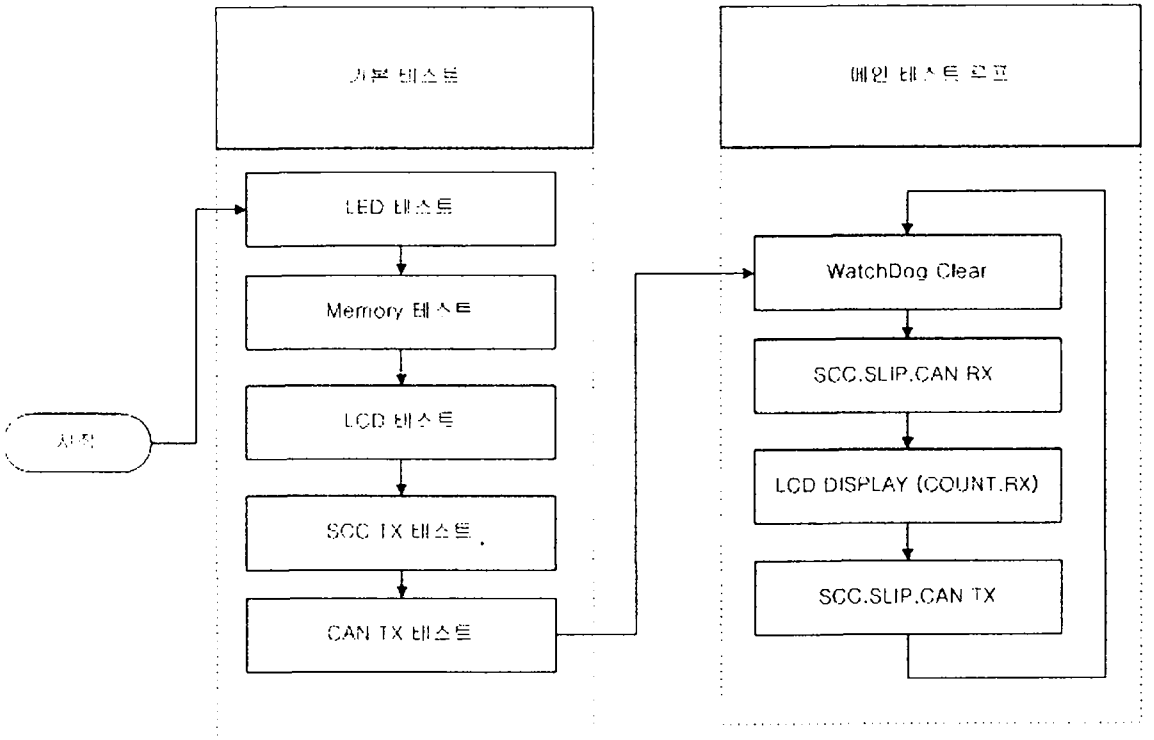


그림 110 603 CPU 프로그램 흐름도

본 연구에서 개발에 사용한 603용 S/W는 개발한 각 모듈을 테스트하는 목적으로 개발하였으며, 기본적으로 간단히 테스트하는 모듈과 통신을 통하여 좀 더 복잡한 일에 대한 테스트를 하는 메인 테스트 루프로 이루어 졌다. 일단 이 프로그램이 수행이 제대로 이루어지려면, CPU는 이미 ROM에서 데이터를 읽어서 수행할 수 있는 환경이 구축된 것이다. 이를 기반으로 개발한 FPGA 모듈을 테스트 할 수 있다. 기본 테스트 모듈에서는 간단한 레지스터 설정 모듈을 체크 할 수 있는 LED 테스트 모듈이 있다. 이 LED 테스트 모듈에서는 적절한 값이 LED에 표시하는 것으로 테스트 하였으며, LED에 출력되는 값을 증가시켜 가면서 LED의 불이 제대로 들어오는지를 보는 것으로 확인 하였다. 일단 LED테스트가 끝나면 필요한 데이터의 상태를 표시 할 수 있다. 다음은 SRAM 메모리 테스트 모듈이다. 기본적으로 메모리 테스트는 읽고 쓰는 것에 달렸으므로, 특정 메모리 영역에 대해서 RULE에 맞는 값을 메모리 영역에 쓰고, 일정 기간 이후 특정 메모리 영역의 값을 읽어서 쓴 데이터와 읽은 데이터 값이 같은지를 체크 한다. LCD 테스트는 LCD에 값을 출력하여 테스트를 한다. 필요한 데이터를 출력하고 그 값이 나오는 가에 대해 알아본다. 이 테스트에서는 적절한 제어 코드를 보내고 데이터를 출력한다. 제어는 LCD를 초기

화하며, 필요한 위치로 커서의 위치를 옮기는 작업을 한다. 테스트용 출력 데이터는 1234678로 이 값이 제대로 LCD에 출력 되는가를 확인한다.

다음은 IO에 대한 기본적인 테스트이다. 일단 IO를 통해서 출력하려면 FPGA3를 이용한다. 이 FPGA3에 출력되는 레지스터 값을 저장하고 DIP S/W를 적절히 조절하면 LED를 통하여 그 값을 알 수 있는 구조이기 때문에 DIP S/W 조절을 통해 SCC TX의 레지스터 값에 적절한 출력 값을 보내고, 이 값을 LED로 확인 할 수 있게 한다. CAN TX 도 SCC TX와 마찬가지로 적당한 출력 값을 출력하고 그에 해당하는 값을 LED로 확인하므로써 기본 적인 IO 에 해당하는 TX 값을 확인 할 수 있다. 초기 FPGA 개발 버전에는 SCC TX시 FPGA3 속에 레지스터 값을 설정하고, SCC RX시 FPGA3의 레지스터 값을 읽게 하여 TX와RX를 한 후 서로 값이 같은 가를 확인하여 IO 모듈에 대한 레지스터 값을 읽거나 쓰는 것에 대한 확인을 하였다.

메인 테스트 루프는 단순히 값을 쓰고 읽는 것을 떠나 주기적인 처리를 하거나, 좀 더 복잡한 작업에 대한 테스트를 하거나 통신을 통해 입력과 출력을 제어하거나 하는 등의 테스트를 하는 모듈이다.

먼저 WatchDog Clear 테스트는 적당한 시간 안에 WatchDog Clear 를 한다. 그렇지 않을 경우 WatchDog에 의해서 프로그램은 재 수행 된다. 이 WatchDog에 대한 테스트는 ROM Emulatio를 통한 실험시 검증된다. 새로운 롬 프로그램을 롬 에뮬레이터에 올리게 되면 이때 CPU는 특정 명령을 수행하다가 원하는 동작을 하지 못하게 된다. 이 경우 WatchDog이 걸리게 되어 다시 CPU가 새로운 롬 코드를 수행하는 것을 알 수 있다. 하지만 프로그램이 제대로 도는 경우에는 계속하여 WatchDog Clear신호를 생성하기 때문에 WatchDog이 걸리지 않음으로써 이를 테스트 할 수 있다.

LCD 화면 출력 테스트는 메인 테스트 루프를 돌면서 카운트 값을 증가하는데, 이 카운트 값을 계속 출력한다. 이를 이용하여 CPU의 동작 여부를 파악할 수 있다. 또한 LCD의 두 번째 줄에는 RX에서 온 데이터 값을 표시하게 하였다. 이를 통하여 현재 RX 의 동작 여부를 파악할 수 있다.

RX 테스트 루프는 SCC나 SLIP,CAN에 의해서 수신된 값을 가지고 처리한다. 기본적으로 LCD에 표시가 되지만, 간단한 수신 명령에 의해서 LED의 값을 변경한다던가 하는 기초적인 작업을 수행하므로 써 RX의 동작 여부를 파악 할 수 있다.

TX 테스트 루프는 각 통신 채널에 필요한 데이터를 출력하여 출력과 연결된 PC에서 출력 값을 제대로 받는지를 확인하여 TX의 출력을 확인하여 점검 할 수 있다.
기본적으로 메인 테스트 루프는 무한 루프를 돌면서 입력에 대한 반응을 보임으로써 보드상에서 프로그램이 제대로 돌고 있는지에 대한 검증을 할 수 있다. 특히 LCD에 계속해서 루프 카운트 값을 출력하여 기간을 두고 CPU의 동작 상태를 점검 할 수 있다.

다음은 기본 테스트에 해당하는 코드를 기술하고 있으며, 초기에 SCC나 LCD를 통해 출력 값을 출력하고 있다 이는 프로그램을 작성하면서 초기 테스트 모듈이라기보다 어느 정도 완성된 이후 어떠한 테스트를 하고 있는지 알려 주기 위해 후에 추가한 것이다. 기본적으로 기본 테스트에서는 LED 테스트를 하는데, 한 uCnt라는 변수를 0에서 256까지 증가하면서 LED1에 출력을 하는 것이다. 이때 LED1출력을 너무 빠르게 하면 눈에 보이지 않기 때문에 적당한 TimeDelay를 이용하여 표시하고 있다. 여기서는 코드의 모든 부분을 나타내지 않고 일부분만을 나타내었다.

```
SccOutStr("----Start----₩n");
LcdOutStr("----Start----");
WatchDogClear();
// first LED TEST //
SccOutStr("LED TEST₩n");
for( uCnt=0; uCnt < 256 ; uCnt++ )
{ SetLed1(uCnt); for( u = 0 ; u < 100 ; u++ ); }
SetLed1(0x00);
WatchDogClear();
.....
```

다음은 주 테스트 부분에 해당하는 코드이다. 이는 주 테스트에서 기술한 바와 같이 무한 루프를 돌면서 폴링 방식으로 필요한 일을 처리한다. WatchDogClear()를 주기적으로 보내며, 만일 수신한 정보가 있으면 그것을 적당한 메모리에 넣는다. 그리고 수신한 바이트가 명령의 종료를 알리는 문자이면 해당 명령을 확인하고 해당 명령에 따라서 적절한 처리를 한다.
수신한 데이터가 없을 경우에는 적절한 시간 분할을 통해서 LCD에 현재 루프 계수와 수신한 데이터를 LCD에 출력한다.

```
while(1)
{
        WatchDogClear();

// read message from scc and set the status for operation and memory
        if( HasScc1RxData() )
        {
            bTmp1=GetScc1();
            bBuffPage=(bBuffPage+1)%256;
            bBuff[bBuffPage]=bTmp1;
            if( bTmp1 == 0x0D )
            {
                if( IsSameBuff(bBuff,"Led1",4) ){ SetLed1(bBuff[5]); bBuffPage=0; }
                if( IsSameBuff(bBuff,"Led2",4) ){ SetLed2(bBuff[5]); bBuffPage=0; }
                ......
            }
        }
        else
        {
           uCnt++ ;
.....
             // display count //
            case 5  : LcdOutCtrl(0x80); break; // First Line
            case 6  : LcdOutStr("Count = "); break;
            case 7  : LcdOutByte(uCnt/100000000%10+'0'); break;
            case 8  : LcdOutByte(uCnt/10000000%10+'0'); break;
            case 9  : LcdOutByte(uCnt/1000000%10+'0'); break;
            case 10 : LcdOutByte(uCnt/100000%10+'0'); break;
            case 11 : LcdOutByte(uCnt/10000%10+'0'); break;
            case 12 : LcdOutByte(uCnt/1000%10+'0'); break;
            case 13 : LcdOutByte(uCnt/100%10+'0'); break;
            case 14 : LcdOutByte(uCnt/10%10+'0'); break;
```

```
            case 15 : LcdOutByte(uCnt%10+'0'); break;
            ..........
            // display buffer in LCD //
            case 0xE0 :
                        LcdOutByte(HighHexaChar(bLcdBuff[(bLcdPage+1)%8])); break;
          ......
    }
}
```

각 모듈의 제어를 위해서 메모리 맵이 있으며, 각 모듈을 제어하기 위해 주소를 일일이 쓰지 않고 사용하기 편리하게 하기 위해서 다음과 같이 C언어의 #define 문으로 주소를 정의한였다.

```
#define PROG_MEM_ADDR       (0x00000000)
#define PROG_MEM_SIZE       (0x0F0000)
#define LED1_ADDR           (0xB0000000)
#define LED2_ADDR           (0xB0100000)
#define LED3_ADDR           (0xC0200000)
#define CAN1_ADDR           (0xD0000000)
#define CAN2_ADDR           (0xD0100000)
#define CAN3_ADDR           (0xD0200000)
#define SCC1_ADDR           (0xC0000000)
#define SCC2_ADDR           (0xC0100000)
#define SCC3_ADDR           (0xC0200000)
#define LCD_DATA_ADDR       (0xE0000000)
#define LCD_CTRL_ADDR       (0xE0100000)
#define WD_ADDR             (0xE0200000)
#define SLIP1_START_ADDR    (0xC0000002)
#define SLIP1_END_ADDR      (0xC0000004)
#define SLIP1_DATA_ADDR     (0xC0000008)
#define SLIP1_SIZE_ADDR     (0xC0000010)
#define SLIP2_START_ADDR    (0xC0100002)
#define SLIP2_END_ADDR      (0xC0100004)
#define SLIP2_DATA_ADDR     (0xC0100008)
#define SLIP2_SIZE_ADDR     (0xC0100010)
```

또한 프로그램에서 각 주소에 대해 일일이 설정하는 것을 대신하여 #define 문을 이용하여 각 모듈에 대한 읽고 쓰는 과정을 추상화(Abstraction)하였다. 이를 이용하여 프로그램 작성이 편리하게 하였으며, 이를 이용하여 다른 시스템에서 같은 코드를 검증하는 데 있어서 간단한 변형만으로 수행할 수 있게 하였다.

```
#define SetLcdData(macV)          { *(uint*)LCD_DATA_ADDR  = (macV) ;}
#define SetLcdCtrl(macV){ *(uint*)LCD_CTRL_ADDR  = (macV) ;}
#define SetLed1(macV)             { *(uint*)LED1_ADDR = (macV) ;}
#define SetMem(macAddr,macV)  { *(uint*)(PROG_MEM_ADDR + (macAddr))= (macV);
}
#define WatchDogClear()           { *(uint*)WD_ADDR    = 0 ; }
.........
#define GetScc1()                 ((byte)*(uint *)SCC1_ADDR)
#define GetMem(macAddr)           ( *(uint*)(PROG_MEM_ADDR+(macAddr)))
..... ...
#define GetSlip1Size()            ((byte)*(uint *)SLIP1_SIZE_ADDR)
#define GetSlip1Data()            ((byte)*(uint *)SLIP1_DATA_ADDR)
#define GetSlip2Size()            ((byte)*(uint *)SLIP2_SIZE_ADDR)
#define GetSlip2Data()            ((byte)*(uint *)SLIP2_DATA_ADDR)
```

4 절 소형화 OBC 보드 분석

1 SEU 메커니즘 및 모델링

가 SEU 메커니즘

위성체가 우주환경에 직면하게 되면 먼 우주로부터 오는 우주선(cosmic ray)에 의한 고에너지 입자, 태양 활동에 의해 분출되는 입자, 또는 지구 자기장에 붙잡혀 있는 여러 에너지 입자들에 의해 위성체의 전자회로가 영향을 받게 된다. 이중에서 메모리에 발생하는 중요한 영향은 이들 고에너지 입자에 의해 메모리에 저장되어있든 정보가 "1" -> "0" 또는 "0" -> "1" 로 변경되는 SEU(Single Event Upset) 현상이다.

대부분의 메모리 소자는 그림 1과 같이 MOS(Metal Oxide Semiconductor) 트랜지스터를 이용하여NOT 게이트(gate) 2개를 서로 연결시킨 기본 구조에 바탕을 두고 있다. 이 회로는 2개의 안정된 상태에, 즉 "0" 또는 "1", 항상 머물러 있으므로 1bit의 2진 정보를 저장할 수 있다. 각각의 상태에서 그림 1의 4개의 MOS 트랜지스터 중 2개는 Turn-ON, 2개는 Turn-OFF 상태에 있게 된다. 이들 MOS 트랜지스터에 그림 2에서와 같이 고에너지 입자가 입사 되면 실리콘 반도체의 P-Substrate에는 이 입자가 지나가는 패스를 따라 반도체 내부에 있던 원자가 이온화 되어 전자(electron) 정공(hole) 쌍이 생성되고, 이때 발생된 전자, 정공 쌍의 일부는 일정 시간 후 재결합하여 없어지지만 일부는 트랜지스터의 Source 와 Drain 쪽으로 이동하여 Source에서 Drain 쪽으로 전류 펄스를 야기 시킨다. 이 전류 펄스의 크기가 일정 이상이 되면 OFF되어 있던 그림 1의 트랜지스터를 Turn-ON 시켜 이 회로의 상태가 다른 상태로 천이하게 된다("1" -> "0" 또는 "0" -> "1"). 이와 같이 회로의 상태가 천이함으로 인해 원래 저장되어있던 1bit 정보가 바뀌는 것이 SEU 현상이다.

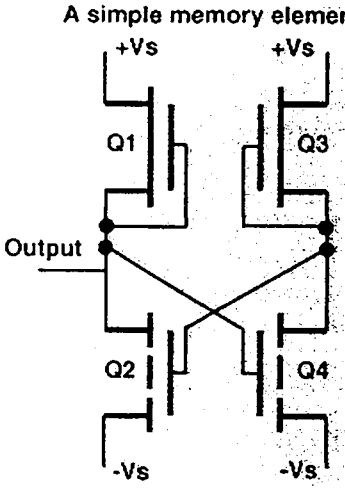


그림 111 메모리기본 구조

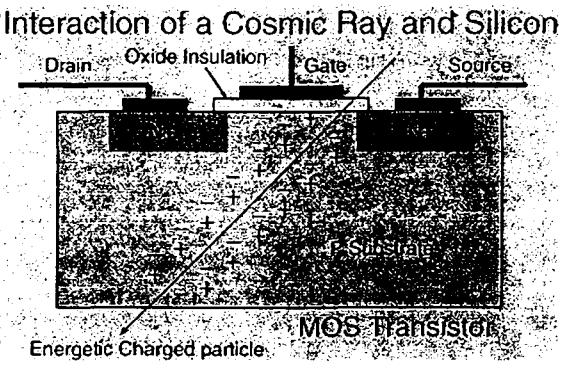


그림 112 고에너지 입자에 의한 SEU 발생 메커니즘

나 모델링

랜덤(random)하게 독립적으로 발생하는 많은 사건들은 주로 Poisson 과정(process)을 이용하여 모델링 된다[11,15]. 따라서 우주의 고에너지 입자에 의해 랜덤하게 발생하는 SEU 역시 Poisson 과정을 사용하여 모델링이 가능하다. Poisson 과정은 다음과 같은 확률 분포를 갖는 확률과정으로 각 사건은 과거에 발생한 사건에 관계없이 독립적으로 발생하며, 사건의 도착시간은 지수 분포를 갖는다.

$$P(n,\lambda,t)=(\lambda t)^{n}\frac{e^{(-\lambda t)}}{n!}$$

여기서 λ 는 SEU 발생율이며, $P(n,\lambda,t)$는 t 시간 간격에 n개의 SEU가 발생할 확률이다.

2 메모리 SEU 확률 해석 및 워쉬 주기

가 메모리 EDAC 알고리즘

대부분의 상용 위성들은 SEU를 방지하기 위해 SOS(Silicon On Sapphire) 기법 등을 이용한 Radiation Hardened 또는 Radiation Tolerant 급의 메모리 소자를 사용한다. 하지만 이러한 메모리를 사용하면 SEU를 소자 차원에서 극복할 수 있으나 너무 고가라서 우리별 씨리즈나 과학위성 씨리즈와 같은 저비용 소형 위성에서는 사용하기에 무리가 있다. 따라서 우리별 씨리즈나 과학위성 씨리즈에서는 훨씬 저

렴한 Military 급의 메모리 소자에 EDAC(Error Detection and Correction) 로직을 추가하여 SEU를 극복하도록 함으로써 비용을 줄이도록 설계되었고, 본 연구에서는 SEU 극복을 위한 TMR 구조를 채택하여 설계하였다. 본 연구에서 개발한 탑재 컴퓨터에서 채택하고 있는 TMR 방식은 1비트 정보를 3비트에 저장하고 이를 이용하여 SEU를 극복하는 방식이다. 이 방식에 의하면 3비트 중 1비트 오류를 탐지하고 복구하며, 2비트 오류는 하나 복구시 오류 값을 갖고, 3비트 오류는 감지할 수 없다.

그러므로 TMR을 이용한 EDAC 알고리즘은 그 특성상 3bit마다 1bit의 SEU를 극복할 수 있지만, 2bit 이상의 SEU는 복구 할 수 없으므로, SEU에 의해 메모리에 저장된 1bit의 값이 변경된 후 이것이 EDAC 로직에 의해 복구되기 전에 또다시 SEU에 의해 이들 3bit중 어느 1bit 이 변경되면 전체적으로 2bit의 오류가 발생하여 EDAC 알고리즘으로는 오류를 복구할 수 없는 경우가 발생한다. 즉 메모리에서 발생한 SEU를 적당한 시간 내에 복구하지 않으면 오류가 누적되어 복구할 수 없는 경우가 발생할 수 있다. 이와 같은 EDAC 알고리즘의 단점 때문에 탑재 컴퓨터의 시스템 소프트웨어는 메모리 전체 데이터가 항상 오류가 없도록 하기 위하여 정기적으로 전 메모리를 읽어 오류를 복구하고 다시 저장하는 작업(memory wash)을 수행하도록 하여야 한다.

따라서 전체 메모리를 워쉬(wash)하는 주기를 어떻게 설정하느냐에 따라 오류 복구 가능성이 달라진다. 즉 주기를 길게 하면 2bit 이상의 SEU 가 발생하여 오류를 복구할 수 없는 가능성이 커지고, 주기를 짧게 하면 오류를 복구할 수 없는 가능성은 줄어들지만 시스템 소프트웨어의 부담이 커져(메모리를 wash하는데 일정 시간을 소모하므로) 탑재컴퓨터의 다른 소프트웨어가 수행될 시간이 줄어드는 단점이 있다. 따라서 메모리 워쉬 주기는 이들을 고려하여 적절하게 설정되어야 한다.

나 메모리 SEU 확률

메모리의 각 셀(cell)에서 SEU 발생이 Poisson 과정을 따른다고 하면 메모리의 각 셀에서 t 시간 내에 n개의 SEU가 발생할 확률은 다음과 같다.

$$\alpha_n(\lambda, t) = (\lambda t)^n \frac{e^{(-\lambda t)}}{n!}$$

여기서 λ는 메모리 각 셀에서SEU 발생율이다.

따라서 메모리의 각 셀에 저장되어 있는 2진 정보가 SEU에 의해 변경되었을 확률은 다음과 같다.

$$p(\lambda, t) = \sum_{k=0}^{\infty} \alpha_{2k+1}(\lambda, t)$$

$$= \sum_{k=0}^{\infty} (\lambda t)^{(2k+1)} \frac{e^{(-\lambda t)}}{(2k+1)!} = e^{(-\lambda t)} \sum_{k=0}^{\infty} \frac{(\lambda t)^{(2k+1)}}{(2k+1)!}$$

각 셀의 2진 정보가 변경되지 않을 확률은 아래의 식으로 표현되고,

$$q(\lambda, t) = 1 - p(\lambda, t) = \sum_{k=0}^{\infty} \alpha_{2k}(\lambda, t)$$

$$= \sum_{k=0}^{\infty} (\lambda t)^{2k} \frac{e^{(-\lambda t)}}{(2k)!} = e^{(-\lambda t)} \sum_{k=0}^{\infty} \frac{(\lambda t)^{2k}}{(2k)!}$$

지수 함수의 Taylor Series 는 다음과 같으므로,

$$e^{x} = 1 + x/1! + x^2/2! + x^3/3! + x^4/4! + \cdots$$

$$e^{-x} = 1 - x/1! + x^2/2! - x^3/3! + x^4/4! - \cdots$$

확률 p(λ,.t), q(λ,.t) 는 다음의 수식으로 나타낼 수 있다.

$$p(\lambda, t) = [\ 1 - e^{-2\lambda t}\ ]/2$$

$$q(\lambda, t) = [\ 1 + e^{-2\lambda t}\ ]/2$$

본 연구에서 개발한 탑재 컴퓨터의 주 메모리에서 채택하고 있는 EDAC 알고리즘을 고려하면 TMR Code로 만들어진 3bit 의 메모리 셀 중에서 2bit 이상이 SEU에 의해 원래값과 다르게 될 확률은 다음과 같이 유도된다.

$$\gamma(\lambda, t) = \sum_{k=2}^{3} (C_k \cdot p(\lambda, t)^k \cdot q(\lambda, t)^{3-k}) = 3 \cdot p(\lambda, t)^2 \cdot q(\lambda, t) + p(\lambda, t)^3$$

즉 $\gamma(\lambda, t)$ 는 3bit로 이루어진 메모리 셀들이 TMR을 이용한 EDAC 알고리즘으로는

복구 불가능한 SEU 확률이다.

따라서 메모리의 전체 크기가 M bytes 이고 각 메모리 셀에서의 SEU 발생율이 일때 t 시간 동안 전체 메모리을 워쉬 하지 않았을 경우 M bytes의 메모리 중에서 EDAC 알고리즘으로 복구 불가능한 SEU가 1개 이상 있을 확률은 다음과 같다.

$$\Gamma(\lambda, t) = 1 - [1 - \gamma(\lambda, t)]^M$$

$\Gamma(\lambda, t)$ 는 메모리 워쉬주기가 t 일때 전체 메모리에 복구 불가능한 SEU 가 있을 확률로 주기 t 가 길어지면 이 값은 증가하고 t 가 작아지면 이 값은 감소한다.

다 메모리 워쉬 주기 선정

탑재 컴퓨터의 시스템 소프트웨어가 t 시간 주기마다 전체 메모리를 워쉬하는 것은 시간적인 오버헤드(overhead)를 요구하며 이것은 다른 소프트웨어가 수행될 시간을 빼앗기 때문에 시스템의 성능을 떨어뜨린다. 이 오버헤드는 다음의 식으로 나타낼 수 있다.

$$\Theta(t) = \frac{\sigma}{t} \quad (\sigma \leq t)$$

여기서 σ 는 전체 메모리를 워쉬하는데 걸리는 시간이며, t 는 워쉬 주기이다.

오버헤드 $\Theta(t)$는 워쉬 주기가 길수록 적어지고, 짧을수록 커져 주기가 σ 가 됐을 때 최대가 된다. 워쉬 주기가 σ 보다 같거나 작게 되면 CPU의 모든 시간을 메모리 워쉬에만 사용하게 되므로 다른 소프트웨어를 수행할 시간이 없어지게 된다. 따라서 SEU 누적을 피하기 위한 메모리 워쉬 주기는 확률 $\Gamma(\lambda, t)$ 와 오버헤드 $\Theta(t)$ 값을 동시에 작게하는 주기 t 를 선정해야 한다. 하지만 주기 t 를 증가시키면 오버헤드 $\Theta(t)$ 는 감소하는 반면 확률 $\Gamma(\lambda, t)$ 는 증가하고, t 를 감소시키면 반대로 확률 $\Gamma(\lambda, t)$ 는 감소하지만 오버헤드 $\Theta(t)$ 는 증가한다. 따라서 다음과 같은 성능 함수(performance index)를 최소화 시키는 주기 t 를 선정하는 것이 가장 바람직하다.

$$J = f(\Gamma(\lambda, t)) + g(\Theta(t))$$

여기서 $f(\Gamma(\lambda, t))$, $g(\Theta(t))$ 는 각각 $\Gamma(\lambda, t)$ 와 $\Theta(t)$ 의 중요도를 나타내는 함수로 성능함수 J 에의 기여 정도를 결정한다. $\Gamma(\lambda, t)$ 와 $\Theta(t)$ 가 J 에 같은 비중으로 기여

하는 가장 단순한 경우의 성능함수는 다음과 같다.

$$J = \Pi(\lambda, t) + \Theta(t)$$

메모리에서의 SEU 확률 $\Pi(\lambda, t)$ 와 CPU의 오버헤드 $\Theta(t)$ 를 구하기 위해서는 본 과제에서 개발한 탑재 컴퓨터에서의 와 가 필요하다. 메모리 각 셀에서의 SEU 발생율은 우리별 씨리즈와 과학위성 씨리즈의 탑재 컴퓨터에서의 SEU 발생율을 조사하면 대략적으로 그 값을 추정할 수 있다. [그림]은 우리별 1호의 탑재 컴퓨터에서 1993년 ~ 1994년까지 관측된 일별 SEU 검출 수를 나타낸다[2,3]. SEU는 180일 근처와 270일 근처에서 큰 값을 보이는데 이것은 태양 플레어에 의한 양성자 플럭스(flux)의 증가에 기인한 것으로 추정된다[2]. [그림]은에 의하면 180일 부근과 270일 부근의 특별한 경우를 제외하면 메모리에 발생한 SEU 개수는 대체적으로 300 SEUs/Day 이내라고 할 수 있다. 우리별 1호의 프로그램 메모리는 512Kbytes 이므로 메모리 각 셀의 SEU를 분으로 나타내면:

300 SEUs/ 512Kbytes /24 /60 4.06901 10 7 SEUs/bit min

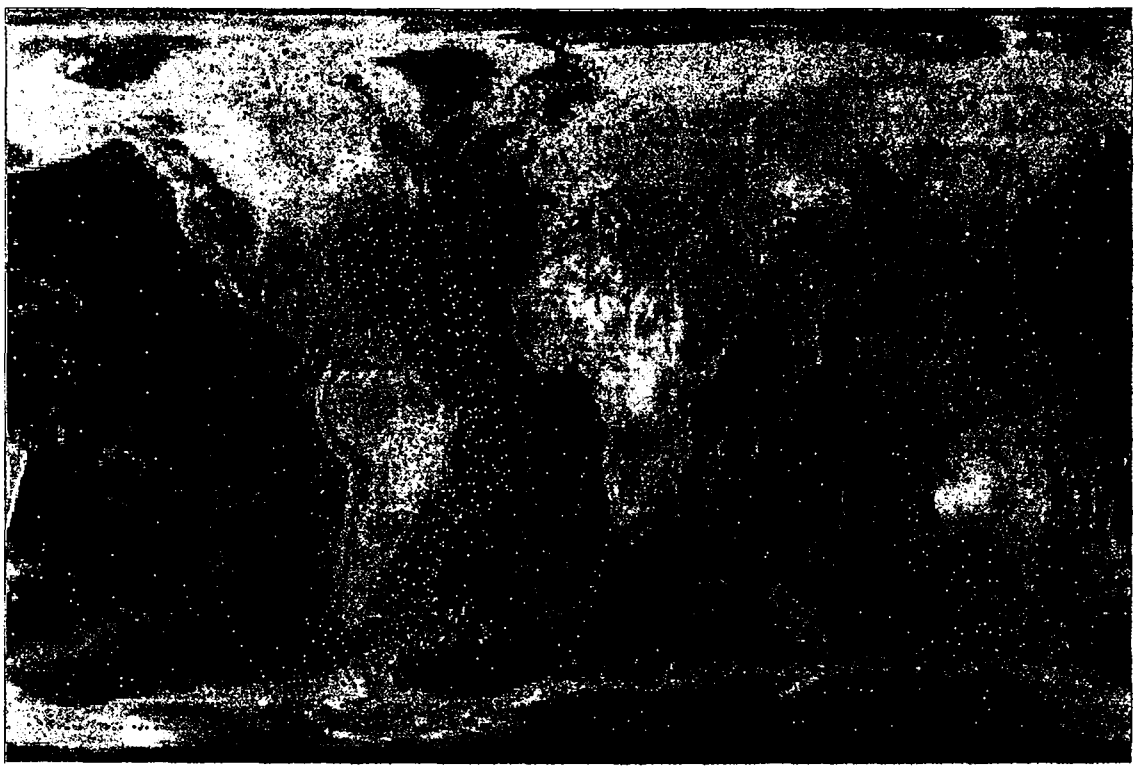


그림 124 과학기술위성 1호의 SEU 발생 장소(2004년 5월20일까지 3달간)

그런데 1 궤도상에서 SEU 가 발생하는 궤도상의 위치와 발생 빈도사이의 관계를 보면 그림에서 보듯이 전 궤도상에서 고르게 분포하는 것이 아니라 SAA(South Atlantic Anomaly) 부근과 극 지방에서 집중되는 것을 알 수 있다[2,3]. 특별히 과학기술위성1호의 Wash주기는 30분 정도이며, 그러므로, 위의 그림에서 나타난 점들은 30분이라는 시간 차가 있을 수 있다. 이것을 감안하여 보면 특히 SAA 지방에 집중적으로 나타나는 것을 알 수 있다. 따라서 위성이 SAA와 극지방을 지나는 시간이 전체 궤도상의 시간의 대략 1/100 이라고 가정하고 여기서 대부분의 SEU가 발생한다고 하면 위성이 궤도상에서 경험하는 최대 SEU는 다음과 같다.

$$4.06901\times10^{7}/100 = 4.06901\times10^{5}\ \text{SEUs/bit min}$$

따라서 메모리 각 셀에서의 발생하는 SEU 발생율은 대략적으로 추정할 수 있다.

$$4.06901\times10^{5}\ \text{SEUs/bit min}$$

위에서 추정한 SEU 발생율은 상한값을 상당히 보수적으로 계산한 값이므로 근사적으로 적용할 수 있을 것으로 판단된다.

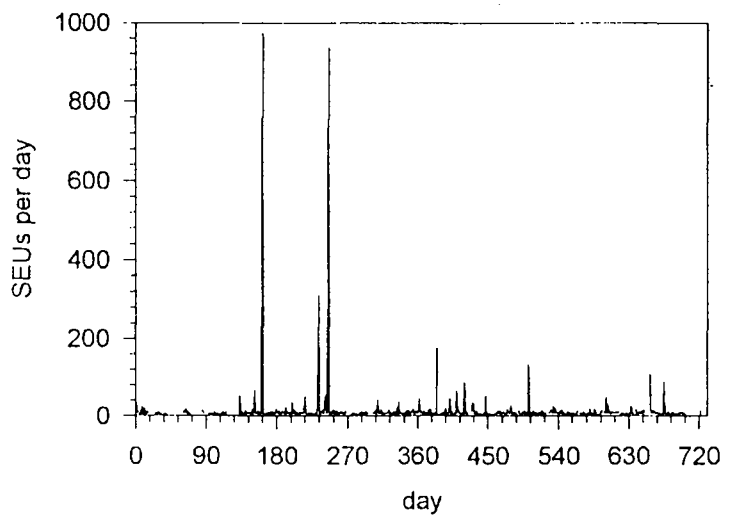


그림 125 우리별 1호에서의 일별 SEU 발생 횟수

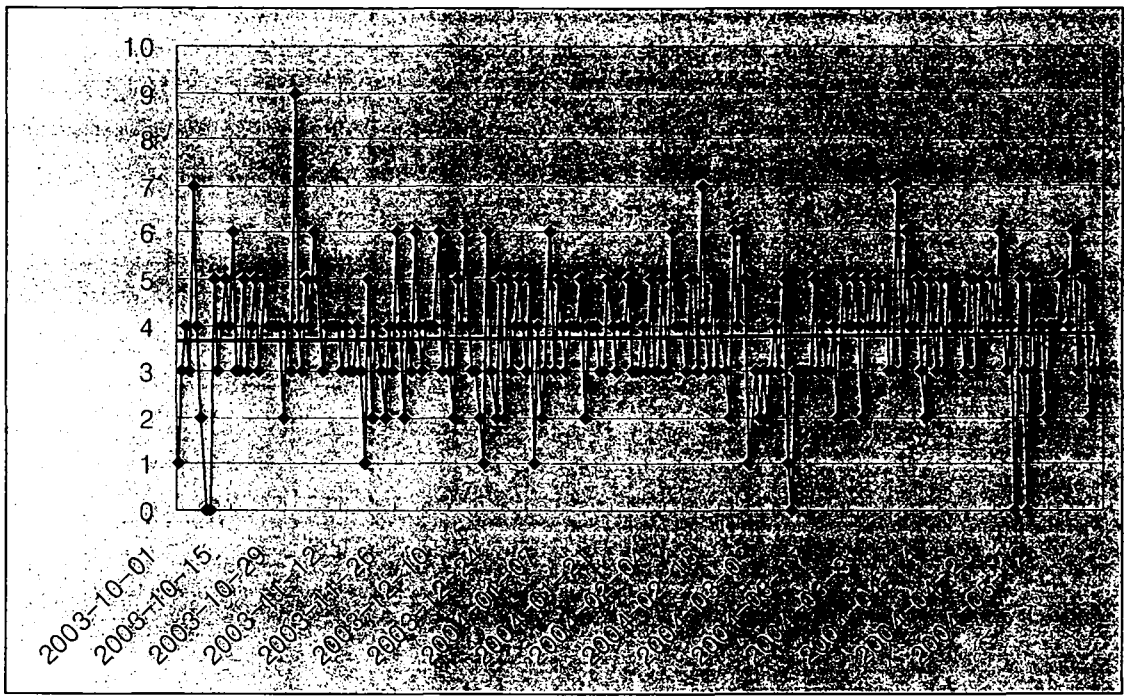


그림 126 과학기술위성 1호에서의 일별 SEU 발생 횟수 (평균 3.745763)

반면, 위성 궤도의 차이, 사용된 메모리의 종류, 위성내에서의 탑재 컴퓨터의 위치, 위성 외부 프레임의 두께등의 조건이 우리별 1호와 과학위성 1호가 우주에서 경험한 실제 SEU 발생율은 차이가 발생한다. 최근 개발된 과학위성 1호의 경우 운용결과 하루평균 3.745763 번의 SEU 가 발생하며, 한 비트당 분당 발생률은

$$3.745763/4\times10^{6}/24/60 = 6.050306\times10^{-10}\ \text{SEUs/bit . minute}$$

이다. 그런데 밀도가 높은 곳에서 100배 정도에서 집중적으로 발생한다고 하면 분당 SEU 발생률은

$$\lambda = 6.50306\times10^{-8}$$

이다.

다음으로 메모리 워쉬가 CPU에 주는 부담을 계산하기 위하여 탑재 컴퓨터의 CPU가 메모리를 엑세스(access)하는데 필요한 클럭 사이클(clock cycle)을 계산한다. 탑재 컴퓨터의 CPU는 PowerPC603로 32bit 데이터(data) 버스를 가지고 있다. 그리고 PowerPC603이 주 메모리에 읽기/쓰기(read/write)를 하는데 에는 각각 6 클럭 사이클을 요구하며[7], 메모리를 워쉬하는 것은 메모리의 특정 부분을 읽고 쓰는 과정을 요구하므로 4bytes (32bit)의 메모리를 엑세스 하는데는 최소 6*2 = 12개의 클럭 사

이클을 필요로 한다. 뿐만 아니라 전체 메모리를 워쉬하기 위해서는 매 4 bytes 마다 어드레스를 증가시키는 과정과 주 메모리를 읽는 Instruction Fetch등의 과정이 필요하다. 따라서 4bytes의 메모리를 워쉬하기 위한 클럭 사이클은 충분한 여분의 시간까지 고려하면 다음과 같이 나타낼 수 있다.

4 bytes 워쉬 = Read/Write + 2*Instruction fetch + Address increasing + etc

= 12 cycles + 2*Instruction fetch + Address increasing + etc

= 40 clock cycles

프로그램 메모리 크기가 1M bytes 이므로 전체 메모리를 워쉬하는데는 다음의 클럭 사이클을 필요로 한다.

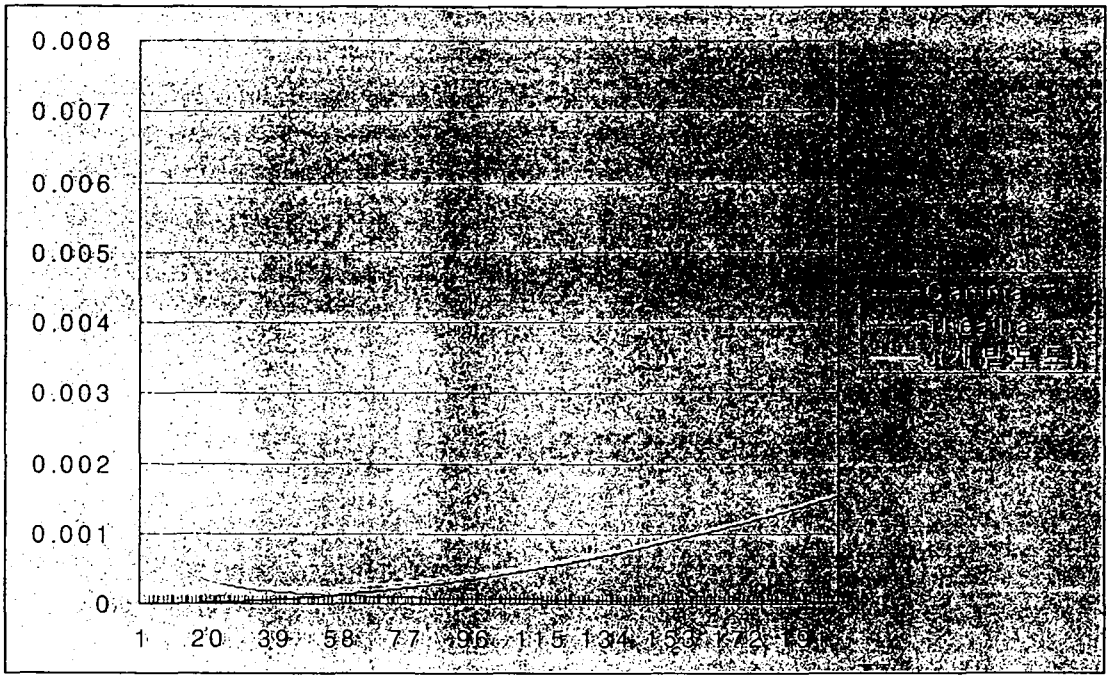
$$40\times10^6/4 \text{ cycles} = 10^7 \text{ cycles}$$

탑재 컴퓨터의 주 클럭 주파수는 25Mhz 이고 이것의 5배인 150Mhz 가 실제 CPU의 클럭으로 사용된다. 그러므로 매 분당 오버헤드의 상한값을 다음과 같이 보수적으로 추정할 수 있다.

$$\sigma = 10^7/25\times10^6/60 = 0.006667/\text{min}$$

그림은 앞 절에서 추정된 SEU 발생율 $\lambda = 6.50306\times10^{-8}$ SEUs/bit minute 와 오버헤드 파라미터 $\sigma = 0.00667$ 을 사용했을 때 워쉬 주기 t 의 변화에 따른 $\Gamma(\lambda, t)$ 와 $\Theta(t)$를 나타낸 것이다. 이 변화를 살펴보면, 워쉬 주기가 짧은 왼쪽에서는 개발 보드의 성능 함수는 오버헤드 $\Theta(t)$ 의 영향을 주로 받으며, 워쉬 주기가 길어지는 오른쪽으로 갈수록 개발 보드의 성능 함수는 메모리 복구 불가능 확률인 $\Gamma(\lambda, t)$ 에 주로 영향을 받는 것을 알 수 있다.

여기서는 워쉬 주기가 50분 근처에서 두 곡선이 서로 교차하므로 이 부근의 값을 주기로 선택하는 것이 $\Gamma(\lambda, t)$ 와 $\Theta(t)$ 을 동시에 적게 할 수 있다. $\lambda = 6.50306\times10^{-8}$ 인 경우 J 가 최소가 되는 워쉬 주기는 20 ~ 70분 부근으로, $\Gamma(\lambda, t)$ 와 $\Theta(t)$ 가 서로 교차하는 부분을 포함한다. 따라서 λ 와 σ 의 추정 오류를 고려하더라도 대략 4 ~ 25 m 사이의 값을 워쉬주기로 선정하는 것이 가장 타당하다고 할 수 있다.



3 과학기술위성 1호 모델과의 비교

메모리를 같은 것을 쓴다는 것을 가정으로 과학기술위성 1호의 모델링과 비교하면, 비트당 발생률은 $\lambda = 6.50306 \times 10^{-8}$ SEUs/bit minute 과 같다고 할 수 있다. 하지만 $\Gamma(\lambda, t)$는 메모리 워쉬 주기가 t 일때 전체 메모리에 복구 불가능한 SEU 가 있을 확률에서 1M에 해당하는 전체 메모리는 2M byte 로 본 프로젝트에서 개발한 3M 바이트와 다르다.

$$\gamma(\lambda, t) = \sum_{k=2}^{7} \left({}_7C_k \cdot p(\lambda, t)^k \cdot q(\lambda, t)^{7-k} \right)$$

$$\Gamma(\lambda, t) = 1 - [1 - \gamma(\lambda, t)]^{2M}$$

또한 과학기술위성 1호 탑재 컴퓨터의 주 클럭 주파수는 9.8Mhz 이고 이것의 1/2인 4.9Mhz가 실제 CPU의 클럭으로 사용되므로 매 분당 오버헤드의 상한값은 σ = 0.034 로 다르다. 즉 본 연구 σ = 0.006667, 과학기술위성 1호 σ = 0.034 이다.

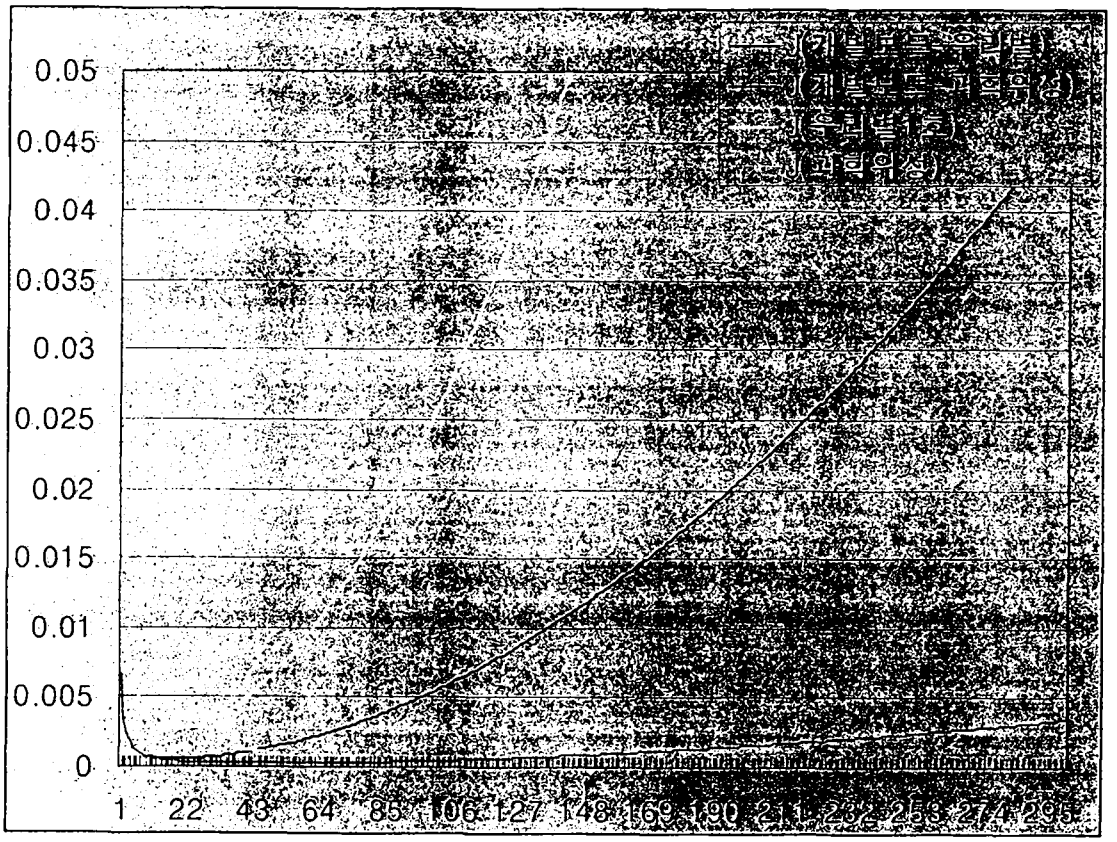


그림 128 개발보드와 과학위성1호와의 성능 함수 추이

그림은 램에 따른 확률함수와 Wash로 인한 Overhead에 의해 생성되는 성능함수 추이를 나타낸 것이다. 먼저 축을 설명하면 x 축은 분 단위의 워시 시간 간격을 나타내며, y 축은 성능 함수의 값을 나타낸다. 그래프 상에서 시간 간격이 짧은 기간에 해당하는 왼쪽 부분은 주로 워쉬 오버헤드에 의해 영향을 받는 구간이며, 워쉬 기간이 길어진 기간에 해당하는 오른쪽 부분은 위험도에 영향을 받는 것을 나타내고 있다. 이를 기반으로 4개의 추이 곡선이 있는 데, 우리별 1호에서 사용한 메모리와 과학위성에서 사용한 메모리 두 종류와 과학위성에서 사용한 방식과 개발한 보드에서 사용한 방식의 차에 따라 4가지 종류를 나타내고 있다. 왼쪽 그림을 기준으로 제일 위에 있는 그래프는 과학위성1호에서 우리별 1호의 램을 사용한 경우에 해당한다. 그 아래 그래프는 과학위성 1호에서 현재의 과학위성 1호의 램을 사용한 경우에 해당한다. 그 아래 그래프는 현재 개발한 보드에서 우리별 1호의 램을 사용한 경우에 해당하며, 가장 아래의 그래프는 현재 개발한 보드에서 과학위성 1호의 램을 사용한 경우에 해당한다.

위의 그림에서 살펴보면 과학위성에서 우리별 메모리를 쓸 경우 성능 함수값을 0.005 로 한다면, 7분에서 40분 정도에 Wash주기를 사용하는 것이 좋다. 이에 반해 과학위성에서 자체 메모리를 쓸 경우는 7분 이상의 Wash 주기를 사용하는 것이 좋다. 그러나 본 연구에서 개발한 보드에서 우리별 메모리를 사용할 경우 3분에서 90분 정도에 Wash주기를 사용하는 것이 좋다. 개발한 보드에서 과학위성용 메모리를 사용한다면, 3분 이상에서 Wash주기를 사용하는 것이 좋다. 여기서 눈여겨 볼 것은 개발한 보드와 과학위성에서 교차하는 지점이 있다는 것이다. 이것은 과학위성에서 사용한 메모리가 SEU에 강하기 때문이다.

본 연구 결과에서 성능 함수 추이는 중요도 함수인 f 함수와 g 함수에 의해서 달라질 수 있다. 하지만 성능함수추이에 대한 다른 해석을 한다면, 본 연구에서 개발한 방식에서 과학위성보다 SEU에 약한 저 가격의 메모리를 쓰더라도 30분 정도의 적당한 워쉬 주기를 가짐으로써 이를 극복할 수 있다는 것을 보여준다. 한 번 더 정리하자면 구조적인 문제를 포함하여 실험상 나타난 SEU 발생률은 우리별에서 4.06801×10^{5}이고 과학위성에서 6.50306×10^{-8} 로서 비트당 SEU 발생 비율은 625.706975로서 우리별 1호에서 사용한 SRAM이 625배 정도 더 많이 발생한다. 이러한 메모리 조건에서 본 연구에서 개발한 보드의 해석 결과 30분 이하의 적당한 워쉬 주기를 갖는다면, 본 연구에서 개발한 보드는 우리별 1호의 SRAM을 사용하더라도 과학 위성 1호와 유사한 성능 함수를 가질 수 있다는 것을 나타낸다. 이는 저 가격의 메모리를 이용하더라도 현재의 과학 위성 수준의 위성 운용이 가능하다는 것을 나타낸다.

5 절 연구 결과

본 과제의 연구 목표는 FPGA를 이용하여 우주용 OBC를 소형화/경량화하여 실제 위성에 적용할 수준의 OBC 보드를 개발하는 것이다. 세부적으로는 다음의 조건을 만족하여야 한다.

- CPU: 32bits, 50MIPS 이상
- EDAC: FPGA 구현
- Communication Controller: 100Kbps 이상, FPGA 구현
- Watchdog Timer: FPGA 구현
- 각종 제어 신호: FPGA 구현
- Fault Tolerant 구조
- 무게: 900g 이하
- 크기: 200x300x30mm
- 공급전압: 5±0.25V
- Module Interface: 9.6K, 19.2K, 59.6K, 100KBps

위의 조건을 위해서 만족하기 위해서 본 연구에서는 보드 개발을 위한 준비 작업으로 우주 환경에 대한 연구를 하였으며, 그에 따라 우주환경에 맞는 부품을 선정하고 보드 설계를 하였으며 FPGA 로직을 통해 보드를 소형화 경량화 하였다. 또한 개발된 보드에서 메모리 종류에 따라 SEU를 극복하기 위한 Wash 주기 분석을 하였다.

1 보드 개발

보드는 크게 PCB 디자인, 보드 개발, FPGA 로직 구현, 보드 검증용 S/W개발 의 4가지 단계를 거쳐서 개발 되고 검증 되었다. 개발된 보드의 CPU는 PowerPC 603E를 사용하며 메인 CLOCK은 25Mhz이고 내부 Clock이 150Mhz에서 동작이 가능하므로 RISC 특성상 50 MIPS 가 가능하므로 CPU Spec을 만족한다. 또한 FPGA를 통하여 메모리를 TMR기법으로 보호하므로 EDAC을 구현하였다. 그리고 FPGA를 통하여 115200 bps의 SCC를 구현하여 100Kbps 이상의 통신이 하게 개발하였으며, 기타 통신을 위해서 SLIP 을 처리하거나, CAN core부분에 대해서 처리하도록 개발하였다. 또한 FPGA를 통하여 WatchDog Timer를 구현

하였다. 그리고 두 개의 CPU를 이용하여 Fault 에 대해서 처리할 수 있는 구조를 갖게 하였으며, 보드 점검을 위하여 LED 나 LCD등의 표시가 가능하게 제어하는 보드를 개발하여 Spec을 만족한다,

가· PCB 디자인

본 과제에서는 VeriBest를 이용하여 크게 3번의 보드를 디자인 하였다. 8751, 603 1차, 603 2차에 해당하는 보드이다. 이 보드의 개발에는 현재 발사되어 궤도상에서 임무를 충실히 수행하는 과학기술위성 1호의 Artwork 설계 규격을 만족시키도록 PCB가 디자인 되어 개발되었다. 이를 위하여 본 과제에서는 STD235, IPC2221의 규격을 기준에 대한 연구를 하였고, 이에 따른 PCB 디자인 규격에 대해 기술하였으며, PCB 층 구성에 대한 내용도 기술하였다. 그러므로 본 과제의 연구를 통하여 우주에 대한 PCB 디자인에 대한 참고 자료가 될 수 있다.

나 보드 개발

본 과제에서는 크게 3번의 보드를 개발하였다. 먼저 디버깅용 8751 CPU 마이크로 프로세서 기반 보드이다. 이 보드는 FPGA에 대한 로직 검증을 위해 개발하였다. 이 보드를 통하여 8751 상에서의 MultiCPU 의 동작여부와 TMR 메모리 구조의 구현을 확인할 수 있었다. 2차 보드는 PowerPC 603e를 기반으로 한 우주용 OBC를 제작하였다. 이 보드는 1차 보드에서의 검증된 부분에 대해 PowerPC에도 동작하는지에 대한 여부와 소형화 및 경량화를 위한 기반 기술로서 개발하였다. 이 2차 보드는 크기보다는 기능적인 동작을 위해서 개발 되었으며 특히 디지털 로직 분석기 등을 통해 쉽게 보드의 동작 여부를 파악하기 위해 디버깅을 위한 많은 핀들이 보드 상에 있었다. 3차 보드는 PowerPC에 대해 소형화/경량화를 하였다. 특히 FPGA는 2차 보드에 비해 3개로 한 개 줄였으며, 디버깅을 위한 부분을 외부로 옮겼다. 이를 통해서 OBC 보드의 크기는 184*200 mm이며, 초기 보드의 제한 값인 200x300 면적 보다 약 30% 이상 작은 면적의 보드로 개발 되었다. 만일 TMR을 쓰지 않고 (7,4) Hamming Code를 사용한다면, IO 핀이 많이 필요하지 않으므로 FPGA수는 더 줄일 수 있고 이로 인해 초기 예상치보다 50% 이상 더 줄일 수도 있으리라 사료된다. 또한 두 개의 CPU 구조를 사용하지 않고 하나의 CPU 구조를 사용한다면 예상치보다 70%까지 줄일 수 있으리라 사료된다.

즉, 본 과제를 통하여 TMR 구조의 PowerPC의 OBC보드를 개발하였으며, 이러한 개발 과정에서 우주용 회로 설계 기술을 확보 하였으며, 고 신뢰도 시스템 기술을 축적하고 우주용 시스템의 경량화/소형화 기술을 습득하였다.

다 FPGA 로직 구현

본 과제에서 소형화 경량화를 위한 많은 부분이 FPGA의 로직 구현을 통하여 이루어 졌다. 최초 보드에서는 인터페이스를 위한 많은 칩들이 사용되었으며, 이들로 인한 공간과 무게가 Overhead가 있었다. 하지만, 이러한 부분들을 FPGA로 처리하므로서 공간과 무게를 줄일 수 있었다. 뿐만 아니라 FPGA를 통한 로직구현을 통하여 우주용 FPGA 회로 설계 기술을 확보하였으며 내 방사선 회로 및 시스템 설계 기술들을 습득할 수 있었다.

본 과제에서 FPGA를 통해 구현한 로직은 SRAM의 SEU 극복을 위한 TMR로직, 프로그램의 오동작을 극복하기 위한 WatchDog Timer 로직, SRAM 인터페이스 로직, SCC 통신 로직, SLIP 통신 로직, CAN core 로직, 보드 검증을 위한 LED 제어 로직과 LCD 제어 로직 등이 있다.

라 보드 검증용 S/W 개발

개발된 보드의 검증은 일차적으로 눈으로 보드를 검사(Eye Inspection)하며, 각종 라인이 제대로 설계되었는지 검증 단계를 거치며, 납땜 후 각각의 부분들이 제대로 동작하는지에 대한 신호에 대한 테스트를 거치며, 결국 CPU에서 프로그램 동작에 의한 보드 검증을 하게 된다.

이러한 과정들은 초기 오실로 스코프나, 디지털 로직 분석기 등의 도움을 받아 처리하며, 파트들이 어느 정도 동작하게 되면서 LED, LCD 등의 도움을 받고, 결국 S/W에 의한 도움을 받게 된다. 그러므로 본 연구에서는 개발된 보드 상에서 PowerPC 603e 프로그램을 수행하여 검증을 하였고 이를 위하여 보드 검증용 S/W를 개발하였다.

검증을 위해 개발한 S/W로는 기본적인 상태 변환 스테이트를 VHDL로 변환해 주는 StateToCode라는 프로그램과, PowerPC 603e의 C 코드를 미리 시뮬레이션 해주는 Environment 프로그램과, CAN에 대해서 Clock Level Simulation 해주는 Can Simulator을 개발하였으며 또한 보드제어를 위한 보드 제어 프로그램도 개발하였다.

2 인공위성의 환경에 대한 연구와 SEU 영향 분석

인공위성은 지상의 일반적인 환경이 아니라 우주라는 특수한 환경에서 일정궤도를 돌며 동작을 한다. 또한 이는 본 과제에서 우주용 OBC 개발에 커다란 영향력을 끼친다. 본 과제에서는 우주 환경에 대한 연구를 하였으며, 이를 극복하기 위한 방식으로 TMR을 이용하였고 메모리에 따라 Wash 주기도 분석 하였다.

가 우주 환경에 대한 연구

본 과제는 위성의 동작 환경과 규격에 대한 조사를 하여 기술하였다. 이러한 기술 부분은 지

구 주변 우주환경과 발사 환경 등에 대해 언급하였으며, 전자 부품의 규격에 대해 기술 하고 있으며, 전자기파에 의해 생길 수 있는 소자의 변화에 대해서도 기술하고 있으며, 이를 극복하는 방식에 대해서도 기술하고 있다. 이러한 본 과제의 연구 결과 우주 환경이 전자 회로에 미치는 영향, 내 방사선 회로 및 시스템 설계를 위한 기술을 가질 수 있게 되었다. 또한 본 과제의 결과를 통하여 우주 환경에서 사용되는 기기에 대한 개발 시 필요한 기술을 이용할 수 있다.

나 SEU에 따른 Wash 주기 분석

본 연구에서는 우리별 1호와 최근 개발된 과학기술위성1호에서의 SEU 발생률을 바탕으로 개발된 보드의 Wash 주기에 대한 분석 하였다. 이때 로그에 의하면, 우리별 1호의 메모리에 발생할 SEU 확률은 과학기술위성1호 SEU 확률보다 무려 600배가 높은 메모리를 사용하고 있다. 이러한 상태에서 분석 결과 과학기술위성 1호보다 SEU에 대해 무려 600배가 약한 메모리를 이용한다 하더라도 과학기술위성 1호가 가지는 약 30분의 Wash 주기를 가질 경우 개발한 보드가 과학기술위성 보다 성능 함수가 좋게 나왔다. 이것은 좀 더 빠른 CPU를 사용하고 Hamming 코드 대신 TMR 의 기능을 사용하면 값싼 메모리를 가지고 저궤도 위성 같은 곳에서 사용할 수 있다는 것을 의미하며 테스트를 통하여 위성 개발비용을 줄일 수 있다는 것을 보여준다.

제 4 장. 목표 달성도 및 관련 분야에의 기여도

1 절 연구 개발 목표의 달성도

본 과제에서 최초 제안한 연구 목표 및 평가의 착안점에 대비하여 개발을 하였다. 그 뿐만 아니라, 우주 환경에 대해 조사를 하고 이에 따른 PCB를 설계 및 개발 하였으며, 검증 보드를 비롯하여 2차에 걸친 PowerPC 보드를 개발하여 보드의 소형화 경량화를 이루었다. 또한 보드 검증을 위해 LED, LCD등의 디버깅용 로직을 추가로 구현하였으며, 철저한 S/W 검증 툴과 코드 생성 툴을 개발하여 제안한 목표 이외의 연구 및 개발하였다.

년도	평가의 착안점	달성도 (%)	개발 내용
1차년도	◦FPGA 구현 Logic 범위설정 및 타당성 검토(CPU Core 포함)	100	FPGA를 통한 구현 로직 검토
	◦Fault Tolerant를 위한 방법 제시(구조적 측면)	100	두 개의 CPU 구조 설계
	◦OBC 시스템 구조 및 접속부 설계	100	8751을 통한 OBC 시스템 설계
2차년도	◦EDAC 회로의 VHDL 구현: (7,4)-Hamming code 이상	100	SEU에 대한 TMR 구현
	◦Communication Controller 회로의 VHDL 구현	100	FPGA를 이용한 SCC 구현 SLIP, CAN core 구현
	◦ Watchdog Timer 및 각종 제어로직의 VHDL 구현	100	Watchdog 기능 및 제어로직 구현
3차년도	◦회로기판의 소형화 설계	100	184*200mm 보드 개발 최초 Spec 보다 30% 최적화
	◦시제품 제작 및 기능시험, 성능 분석	100	PowerPC 603 기반 보드 개발, FPGA 성능 및 기타 점검 기능을 통한 기능 시험 우리별 1호와 과학기술위성1호의 SEU 확률을 이용한 보드의 성능 분석
최종평가	◦ FPGA를 이용한 주요 로직 설계	100	주요 로직의 FPGA code 구현
	◦OBC 시제품 개발	100	소형화 경량화된 603보드 SEU에 따른 분석 Two CPU에 따른 분석

2 절 관련 분야에의 기여도

1 관련 분야 기술 발전에의 기여도

본 과제를 통하여 기술적 측면에서 우주용 회로 설계 기술 확보 및 향상, 우주환경이 전자 회로에 미치는 영향에 대한 이해 증가, 우주용 FPGA 회로 설계 기술 확

보, 우주용 고 신뢰도 시스템 기술 확보, 우주용 시스템 경량화/소형화 기술 습득, 내방사선 회로 및 시스템 설계 기술 습득 등의 기술을 얻었으며, 이는 우주용 제어 보드의 설계에 크게 기여하였다. 특히 본 과제를 통하여 습득한 기술은 과학기술위성 2호의 OBC를 설계하는데 많은 부분이 사용되어, 과학기술위성 2호 OBC의 CPU가 PowerPC603으로 선정이 되고 이에 따른 관련 FPGA기술은 OBC 이외의 TCU(Telemetry and Command Unit)와 같은 기타 모듈들에서 채택하여 개발 하고 있다. 또한 우주 환경에 대한 조사를 통하여 얻은 자료와 우리별 1호와 과학기술위성 1호의 운용 데이터를 바탕으로 SEU의 발생에 대해 모델링하고, 이를 이용하여 분석을 하여 얻어진 결과를 이용하면 저가격의 위성 개발에 도움을 줄 것으로 기대된다.

2 연구 성과의 우수성

본 과제에서는 FPGA를 이용하여 OBC를 소형화, 경량화 하였으며, 그것은 초기 제안한 것에 비해 30% 이상 소형화 하였다. 이것은 과학기술위성 1호 대비 60% 이상 소형화 했다는 것을 의미한다. 또한 개발된 보드는 Intel i960을 이용한 과학기술위성 1호 대비 10배 이상 빠른 정보 처리 능력을 가질 수 있으며, 이는 PowerPC 603 CPU를 이용하여 가능하다. 그리고 보드의 개발에 있어서 우주 환경에 대한 조사와 규격에 대한 기술을 기반으로 보드를 제작하였다. 또한 SEU 분석을 통하여 저가격의 메모리 사용에 대한 가능성을 도출하였다.

3 연구 결과의 파급 효과

가 경제적 파급 효과

우주용으로 사용되는 대부분의 부품은 매우 고가이며, 또한 구하기도 어려운 경우가 많다. 그렇기 때문에 본 과제를 통한 얻은 기술들을 바탕으로 우주용 OBC나 기타 모듈을 개발할 경우, FPGA 내부 코딩을 통하여 개발을 하므로 추가 부품이 줄어들어 개발에 따른 부담을 줄일 수 있다. 뿐만 아니라, 또한 본 과제에서 개발한 보드에 비교적 싼 메모리를 사용하여 보드의 전체적인 가격을 낮출 수 있다. 이러한 기술과 개발된 보드를 이용하면 저 비용 고 효율의 기기를 설계하고 개발할 수

있으리라 예상 된다.

나 기술적 파급 효과

우주용 시스템의 설계 기술을 보유하고 있는 대부분 선진국들은 기술이전을 꺼리는 경우가 많기 때문에, 본 과제를 통한 우주용 OBC의 소형화/경량화 기술 확보를 통하여 신뢰도를 요구하는 여러 다른 분야에도 영향을 줄 수 있다. 예를 들어 아래와 같은 분야에 직·간접적으로 적용될 수 있을 것으로 예상된다.

- 항공기, 선박, 고속철도, 원자력 발전소, 대규모 공장의 주 제어 컴퓨터
- 자동차용 전자 기기 제어 시스템 (ABS, ECU 등)
- 은행 전산실, 교환기 등의 주 컴퓨터
- 지능형 빌딩 제어 시스템
- 각종 군용 장비(missile 탑재 컴퓨터, 방공 시스템등)
- 기타 대형 시스템 제어 컴퓨터

또한 우주용 회로 설계 기술, 우주 환경에 대한 영향 분석, 우주용 FPGA 회로 설계 기술, 우주용 고 신뢰도 시스템 기술, 우주용 시스템 경량화/소형화 기술, 내방사선 회로 및 시스템 설계 기술 등의 기술들의 확보를 통해 저 비용, 고 효율의 우주용 기기들을 만들어 낼 수는 기술의 진전이 있을 것으로 예상된다. 또한 FPGA를 이용한 위성 OBC 설계는 기존 많은 부품의 기능을 FPGA Code로 구현할 수 있기 때문에 부품의 단종에 따른 부품 의존도를 최소화 할 수 있을 것으로 예상된다.

제 5 장. 연구 개발 결과의 활용 계획

1 절 추가 연구의 필요성

본 과제에서는 FPGA를 이용한 소형화 경량화된 OBC 보드의 개발에 목적을 가지고 보드를 개발하였다. 이것은 H/W를 개발하는 것이었으며, 본 과제에서 개발한 보드를 좀 더 효율적으로 쓰기 위해서는 OS 등의 S/W 환경 구축이 필요하다.

2 절 타 연구에의 응용 및 기업화 추진 방안

1 타 연구에의 활용 방안

본 과제에서 조사, 개발 , 분석한 내용은 타 연구에 활용이 가능하다. 우주 환경에 대한 연구 결과는 우주용 부품 개발이나, 모듈 개발, 위성 개발 등의 자료로 이용이 가능하다. 또한 개발한 보드는 우주용 OBC로도 활용이 가능하며, 기타 산업용 기기 등에 활용이 가능하다. 개발시 축척된 FPGA 코드 기술을 이용하면, PowerPC 603e를 사용하는 많은 시스템에 이용 가능하며, 특히 과학기술위성 2호의 경우 본 연구의 많은 기술들이 활용 가능하다.

2 기업화 추진 방안

본 과제에서 개발한 시스템 또는 특정 장치에 대해 산업화를 원하는 기업이 있을 경우, 핵심 우주 기술 사업 처리 규정에 따라 추진할 예정이다. 우주용 소형화 OBC 보드에 대한 기업화 가능한 세부 분야로는 우주용 OBC 보드 개발이나, 고속 탑재체 개발 분야나, 항공기, 선박, 고속철도, 원자력 발전소, 대규모 공장의 주 제어 컴퓨터 , 동차용 전자 기기 제어 시스템 (ABS, ECU 등), 은행 전산실, 교환기 등의 주 컴퓨터, 지능형 빌딩 제어 시스템, 각종 군용 장비(missile 탑재 컴퓨터, 방공 시스템 등), 타 대형 시스템 제어 컴퓨터 등 다양한 여러 분야에서 기업화가 가능하다고 예상 된다.