μPSD

최종 출력일 : 2004년 3월 22일

File : PAT_APP_D105_uPSD정리.hwp

문서번호 : PAT-APP-D105			
번 호	갱 신	내 용	담 당
V1.0	2003.01.19	PSD 관련 사용법 활용 세미나를 위해 작성	Conan
V1.1	2003.07.15	ST사에서 제공된 μPSD 관련 내용 추가	Conan
V1.2	2003.08.22	YHJ G-LCD 소스추가	YHJ
V1.3	2003.09.04	내용, 회로도, 코드추가_관련 엔지니어(ST사 포함) 모니터링	All
V1.4	2003.09.14	(주) 첨단, 전자기술 2003년 11월~2004년 3월 시리즈 연재 제목 : STMicroelectronics사의 μPSD Quick Start	Conan
V1.5	2004.03.10	Conan's Firmware Community 자료용 수정	Conan

읽어두기

1. 본 문서는 STMicroelectronics사에서 출시되는 8032 Base 코어와 PLD 그리고 내장 메모리가 들어 있는 μPSD를 효율적으로 사용하기 위해 정리된 문서입니다.
2. 마이크로프로세서를 익히기에는 많은 시간과 꼼꼼한 점검과 시행착오가 필요하며 날로 복잡해지는 코드를 익히기에 무척 힘이 들곤 합니다. 본 원고는 익히고 숙달하는데 도움이 되고자 했으며 오류를 되풀이 하는 것을 방지하기 위해 작성되었습니다.
3. 저자는 처음 μPSD를 익힐 때 Data Sheet를 주로 참고하고 ST사에서 제공해 주는 Training Board를 이용했습니다.
4. 원고 원본은 HWP2002 로 작성되었습니다.
5. 본 원고를 작성하기 위해 사용된 컴파일러는 KEIL 7.05버전이 사용되었으며 μPSD 관련 프로그램은 PSD Soft Express입니다.
6. 내용에 대한 문의나 의문 사항 그리고 원고에서 저의 부족한 점이 발견 되었다면 동호회의 Q&A를 이용하길 바랍니다. Community : www.icbank.com/community/conan
7. 이 작은 Document를 이용하여 보다 나은 엔지니어가 되길 기원합니다.
8. μPSD에 대해 자료 만들기와 Study를 결성 하시고 싶은 분은 메일을 보내 주시기를 바랍니다.
9. 본 기술노트는 다음 회사의 도움과 지원으로 작성 되었습니다. 거론된 회사는 Distribute/한국본사이므로 Application, 자료, μPSD, PSD Chip의 구입과 Sample 그리고 기술지원을 받을 수 있습니다.

μPSD

차례

1. 개요

PSD(Programmable System Device)는 마이크로프로세서와 인터페이스를 보다 쉽게 하기 위해 CPLD는 물론 플래시메모리, RAM, EEPROM I/O 등의 자원까지 한 개의 웨이퍼에 집적되어 있는 고기능 칩이다. 무엇보다 손쉽게 소프트웨어로만으로도 메모리 Map을 포함한 PLD, I/O 등의 시스템 디자인이 가능하며 디자인된 하드웨어 정보와 운영프로그램 및 데이터를 JTAG 케이블을 이용하여 다운로딩이 가능하다.

PSD는 WSI(Wafer Scale Integration Inc : www.waferscale.com)사가 처음에 생산했으나 200년 이후에는 ST(www.st.com)사에서 인수 생산되고 있다. 현재 양산중인 제품의 종류로는 OTP의 PSD3XX, Flash의 PSD8XX/PSD42XX, DSP의 DSM(DSP System Memory) 그리고 이후로 설명하고자 하는 μPSD(Micro Programmable System Device) 제품군이 있다.

본 설명서는 위에서의 모든 제품(PSD와 μPSD)을 다루기에는 부족한 것이 많아 Micro PSD에 대해서 언급하고 Application을 설명하고자 했다. 특히 개발기간과 관리의 효율을 혁신적으로 할 수 있는 8051과 메모리 그리고 PLD가 하나의 칩에 구현된 μPSD를 잘 익히는 것이 날로 발전하는 전자공학의 세계에서 보다 유익하다고 판단되어 그에 대한 설계 방법과 Coding 그리고 틈틈이 적은 실험 노트를 정리했다.

μPSD는 칩 하나에 8032 Corer와 두개의 독립된 Flash 메모리 SRAM, Logic, PWM, ADC, I^2C, PLD, DDC, I/O, ISP, IAP가 들어 있어 PLD(Program Logic Device)하드웨어 구현을 프로그램으로 할 수 있으며 전원과 클럭만 연결하면 기본적인 사용이 가능하다. 설계와 동작은 μPSD안에 8032 코어로 되어 있기 때문에 운영 코드와 하드웨어 정보를 칩에 이식해야만 가능하다. 운영 프로그램은 8051 전용 컴파일러나 어셈블리를 이용하여 HEX 코드를 생성한 뒤 PSD의 하드웨어 구성 정보와 함께 JTAG을 통해 칩에 로딩하는 구조로 되어 있다. 그래서 컴파일러 외에 μPSD/PSD 칩과 PC가 인터페이스 할 수 있는 프로그램인 'PSDsoft Express'가 있어야 한다. 이 프로그램은 다음의 그림처럼 되어 있으며 다운은 'www.st.com/micropsd' 웹 사이트에서 무료로 받아 설치 할 수 있다. 2003년 5월에 나온 V7.91은 원고를 작성 할 때의 가장 최근 버전이지만 현재까지 Update가 되고 있으므로 웹 방문이 필요하다. 하지만 ST 사이트에서 C 컴파일러는 제공되지 않는다.

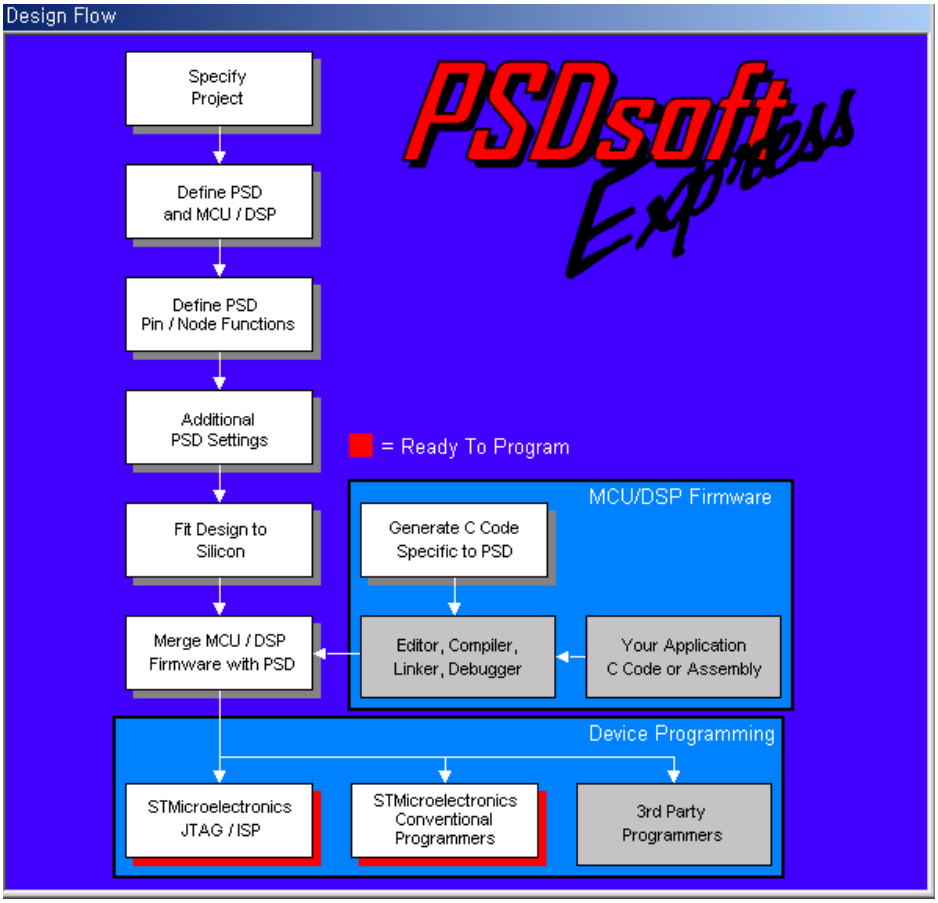

그림 1. µPSD/PSD 개발 소프트웨어 환경

µPSD의 강점은 8051 코어와 메모리가 붙어 있는데 그치지 않고 다품종 소량 생산 같은 자주 업그레이드되고 하드웨어가 수시로 변경되는 제품의 개발에 알맞도록 되어있다. 또한 하드웨어 소프트웨어 변경을 한 뒤 손쉽고 빠르게 그리고 동시에 로딩이 가능하고 바로 사용할 수 있다는 것이 특징이다. 그리고 USB, A/D 변환기, PWM, I2C, DDC, 등의 다양한 IP(Integrated Property)들을 제공하여 Firmware 엔지니어가 적은 노력으로 다양한 설계를 할 수 있도록 해준다.

알나간(알고나면간단)　　JTAG(Joint Test Access Group) ISP(In-System Programming)

ISP와 JTAG를 합한 것으로 Code 프로그램과 µPSD의 로직에 해당하는 DPLD, CPLD의 하드웨어 코드를 10~25초에 다운로드 가능하게 해주며 동작은 ISP Loader 블록을 사용하여 MCU와 별도로 행해진다. 개발할 때 포팅이 자유로워 개발 기간을 단축 할 수 있고 대량 이식이 가능하여 생산성과 개발 단가 면에서 효과적이다.

µPSD는 고용량과 고속 동작 그리고 저 소비전력의 방향으로 진화하고 있으며 ST사에서는 다음과 같은 양산 일정을 제시했다. 다음 3가지 시리즈의 요약을 표로 제공한다.

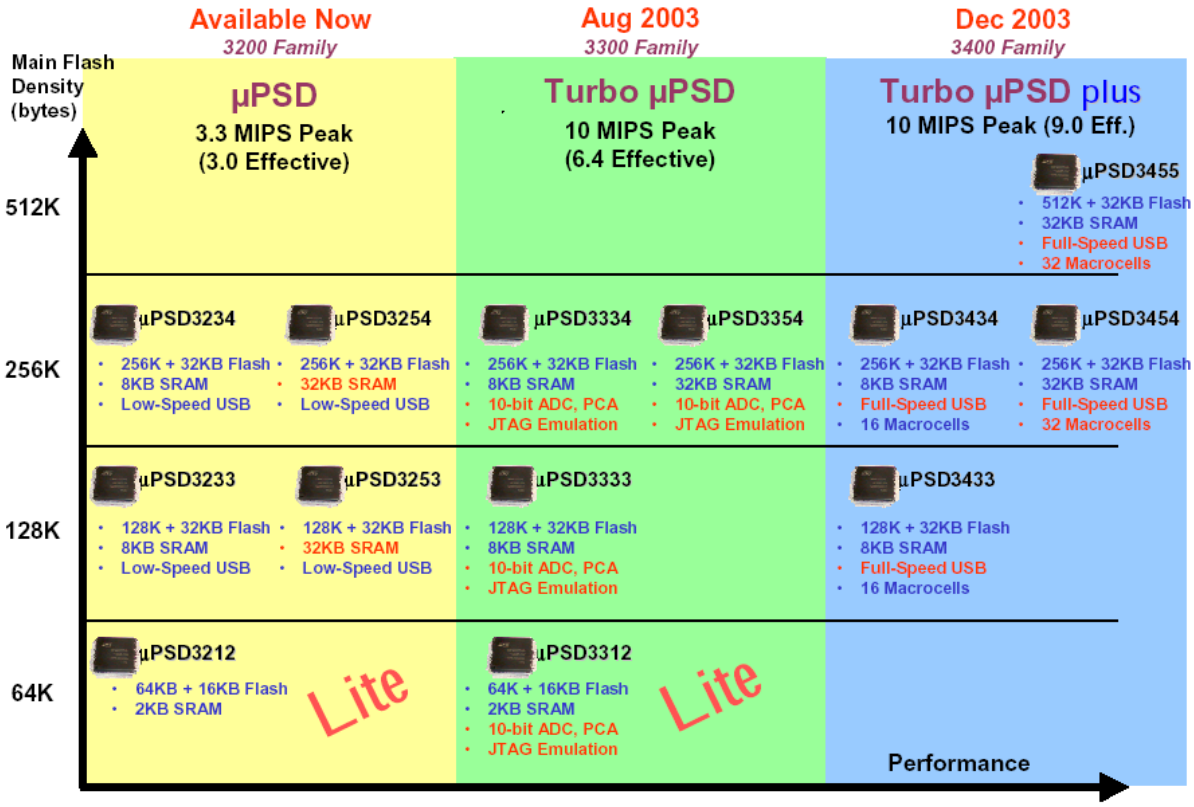

그림 2. µPSD Road map 자료출처:µPSD_Mar2K3.pdf

	µPSD Part Number	Peak Perf @Clock MHz	Main Flash bytes	2nd Flash bytes	SRAM bytes	PLD Macrocells	USB 1.1 Speed	ADC Channels	Timer/Counters	PWM Channels	UART Channels	I2C Channel	SPI Channel	IrDA Protocol	Wdog/Vcc Reset	JTAG Emulation	Display Data Chn	Max GPIO	8032 Bus Access	Operating Voltage	Temp -40 to +85C	Package
Lite Series	µPSD3212CV-24T6	2 MIP/24	64K	16K	2K	16	N	(4)8bit	3	5	2	1	N	N	Y	N	N	37	N	3.0-3.6	Ind	52-TQFP
	µPSD3212C-40T6	3.3 MIP/40	64K	16K	2K	16	N	(4)8bit	3	5	2	1	N	N	Y	N	N	37	N	4.5-5.5	Ind	52-TQFP
	µPSD3212CV-24U6	2 MIP/24	64K	16K	2K	16	N	(4)8bit	3	5	2	1	N	N	Y	N	N	46	Y	3.0-3.6	Ind	80-TQFP
	µPSD3212C-40U6	3.3 MIP/40	64K	16K	2K	16	N	(4)8bit	3	5	2	1	N	N	Y	N	N	46	Y	4.5-5.5	Ind	80-TQFP
Standard Series	µPSD3233BV-24T6	2 MIP/24	128K	32K	8K	16	N	(4)8bit	3	5	2	1	N	N	Y	N	Y	37	N	3.0-3.6	Ind	52-TQFP
	µPSD3233B-40T6	3.3 MIP/40	128K	32K	8K	16	N	(4)8bit	3	5	2	1	N	N	Y	N	Y	37	N	4.5-5.5	Ind	52-TQFP
	µPSD3233BV-24U6	2 MIP/24	128K	32K	8K	16	N	(4)8bit	3	5	2	1	N	N	Y	N	Y	46	Y	3.0-3.6	Ind	80-TQFP
	µPSD3233B-40U6	3.3 MIP/40	128K	32K	8K	16	N	(4)8bit	3	5	2	1	N	N	Y	N	Y	46	Y	4.5-5.5	Ind	80-TQFP
	µPSD3234BV-24U6	2 MIP/24	256K	32K	8K	16	N	(4)8bit	3	5	2	1	N	N	Y	N	Y	46	Y	3.0-3.6	Ind	80-TQFP
	µPSD3234A-40T6	3.3 MIP/40	256K	32K	8K	16	Low	(4)8bit	3	5	2	1	N	N	Y	N	Y	37	N	4.5-5.5	Ind	52-TQFP
	µPSD3234A-40U6	3.3 MIP/40	256K	32K	8K	16	Low	(4)8bit	3	5	2	1	N	N	Y	N	Y	46	Y	4.5-5.5	Ind	80-TQFP
Large SRAM Series	µPSD3253BV-24T6	2 MIP/24	128K	32K	32K	16	N	(4)8bit	3	5	2	1	N	N	Y	N	Y	37	N	3.0-3.6	Ind	52-TQFP
	µPSD3253B-40T6	3.3 MIP/40	128K	32K	32K	16	N	(4)8bit	3	5	2	1	N	N	Y	N	Y	37	N	4.5-5.5	Ind	52-TQFP
	µPSD3254BV-24U6	2 MIP/24	256K	32K	32K	16	N	(4)8bit	3	5	2	1	N	N	Y	N	Y	46	Y	3.0-3.6	Ind	80-TQFP
	µPSD3254A-40T6	3.3 MIP/40	256K	32K	32K	16	Low	(4)8bit	3	5	2	1	N	N	Y	N	Y	37	N	4.5-5.5	Ind	52-TQFP
	µPSD3254A-40U6	3.3 MIP/40	256K	32K	32K	16	Low	(4)8bit	3	5	2	1	N	N	Y	N	Y	46	Y	4.5-5.5	Ind	80-TQFP

그림 3. µPSD3200 시리즈

Turbo Series Aug 2003

µPSD Part Number	Peak Perf @ Clock MHz	Main Flash bytes	2nd Flash bytes	SRAM bytes	PLD Macrocells	USB 1.1 Speed	ADC Channels	Timer/Counters	PWM Channels	UART Channels	I2C Channel	SPI Channel	IrDA Protocol	Wdog/Vcc Reset	JTAG Emulation	Display Data Chn	Max GPIO	8032 Bus Access	Operating Voltage	Temp -40 to +85C	Package
µPSD3312DV-40T6	8 MIP/40	64K	16K	2K	16	N	(8)10bit	9	6	2	1	1	1	Y	Y	N	36	N	3.0-3.6	Ind	52-TQFP
µPSD3312D-40T6	10 MIP/40	64K	16K	2K	16	N	(8)10bit	9	6	2	1	1	1	Y	Y	N	36	N	4.5-5.5	Ind	52-TQFP
µPSD3333DV-40T6	8 MIP/40	128K	32K	8K	16	N	(8)10bit	9	6	2	1	1	1	Y	Y	N	36	N	3.0-3.6	Ind	52-TQFP
µPSD3333D-40T6	10 MIP/40	128K	32K	8K	16	N	(8)10bit	9	6	2	1	1	1	Y	Y	N	36	N	4.5-5.5	Ind	52-TQFP
µPSD3334DV-40U6	8 MIP/40	256K	32K	8K	16	N	(8)10bit	9	6	2	1	1	1	Y	Y	N	45	Y	3.0-3.6	Ind	80-TQFP
µPSD3334D-40U6	10 MIP/40	256K	32K	8K	16	N	(8)10bit	9	6	2	1	1	1	Y	Y	N	36	N	4.5-5.5	Ind	80-TQFP
µPSD3354DV-40T6	8 MIP/40	256K	32K	32K	16	N	(8)10bit	9	6	2	1	1	1	Y	Y	N	36	N	3.0-3.6	Ind	52-TQFP
µPSD3354D-40T6	8 MIP/40	256K	32K	32K	16	N	(8)10bit	9	6	2	1	1	1	Y	Y	N	36	N	4.5-5.5	Ind	52-TQFP
µPSD3354DV-40U6	8 MIP/40	256K	32K	32K	16	N	(8)10bit	9	6	2	1	1	1	Y	Y	N	45	Y	3.0-3.6	Ind	80-TQFP
µPSD3354D-40U6	8 MIP/40	256K	32K	32K	16	N	(8)10bit	9	6	2	1	1	1	Y	Y	N	45	Y	4.5-5.5	Ind	80-TQFP

그림 4. µPSD3300 Turbo 시리즈

Turbo 시리즈에 대해 간단히 요약하면 다음과 같다.

- 8051과 호환되는 명령어로 그것보다 3배(8-10MIPS)정도 빠름. 4개의 클락에서 한 개의 명령처리
- SPI와 IrDA 기능 추가 되었다.
- 다용도의 9개의 PCA(programmable Counter Array)
- 8개의 10비트 ADC
- Slow Clock Mode(Low Power 기능)
- 5V Tolerant I/O
- Dual data Pointer
- 3200시리즈와의 핀 호환성

Turbo plus Series Dec 2003

µPSD Part Number	Peak Perf @ Clock MHz	Main Flash bytes	2nd Flash bytes	SRAM bytes	PLD Macrocells	USB 1.1 Speed	ADC Channels	Timer/Counters	PWM Channels	UART Channels	I2C Channel	SPI Channel	IrDA Protocol	Wdog/Vcc Reset	JTAG Emulation	Display Data Chn	Max GPIO	8032 Bus Access	Operating Voltage (* = 5V Tolerant I/O)	Temp -40 to +85C	Package
µPSD3433EV-40U6	10 MIP/40	128K	32K	8K	16	Full	(8)10bit	9	6	2	1	1	1	Y	Y	N	62	N	3.0-3.6*	Ind	80-TQFP
µPSD3434EV-40U6	10 MIP/40	256K	32K	8K	16	Full	(8)10bit	9	6	2	1	1	1	Y	Y	N	62	N	3.0-3.6*	Ind	80-TQFP
µPSD3434FV-40H6	10 MIP/40	256K	32K	8K	32	Full	(8)10bit	9	6	2	1	1	1	Y	Y	N	62	Y	3.0-3.6*	Ind	100-TQFP
µPSD3454FV-40U6	10 MIP/40	256K	32K	32K	32	Full	(8)10bit	9	6	2	1	1	1	Y	Y	N	62	N	3.0-3.6*	Ind	80-TQFP
µPSD3455FV-40H6	10 MIP/40	512K	32K	32K	32	Full	(8)10bit	9	6	2	1	1	1	Y	Y	N	62	Y	3.0-3.6*	Ind	100-TQFP

그림 5. µPSD3400 Turbo plus 시리즈

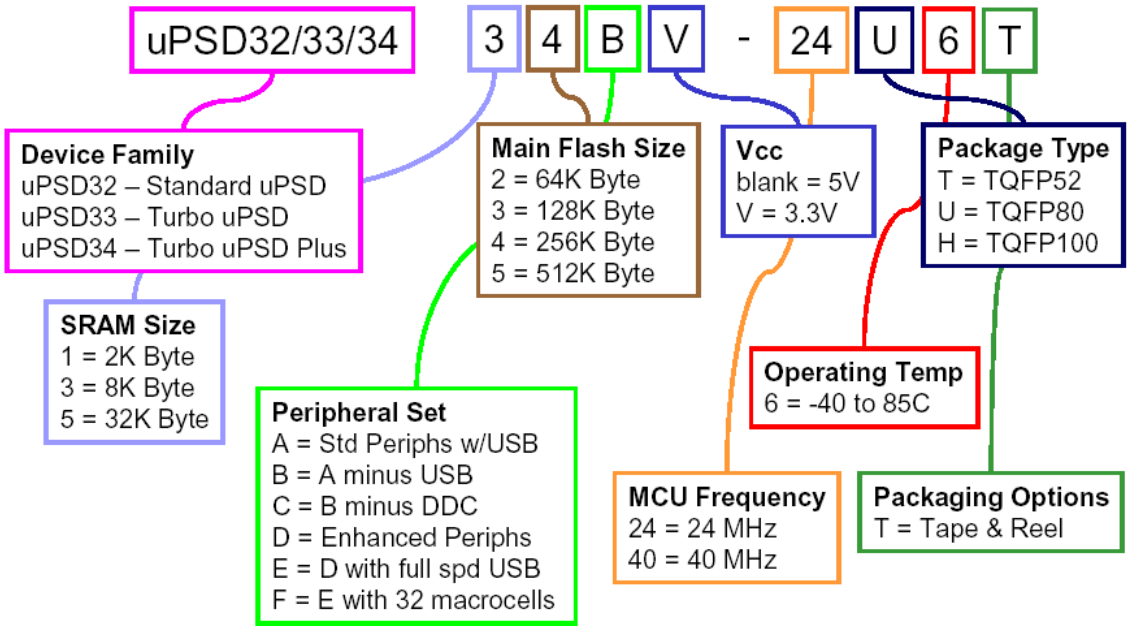

그림 6. Part 번호 구분 방법

여기까지 μPSD의 특징과 종류 형명을 보는 방법 그리고 앞으로 기대 되는 제품에 대해 살펴보았다. 능숙하게 잘 사용하기 위해서는 손에 익을 정도로 System을 구성해 만들어 보고 오류에 대해서는 Debug를 해야 하는데 무엇이든 절차가 있는 법, 우선 넓게 학습해 보자.

2. PSDsoft Quick Start

어느 정도 PSD를 다루어 본 사용자 엔지니어라면 집적 프로젝트를 생성해서 다음을 진행해도 무리가 없겠지만 그렇지 않고 초보자라면 한번 정도 매뉴얼을 보고 따라해 보면서 순서와 원리를 익히도록 하는 것이 바람직하다. 저자인 경우에도 이 문서를 작성할 때는 처음으로 PSD를 접한 시점이기 때문에 지식을 새로이 터득하는 습득의 개념에서 원고를 작성했다.

μPSD는 소프트웨어적으로는 두개의 프로그램을 운영할줄 알아야 하며 특히 8032 코어에 대한 어느 정도의 기본 지식과 컴파일러를 사용할 경우 C 언어에 대해서 별도의 설명 없이도 이미 알고 있어야 하는 기본 기술을 필요로 하는 장르이다. 다시 말해 마이크로프로세서와 개발 환경에 어느 정도 기본이 되어야 하므로 빠른 진도와 효율을 높이기 위해서는 초기 기본 학습이 필요하다고 하겠다. 두개의 PC 운영 프로그램은 ST사에서 제공되는 'PSDsoft Express'와 독일 KEIL사의 컴파일러 개발 환경에 관한 것이다. 하나씩 실행을 하면서 익히면 μPSD 디자인 환경과 Firmware 개발은 어느 정도 익힐 수 있으리라 본다. 근래에 들어와서는 기술의 발달로 인해 지식을 접할 때 예전보다 많은 학습량과 집중을 요구하므로 인내가 그 어느 때보다도 더욱 절실하다고 하겠다.

본 챕터는 μPSD에 대해 개념과 적응을 하기 위해 전체를 실행해보는 단계로 PSD웹 사이트(http://www.st.com/psd/)에서 제공되는 'PSDsoft Express User Manual'을 참고했다. 운영 프로그램은 V7.80과 V7.91을 혼용하여 사용했으며 http://www.st.com/stonline/products/families/memories/psm/so

ft_c2.htm의 웹 사이트에서 구해서 설치했다. 간단한 시트만 작성하면 'PSDsoft Express' 무료로 다운 받을 수 있으며 사이즈는 약 9M 정도 된다. 'PSDsoft express' 소프트웨어는 PLD 디자인을 위한 HDL의 코드를 생성하고 8032 실행 코드를 합해 다운로드 할 때 사용하며 ST사의 PSD(8032 코어가 빠진 것)와 μPSD는 모두 이 프로그램 하나로 사용할 수 있다. 물리적으로 프린터 포트에 연결하는 JTAG 케이블이 한개 있어야 하는데 구입 형편이 여의치 않을 경우는 JTAG 케이블을 자작해야 하며 회로도는 후반부에 소개했다.

설명되는 원서의 기본은 웹 사이트 혹은 설치 CD에서 제공되는 Application 'AN1560'을 토대로 작성 되었으며 원서에서 몇 가지 오류와 이해하기 힘든 부분은 저자가 임의대로 각색해서 작성했고 프로그램 소스는 웹에서 다운 받은 'dk32dsn1.zip'을 기본으로 진행했다.

프로그램을 다 설치하면 윈도우의 '시작/STMicro Electronics-PSDsoft Express'에 프로그램이 설치되어 있을 것이다. 이곳에서 'PSDsoft Express'를 실행하면 된다. 개발 환경은 프로젝트로 관리되므로 새로운 프로젝트를 만들든지 아니면 기존의 프로젝트를 불러 오든지 하면 된다. 새로운 프로젝트는 'Project/Create.../ New project'에서 경로를 브라우저로 선택한다.

이미 한번 이라도 실행되었다면 프로젝트 환경을 몽땅 물러와서 아마 다음과 같은 화면이 나올지도 모른다. 프로젝트 개개인의 항목은 그림에 있는 네모박스를 직접 클릭해도 되고 풀다운 메뉴의 항목을 선택해도 되며 적색으로 된 박스 부분은 다음 실행하는 항목을 지시하는 것으로 GUI를 가능한 개발자가 차례대로 하여 혼동을 방지 하도록 배려한 것이 보인다. 이제 이것을 하나씩 익혀보자.

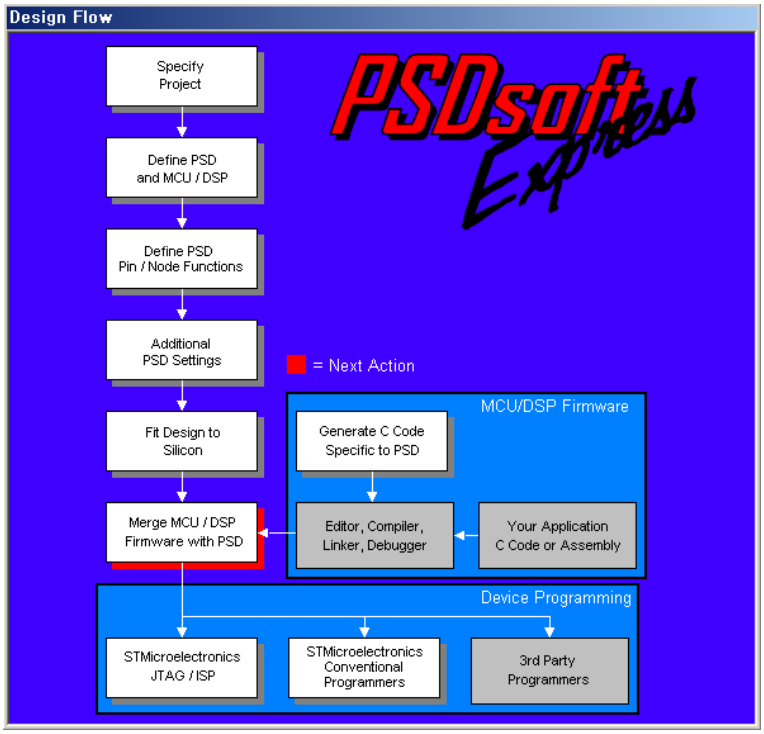

그림 7. PSDsoft를 이용한 개발 절차

일단 프로젝트를 만들어 보자. 여기서는 'PSD3234A.ini'라고 했는데 자동으로 ini 확장자가 프로젝트 이름에 추가된다. 경로는 나중에 만들게 될 8032 운영 프로그램 Hex 코드와 혼용이 되지 않도록 별개의 디렉터리에 저장하는 것이 바람직하고 확장자가 'ini'가 된다는 것에 유념할 필요가 있다. 프로젝트를 불렀을 때 자신이 문장을 넣을 수 있으므로 십분 활용하도록 하자. 한글 OS를 사용했을 경우 한글 입력도 가능하다.

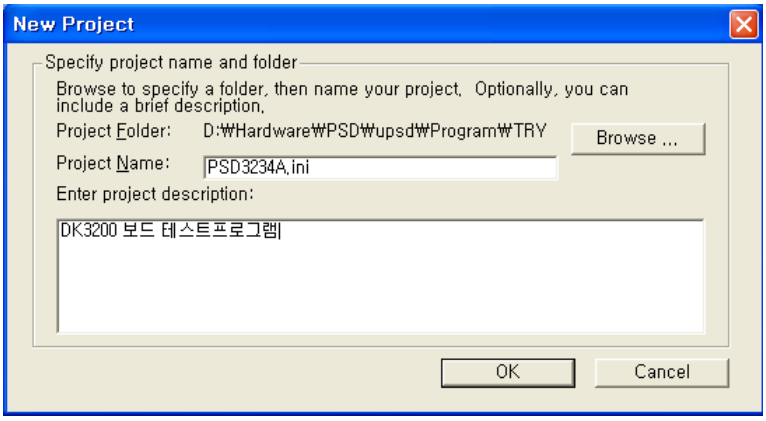

그림 8. Project 생성

다음은 MCU의 제조회사와 세부 칩 번호 그리고 메모리의 구분에 대해 선택한다. Step1에서는 달리 혼돈스러운 것이 없겠지만 Step2에서는 칩의 형명과 패키지를 선택하는데 칩을 설정하는 경우의 수가 많다면 'Wizard'를 선택하여 도우미를 통해 선정이 가능하다. 그리고 Step3은 버스와 방식을 선택하는 것인데 특히 Main Flash와 Secondary Flash의 방식에 대해서는 초기 전원 인가 상태에서 각각의 영역이 Program 영역인지, Data 영역인지를 설정해 주기 위한 메뉴이다. 즉 동작 중에 Firmware가 CSIOP 영역의 레지스터를 통하여 제어 할 수도 있다. 예를 들어 IAP를 수행하기 위해서는 Update 하기 위한 Program 영역을 일시적으로 Data 영역으로 바꾸어야 할 때 매우 유용한 자원이 될 수 있다.

우리가 Firmware를 만들고 코딩하기 위한 칩은 5V(V는 3.3V)용의 μPSD3234A 80핀이다. 본격적인 개발에 착수하기 전에 꼼꼼히 데이터시트 'uPSD323X.pdf'를 꼭 탐독하길 바란다. 개발할 때 마다 느끼는 건데 역시 답은 데이터시트에 있다.

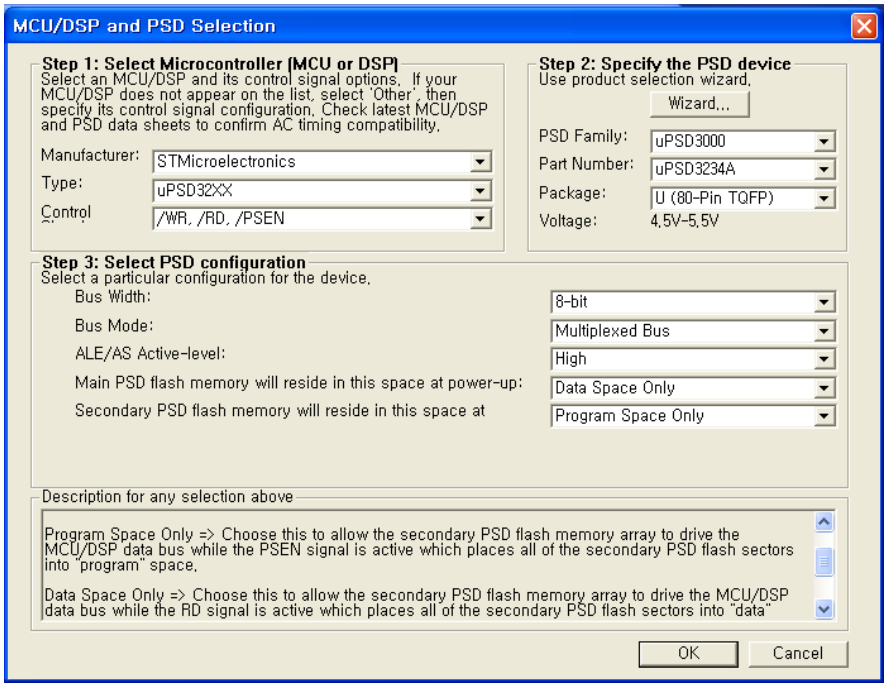

그림 9. μPSD 주요 항목 선택

'OK'를 선택하면 3가지 디자인 가이드가 나오는데 일반적인 디자인의 'Design Assistant', HDL 언어와 PLD 코드를 생성해 주는 'Extended Design Assistant', 그리고 샘플 프로젝트를 생성해주는 'Example Temple Selection' 등이 있다. 여기서는 두 번째를 선택 후에 진행했다.

여기까지 칩과 프로젝트 항목들을 정했다면 다음에는 핀에 대한 해당 기능과 이름을 부여하는 'Define PSD Pin/Node Function'이다. 이곳에서는 핀에 기능을 부여하므로 설계한대로 적용하도록 하면 되는데 무엇보다도 용어에 대한 정의를 바로 알아 두는 것이 관건이라 하겠다. 우선 회로도에서 보는 것처럼 16*2줄 Text LCD와 μPSD를 인터페이스 하기위해 설계했다.

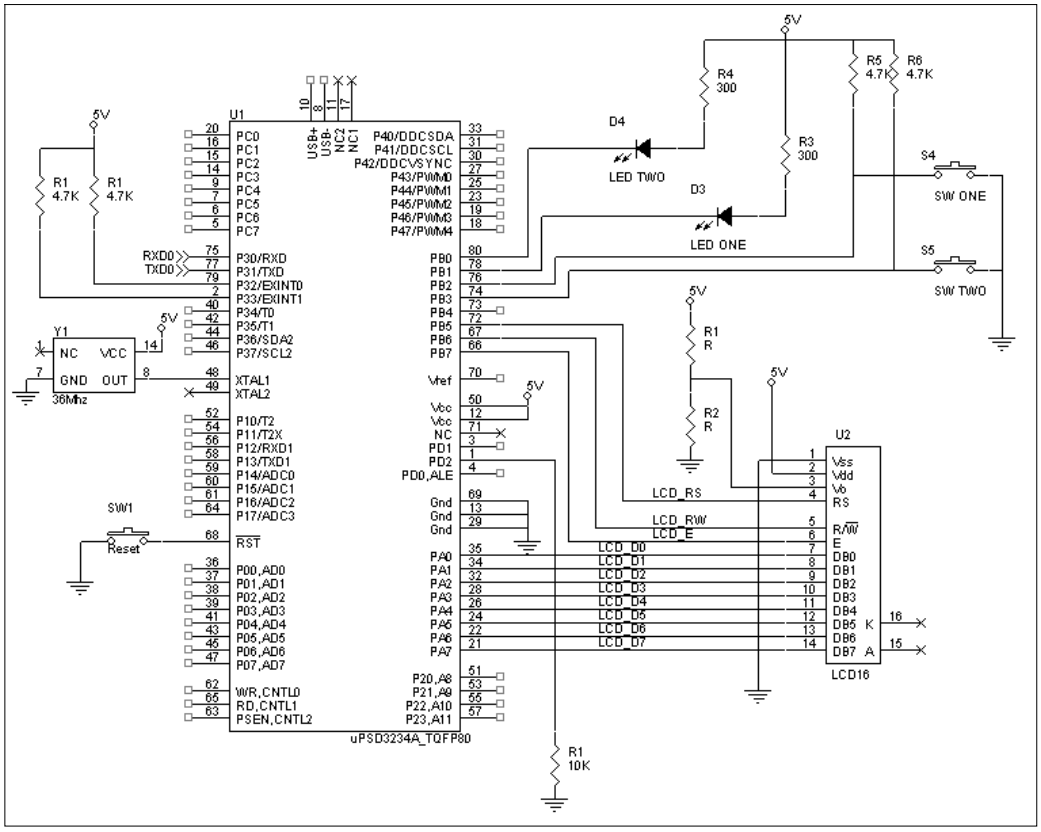

그림 10. μPSD와 LCD, LED 설계도

LCD_D0 ~ LCD_D7 포트는 PA 포트를 사용했으며 병렬(Peripheral) I/O로 설정했다. 여기서 Peripheral I/O란 그룹 Data 선을 말하는 것으로 User가 선택한 Data 영역에만 신호가 나오는 기능이다. 물론 그 이외의 영역에는 High Impendence 상태로 유지된다. PB7은 LCD_e 신호는 LCD의 Chip Select로 B포트를 사용하였고 Active Hi에서 동작된다. LCD_rw, LCD_rs 신호는Combinatorial 동그란 라디오 버튼을 클릭하여 후에 Address와 연결 되도록 했다. 여기서 해당 핀의 이름은 Express를 사용할 때 이외에는 별 효용가치가 없지만 이해하기 좋게 이름(예약어 조심)을 부여하고 관리하는 것이 바람직하다. 그리고 'View' 버튼을 누르면 중간보고서가 나오게 되는데 이때까지 설정한 상태의 결과라고 할 수 있다. 파일이름은 프로젝트이름.sum이다.

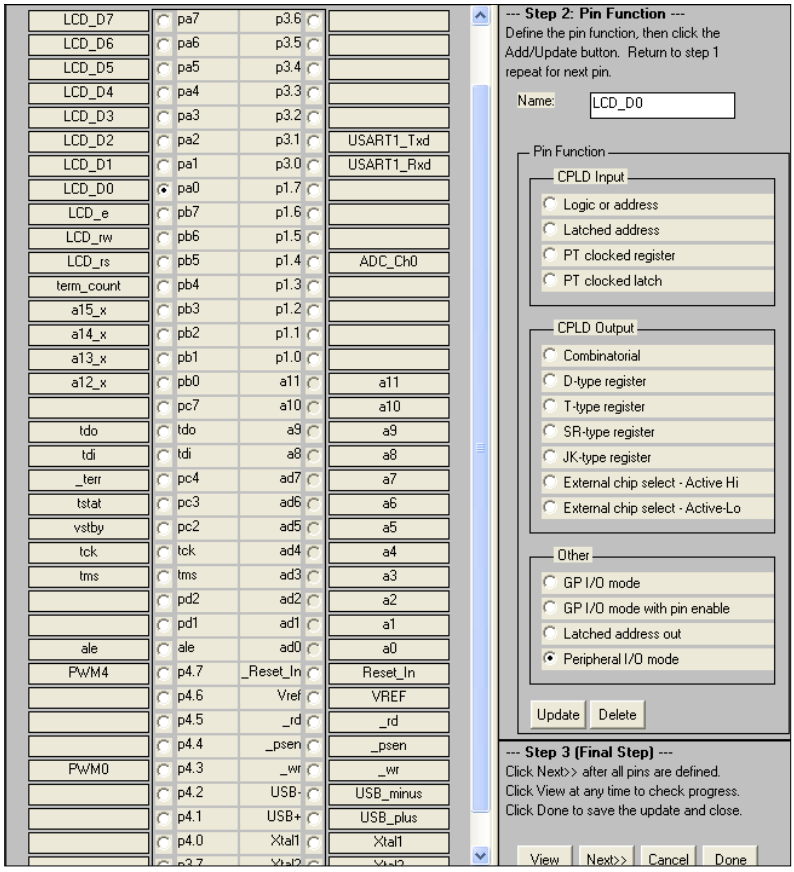

그림 11. Pin Definition

핀 설정 화면의 오른쪽 하단에 보면 'Next' 버튼이 있는데 누르면 Page Register와 PLD부분을 설정할 수 있는 항목으로 넘어간다. Page Register 부분은 64K 이상의 메모리를 사용하기 위해 'Bank Register'로 쓰기위한 용도와 일반적인 Logic Register의 용도로 나눌 수 있다. 로직 부분은 상위 포트 7번부터 시작하고 페이지는 당연히 낮은 포트 0번부터 시작한다. Paging 뱅크를 사용하면 최대 256개의 선택적으로 사용할 수 있는 메모리가 있는 것과 같다. 여기서는 프로그램이 크지 않아 뱅크를 추가 할 나눌 필요가 없으므로 아무것도 선택하지 않았다.

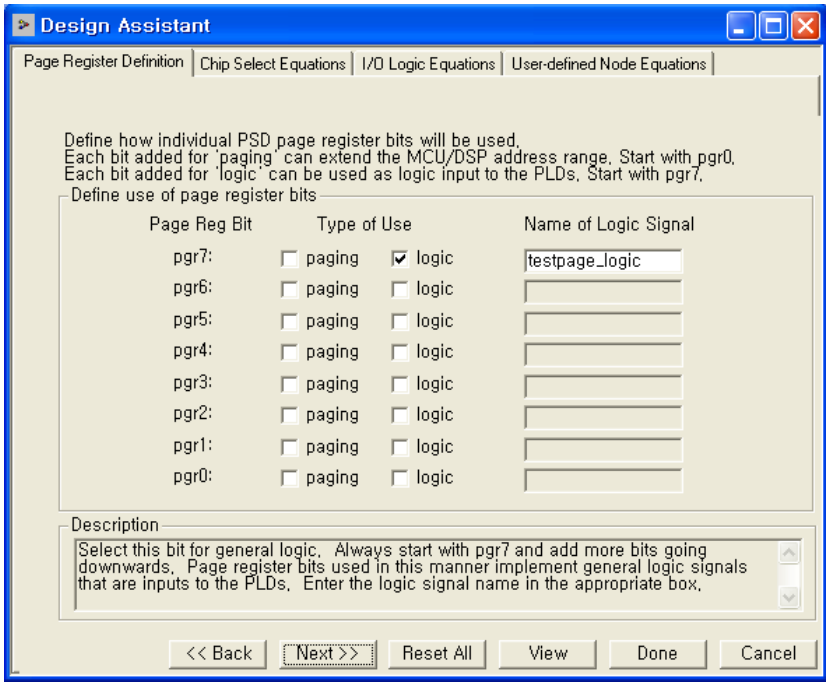

그림 12. Logic에 의한 Page 설정 예

다음은 메모리맵과 관계가 있는 'Chip Select Equation' 부분이다. 정리된 메모리 맵을 보고 방정식을 넣어 핀이 특정 어드레스가 될 때 선택되게 하는 것이다. 하드웨어적으로는 Decode에 해당 되는 부분이며 이것을 나중에 소스 프로그램에서 어드레스로 설정해준 다음 코딩해야 한다. 변경할 때는 다시 언제든지 가능하므로 관리가 편하다고 하겠다. 그러나 방정식에 있는 어드레스 번지는 메모리의 크기를 나타내며 또 C 언어로 코딩할 때 데이터와 메모리 영역의 중요한 값이므로 서로 다른 값으로 중복되지 않게 하자. 어드레스 영역은 사실 사용자가 설정해 주기 나름이지만(이 부분은 대단히 중요함) 하드웨어의 설정값과 컴파일러 혹은 어셈블리의 소스 코드에서도 동일하게 만들어주어야 한다. 당연한 얘기지만 Hardware 디코드 값과 소스에서의 설정값이 다르면 잘못된 부분을 알고 수정하기 가지의 시간이 예상외로 많이 걸린다.

어드레스 설정 예	기호	내 용	방정식
0000~00FF		SFR이 있는 위치	특별히 설정하지 않아도 된다. data 부분이 된다.
0000~1FFF	csboot0	Boot Flesh Memory	(address >= ^h0000) & (address <= ^h1FFFF), Code영역
0200~02FF	csiop	A, B, C, D 포트제어	(address >= ^h0200) & (address <= ^h02FF), µPSD Only
0300~03FF	LCD_e	LCD Enable 신호, 어드레스와 AND WR, RD로 OR 취함	(address >= ^h0300) & (address <= ^h03FF) & (!_rd))# ((address >= ^h0300) & (address <= ^h03FF) & (!_wr) 300~3FF 번지중의 Read, Write 신호가 High 일 때 신호발생
0400~1FFF		사용되지 않음	
2000~3FFF	rs0	SRAM이 있는 위치	(address >= ^h2000) & (address <= ^h3FFF)
4000~7FFF		사용되지 않음	
8000~FFFF	fs0~fs7	8개의 Page	(page == 0번부터7번까지) &(address >= ^h8000) & (address <= ^hFFFF)

표 2. XDATA, Code 영역의 Memory Map 설정 예

LCD의 RS, RW 신호는 각각 A1, A0 어드레스를 집적 연결해서 실제적으로 LCD의 선택에서는 0번부터 3번지까지의 경우로 LCD 제어를 했다. 8051 칩을 가지고 Firmware를 제작해 본 엔지니어라면 이 PLD가 얼마나 편한지 알 것이라 본다. 메모리맵 부분과 PLD 방정식을 다 기입 했다면 'Done' 버튼을 누르자. 뒤편의 알고리즘 코딩할 때 만족할 만한 결과가 나오지 않았거나 에러가 발생했을 경우는 이곳으로 와서 수정이 가능하다.

Memory Map의 PSDsoft에 의해 설정된 번지는 C 소스에서 다음과 같은 값을 사용하게 된다. 예를 들어 외부 RAM 어드레스에서 사용된 csiop 레지스터는 디코드에 의해 설정되므로 PSDsoft에서의 어드레스 번지수와 동일한 값을 사용해야 한다.

```
//=============================================================================
#define PSD_REG_ADDR    0x200    // μPSD Register I/O base address
typedef xdata struct REG_PSD_struct {
    unsigned char DATAIN_A;      // PSD_REG_BASE +0x00(Offer Set 값)
    unsigned char DATAIN_B;      // +0x01
    unsigned char CONTROL_A;     // +0x02
    unsigned char CONTROL_B;     // +0x03
    중략...
    unsigned char VM;            // +0xE2, 마지막 레지스터
} PSD_REGS;
PSD_REGS xdata PSD8xx_reg _at_ PSD_REG_ADDR;  // 200번지의 xdata 영역에 csiop 번지할당
//=============================================================================
```

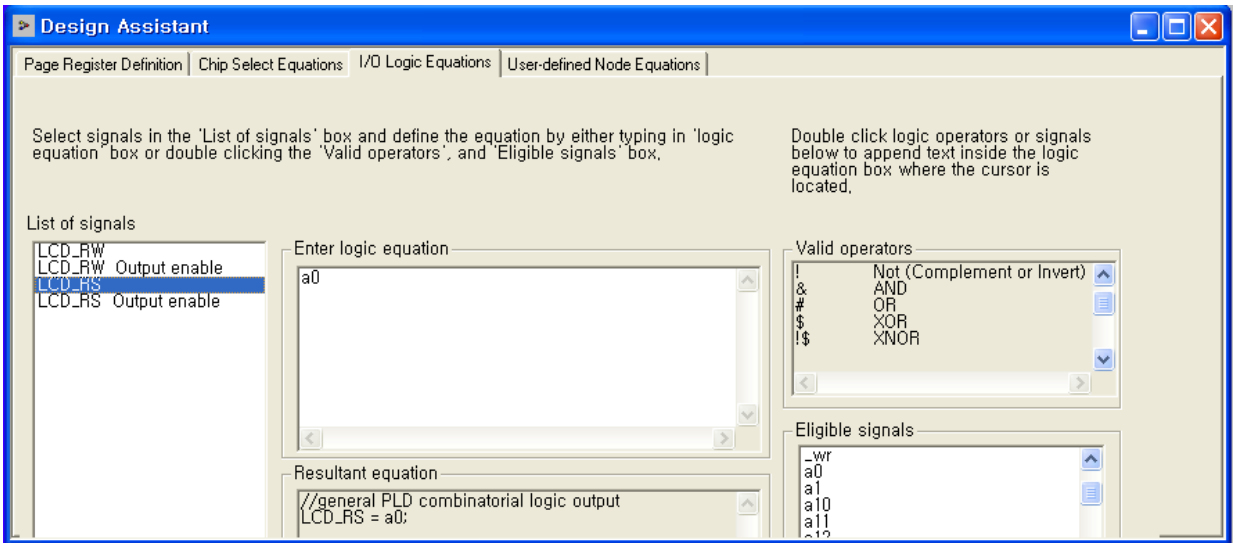

그림 13. LCD의 제어신호와 어드레스 연결

'View' 버튼을 누르면 그동안 설정되고 선택된 항목을 Text 파일로 보여 준다. 확장자는 *.sum 이며 다음은 설정된 값들과 Config의 일부를 나타낸다. 만약 초과 값이 사용되었다면 'Done' 버튼을 누른 뒤에 에러, 경고 메시지가 돌출한다.

```
PSDsoft Express Version 7.80 Summary of Design Assistant
************************************************************************
PROJECT    : PSD3234A              DEVICE     : uPSD3234A             MCU/DSP    : uPSD32XX
************************************************************************
Initial setting for Program and Data Space:
===========================================
 Main PSD flash memory will reside in this space at power-up:      Data Space Only
 Secondary PSD flash memory will reside in this space at power-up: Program Space Only

Pin Definitions:
================
Pin         Signal                  Pin Type
----------- ------------------------ ------------
pa7         LCD_D7                   Peripheral I/O mode
pa6         LCD_D6                   Peripheral I/O mode
중략...
pa0         LCD_D0                   Peripheral I/O mode
pb7         LCD_E                   External chip select - Active Hi
pb6         LCD_RW                   Combinatorial
pb5         LCD_RS                   Combinatorial
pb3         SW_TWO                    GP I/O mode
pb2         SW_ONE                   GP I/O mode
pb1         LED_ONE                   GP I/O mode
pb0         LED_TWO                   GP I/O mode
중략...

Page Register settings:
=======================
pgr0 is not used
중략...
pgr7 is not used

Equations:
==========
rs0 = ((address >= ^h2000) & (address <= ^h3FFF));   csiop = ((address >= ^h0200) & (address <= ^h02FF));
fs0 = ((address >= ^h8000) & (address <= ^hFFFF));   csboot0 = ((address >= ^h0000) & (address <= ^h1FFF));
LCD_E = ((address >= ^h0300) & (address <= ^h03FF) & (!_rd))  # ((address >= ^h0300) & (address <= ^h03FF) & (!_wr));
LCD_RW = a1;          LCD_RW.oe = Vcc;          LCD_RS = a0;      LCD_RS.oe = Vcc;
```

알나간 IAP(In-Application Programming)

마이크로프로세서를 운영할 때는 프로그램 코드와 데이터가 저장되거나 MPU에 의해 가공되어 처리되지만 외부의 다른 통신 채널에 의해서 응용 프로그램이 기록되기도 한다. 보통 CAN, UART, Ethernet, J1850, USB 통신 선로를 통하여 응용 프로그램을 로딩하거나 플래시 메모리 또는

Application 메모리에 기록 할 수 있는 구조로 되어 있다. 일반적인 Flash Memory로는 서로 다른 Sector에서라도 동일한 메모리 내에서는 Erase/Write 동작과 Read가 동시에 진행 될 수 없으므로 단점을 극복하기 위하여 별도의 독립된 Flash가 있는 구조로 되어야 한다. 이렇게 하면 두개의 Flash를 모두 프로그램 영역으로 혹은 데이터 영역으로 나누어서도 운영이 가능하다. 이것은 하나의 메모리로부터 프로그램이 동작하는 동안 다른 하나의 메모리에의 삭제나 로딩이 가능하다. 단 모든 처리는 MPU를 통하여 이루어진다.

'Fit Design to Silicon' 항목을 실행하면 이 프로그램이 얼마나 사용자를 편하게 하려고 노력했는가를 알 수 있다. 마치 회로도처럼 각 핀에 대한 설정과 칩에 이미지를 올렸을 때의 정보를 보여 준다.

다음에 할일은 뭘까? 하드웨어정보와 실행 Application 프로그램을 μPSD에 올리는 것이 남아 있다. 칩의 디코드와 PLD/하드웨어 정보는 *.obj 이고 LCD에 어떤 문자를 디스플레이 할 것인가 등에 대한 응용 프로그램은 8032에서 운영되는 *.hex 코드 파일로 만들어 진다. 누군가에 의해 만들어져서 넘겨받은 다음 칩에 로딩 하는 것은 일도 아니지만 알고리듬을 집적 만들고 Firmware 수준에서 오류를 찾고 코딩 한다고 하면 세세한 부분까지 일일이 신경 써야 하는 인내의 기술이 필요하게 된다. 여기서 일차적으로 Firmware를 완성하기 위해 하드웨어 인터페이스 케이블을 프린터 포트와 연결하고 로딩 준비를 한다. 특히 JTAG 케이블을 Target과 연결할 때 올바로 체결하고 전원을 넣는다. 저자는 14핀 JTAG 핀을 뒤집어서 연결한 다음 전원을 넣은 때가 있었는데 케이블의 보호회로 때문인지 쉽사리 고장 나지는 않았다.

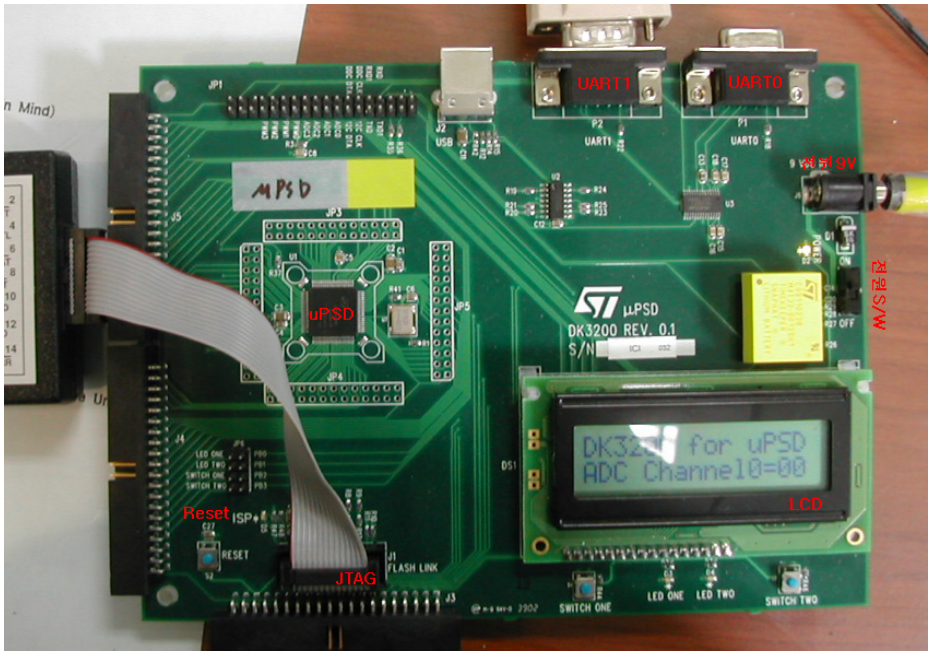

그림 15. JTAG과 보드 연결 모습

Firmware는 하드웨어와 응용프로그램으로 나눌 수 있는데 하드웨어는 'PSDsoft Express'에서 라디오

또는 사각 버튼을 클릭하고 이름을 설정하면 되지만 응용프로그램은 알고리즘을 부여하는 방법으로 기계어를 만들 수 있는 도구가 있어야 한다. 요즈음은 어셈블리보다는 복잡한 알고리즘을 구현하고 정확히 빨리 개발하기 위해서 C 언어로 작성된 프로그램을 기계어로 번역하는 Compiler를 선호한다. 어셈블리 언어로 작성할 정도의 기술을 가지고 있으면 상관없지만 복잡하고 어려운 코딩을 하기에는 C 언어가 적합하다고 본다. 컴파일러는 독일사의 KEIL 7.05 버전을 이용했다. 아마 7.0 버전 이상부터 μPSD와 디버그 인터페이스가 되므로 Full로 사용하기 위해서는 상위 버전의 프로그램이 있어야 하겠다. 컴파일러를 운영하여 프로그램을 만드는 것은 다음 장에서 하도록 하고 이곳에서는 이미 만들어진 코드를 로딩하고 실행하는 절차를 다루었다.

알나간　　JTAG(Joint Test Access Group)

70년대 중반에는 “bed-of-nails"라는 테크닉을 사용하여 직접 PCB(Printed Circuit Boards)에 접촉하는 방식으로 Board를 테스트하였으나 기술이 발달함에 따라 Board 단자 사이의 거리가 좁아지고 다층기판이 나타남에 따라 기존의 방법으로는 검사가 불가능하게 되었다. 문제점을 해결하기 위해 80년대 중반에 Joint European Test Access Group 이란 단체가 결성되었는데 그들은 a serial shift register around the boundary of the device라는 개념을 발전시켰다. 후에 이 단체는 North American company와 join하게 되었으며 이에 따라 E(European)문자가 없어지고 JTAG으로 남게 되었고 결국에는 IEEE에서 1990년에 표준화하여 IEEE std 1149.1로 제정되었다.

JTAG 또는 Boundary-Scan 이라고도 하는데 JTAG은 칩 내부에 Boundary Cell이란 것을 두어 외부의 핀과 일대 일로 연결시켜 프로세서가 할 수 있는 동작을 중간에 Cell을 통해 모든 동작을 인위적으로 수행할 수 있도록 하여 여러 가지 하드웨어 테스트나 연결 상태 등을 체크할 수 있다.

JTAG은 프로세서(CPU)의 상태와는 상관없이 디바이스의 모든 외부 핀을 구동시키거나 값을 읽어들일 수 있는 기능을 제공하므로 이를 이용해서 메모리나 I/O에 직접 명령을 보내어 디버그용으로도 사용된다. JTAG의 기능을 좀더 보면 다음과 같다.

① 디바이스 내에서 외부로 나가는 각각의 핀들과 일대 일로 연결 되어 연결점을 가로챌 수 있다.
② 각각의 셀은 시리얼 쉬프트 레지스터(바운더리 스캔 레지스터)를 형성하기 위해서 서로 연결되어 있다.
③ 전체적인 인터페이스는 TDI, TMS, TCK, nTRST, TDO 5개의 핀에 의해서 제어된다.
④ 회로의 배선과 소자의 전기적 연결 상태 test
⑤ 디바이스간의 연결 상태 test
⑥ Flash memory fusing

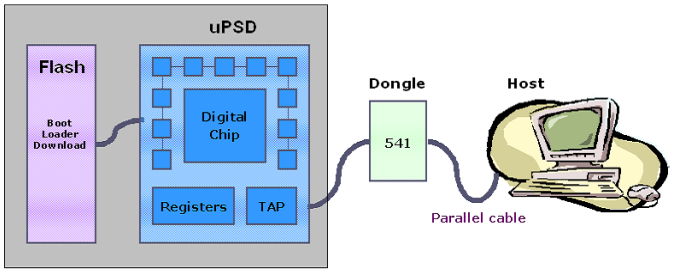

그림 16. JTAG, Chip, PC와의 인터페이스

하드웨어 Configuration에 의한 *.OBJ 코드와 알고리즘 Source가 컴파일 되어 기계어로 변환된 *.HEX 코드가 준비 되었다면 최종적으로 μPSD에 다운로드하기 위한 파일을 만들기 위해 'Merge MCU/DSP Firmware with PSD'를 실행 시킨다. 이때 메모리 섹트별로 Hex 파일을 입력하고 'OK'를 클릭한다. 그리고 최종적인 OBJ 파일이 생성되면 플래시메모리와 PLD, I/O Config 부분에 다운로딩 해야 하는데 이때의 버튼이 'ST Microelectronics JTAG/ISP'이다. 실행하면 복수 동작 정보를 묻는 항목이 나오게 된다. 원래 JTAG는 여러 개의 서로 다른 칩을 연결할 수 있는데 로딩이 한 개의 칩에만 연결되어 있는가 아니면 하나 이상이 연결되어 있는가를 물어보는 메시지이다. 보통은 한개 이므로 'Only'를 선택하자.

'OK'를 누르면 다음의 오른쪽 화면처럼 Step1~3번까지의 항목이 나오는데 스텝1은 하드웨어 정보 *.obj 파일의 위치와 PSD 종류, 형명에 대해 선택하는 것이다. 스텝2는 칩에 포팅 후 올바로 되었는지 점검하는 방법에 대해 선택하고 무엇보다 JTAG가 4핀인지 6핀이지를 명확히 지정해야 한다. 보통 자작 하면 4핀이지만(핀 수를 줄이기 위해) ST사에서 제공되는 'FlashLINK'를 사용하면(손바닥만한 장비에 나와 있는 핀 수는 14핀이다) 6핀을 사용한다. 나머지는 에러에 대한 신호와 리셋발생의 기능으로 이용된다. 스텝3은 이때까지의 과정을 파일로 저장하는 것으로 다음에 불러오면 설정을 한꺼번에 할 수 있다. 실행은 오른쪽 중간에 있는 'Execute'를 하면 μPSD의 내부를 지우는 확인 메시지가 뜬 다음 'Yes' 하면 실행된다. 실행된 뒤에는 소요된 포팅시간과 결과가 디스플레이 되고 하드웨어를 Reset 해준다.

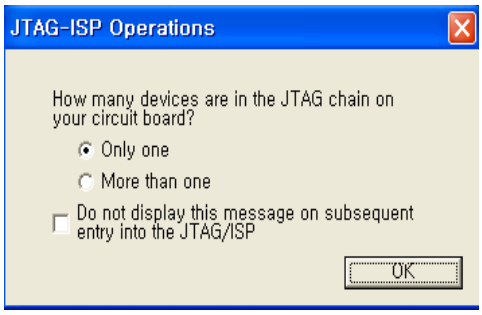

그림 17. 디바이스 수 선택

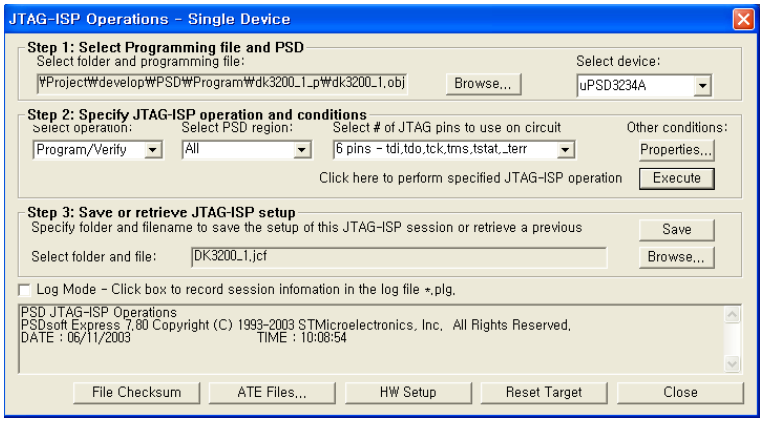

그림 18. Porting 준비

μPSD에 포팅을 마치고 파일의 Checker Sum이나 하드웨어의 리셋 등의 동작을 필요하다면 행해도 된다. 이제 소스 파일에 대해 알아보고 어떻게 동작하는지에 대해 체크할 차례인데 KEIL Compiler 항목을 참고 하자.

3. KEIL Compiler Quick Start

여기서 잠깐 8051에 대해 약간 알고 있어야 부드럽게 Quick Start를 할 수 있으므로 몇 가지 언급하도록 하겠다. 오리지널의 8051은 외부 모양이 40핀으로 직사각형 다이를 이루고 있는 디바이스로

5V에 동작한다. 2개의 머신 사이클(One Machine Cycle=12 Clock)동안 하이(High, 로직 1)가 유지되면 리셋이 되는 RESET 핀이 있다.

포트0은 8비트 오픈 드레인(I/O로 사용할 경우 외부 풀업)으로 구성되어 있으며 양방향 I/O, A0~A7, D0~D7로 이용한다. 포트1은 내부에 풀업이 되어 있는 양방향 I/O이다. 포트2는 내부에 풀업이 되어 있고 양방향 I/O, A8~A15로 이용한다. 포트3은 내부 풀업이 되어 있는 양방향 I/O이며 Serial 통신, 외부인터럽트, 타이머, Read, Write로 사용할 수 있는 특수 기능도 가지고 있다. 1개의 핀에 두개 이상의 기능이 있는 마이크로프로세서가 대부분이며 이런 기능을 Multiplex I/O 라고 한다. X1은 발진기 증폭기의 입력이 반전되어 들어가는 핀이며 X2는 반전되어 출력되는 핀임으로 Crystal일 경우는 X1, X2에 병렬로 연결하지만 Oscillator일 경우는 X1만 연결한다.

ALE/PROG(Address Latch Enable Output/Program Pulse Input)핀에서 ALE는 외부 메모리를 호출하는 동안에 하위 번지 바이트를 래치하기 위해 사용되며 PROG는 EPROM일 경우 프로그램을 기록할 때 펄스의 입력으로 사용한다. PSEN(Program Strobe Enable)핀은 외부 코드메모리를 읽을 경우 사용하는 신호이며 각 머신 사이클에서 2번 출력되며 ALE의 하강에서 포트0이 어드레스 번지로 사용된다. EA/Vpp(External Access Enable)핀은 GND에 연결되면 0~FFFFh 번지까지 코드메모리는 외부 ROM을 사용하게 되고 Vcc에 연결되면 코드메모리는 내부 ROM을 사용하게 된다. Vpp는 8751일 경우 프로그램 기록용 전원입력 단자로 12.75V~21V 사이의 전압을 스팩에 맞게 공급해 주어야 한다.

알나간　　인텔의 8031/8051/8751의 특징

8비트 마이크로프로세서 8051 코어를 사용한 LSI를 말한다. 8031은 ROM이 없는 이름이며 8051은 4Kbyte ROM이나 EPROM(창이 달린 8751)이 내장된 On-Chip이다. 프로세서 내부를 CMOS로 반도체를 만들면 '5' 혹은 '3' 앞에 'C'를 붙여 80C51등으로 불린다. 128*8비트의 내부 RAM이 있으며 32개의 양방향 프로그램 가능한 I/O(Input/Output), 2개의 16비트 타이머, 우선순위를 가진 5개의 인터럽트, 1개의 Serial 포트(완전한 이중 UART), 클락 발진기 내장, 64K 바이트의 외부 Code 어드레스 공간, 64K 바이트의 외부 Data 어드레스 공간, IDLE와 Power Down Mode의 특징을 가지고 있다.

µVision2의 통합 아이콘을 실행하면 다음과 같은 화면이 나오게 된다. 초기 개발 통합 화면이 실행되면 그 동안 작업한 프로젝트가 없음으로 바탕화면이 텅 비어 있게 된다. 일단 상단에 10가지 메뉴가 있는데(V7.0 이상은 11가지) 윈도우 사용자라면 그리 어렵지 않은 Pull Down 메뉴와 아이콘 이다. C51은 프로그램을 관리 할 때 Project라는 파일로 통괄 처리한다. 저자도 처음에는 상당히 어색했는데 여러 개의 프로그램이나 루틴을 일괄처리 하는 데는 당연히 사용해야 한다고 보고 익숙해지면 편하다. Project는 프로그램 환경 심지어 편집에 관련된 정보까지도 고스란히 가지고 가기 때문에 사용자는 책상 위에서 진짜로 여러 가지의 파일을 가져다 놓고 개발하듯이 프로젝트 단위로 C51을 사용하면 된다. 어셈블리와 컴파일러 파일관리를 통합적으로 관리하는 것을 µVision2라 하고 Help 메뉴의 C 컴파일러 버전으로 실제의 KEIL 프로그램 버전번호를 나타낸다.

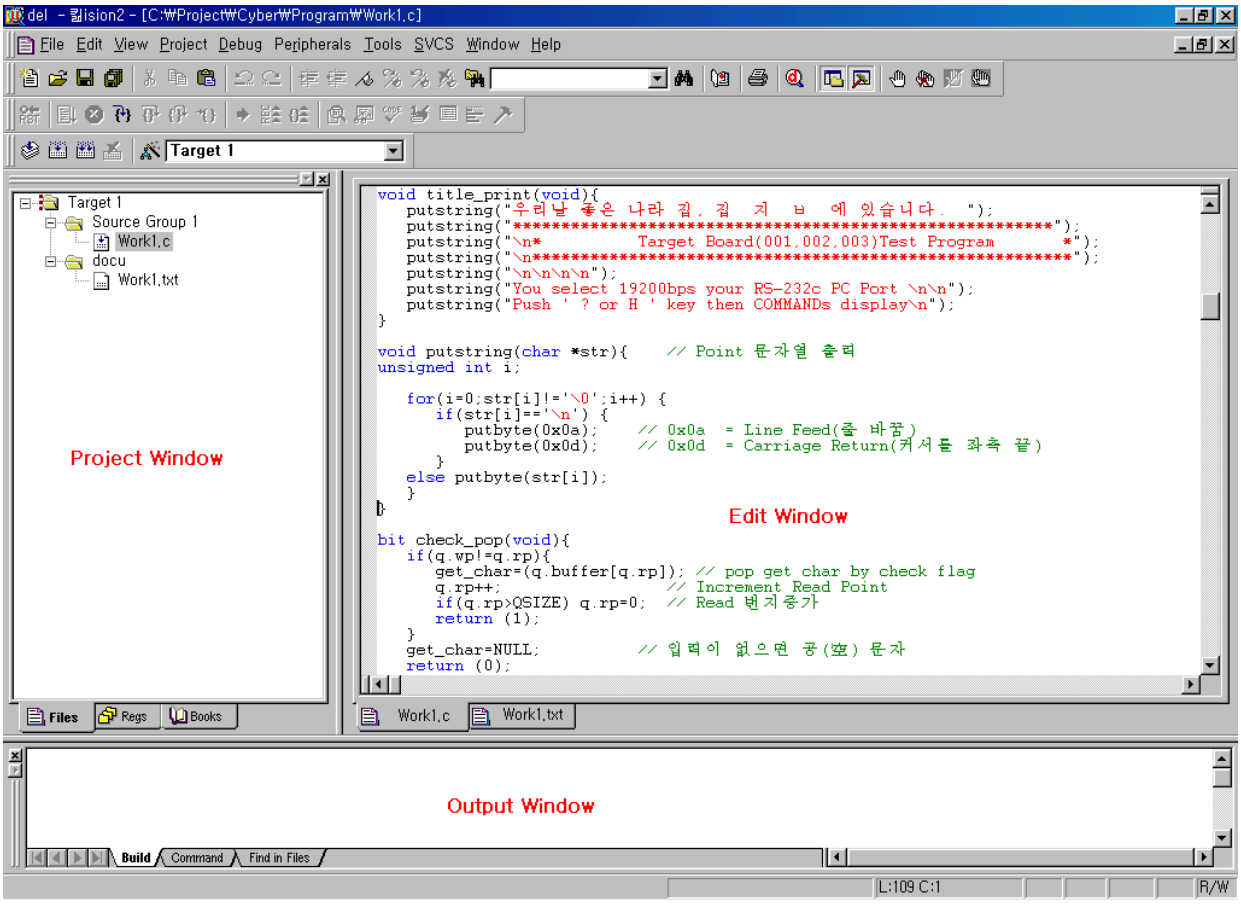

그림 19. µVision2의 실행(V6.1)

C51 8051 카일 컴파일러는 통합 환경으로 바뀌면서 구태여 어셈블리를 모르거나 8051에 대해 그렇게 자세히 알지 못해도 Application 운영 프로그램을 만들 수 있는 환경을 제공해 주는 장점이 있다. 여기서는 엔지니어와 관련되시는 분들의 빠르고 정확한 컴파일러 환경에 대해 익히는 기회가 될 것이다. 통합 환경 C51을 실행하면 화면처럼 크게 3개의 윈도우(Project, Edit, Output)가 나오게 되고 풀다운 메뉴에는 File, Edit, View, Project, Debug, Peripherals, Tools, SVCS, Window, Help 등의 항목이 있다.

우선 새로운 프로젝트를 시작하기 위해 'Project/New Project' 선택한 뒤 가능한 소스 프로그램이 있는 경로에 프로젝트 이름을 기입한 뒤 저장을 누르면 된다. 프로젝트 이름은 통합 환경 왼쪽 상단에 표기되며 나중에 핵사 파일로 만들어지는 파일이름이 되기도 하기 때문에 8자 이상을 넘지 않고 한글 문자를 사용하지 않는 것이 유리하다.

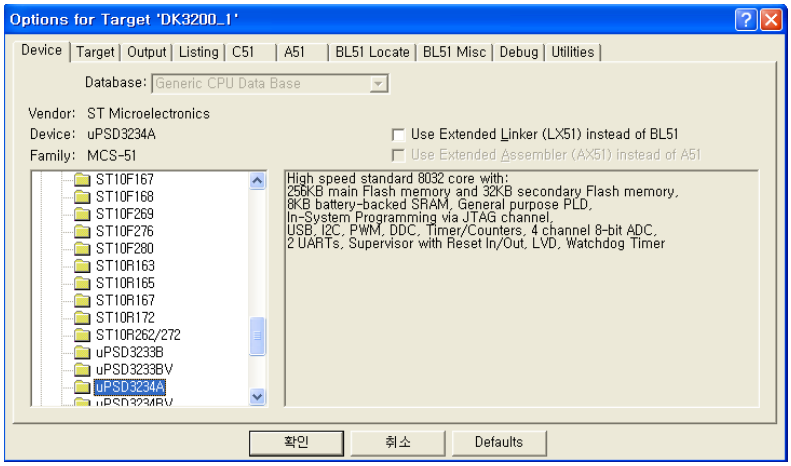

그림 20. Select Device for Target(V7.05)

이어서 선택할 항목은 8051로 개발할 Target의 MPU에 대해 선택하는 'Select Device for Target' 인데 (Project Window에서 마우스의 오른쪽 버튼을 선택해도 된다) 이 항목은 여러 회사에서 만들어진 약간씩 틀리고 특별한 기능이 내장되어 있는 8051을 반영하기 위해서 만들어진 것이므로 자신의 Target에 사용하는 MPU 항목을 정확히 선택하면 된다. V7.05를 기준으로 Acer ~ Winbond까지 현재 44개의 밴드가 나열되어 있으며 ST사의 µPSD 전제품이 등록되어 있다. 디바이스의 선택은 개발환경의 일부분을 변경하기도 하므로 신중을 기울여야 한다. 만약 아직 MPU가 결정되지 않았다면 일단 적절한 PSD를 선택하고 나중에 결정된 다음 변경이 가능하다. 상기의 '그림:Select Device for Target(V7.05)'에서 '확인'을 선택하면 Startup Code를 프로젝트 폴더에 등록 여부를 물어본다. 만들 자신이 없는 개발자라면 자동 등록하는 'Yes'를 선택하자.

그림 21. µPSD 선택 Start up Code 추가여부

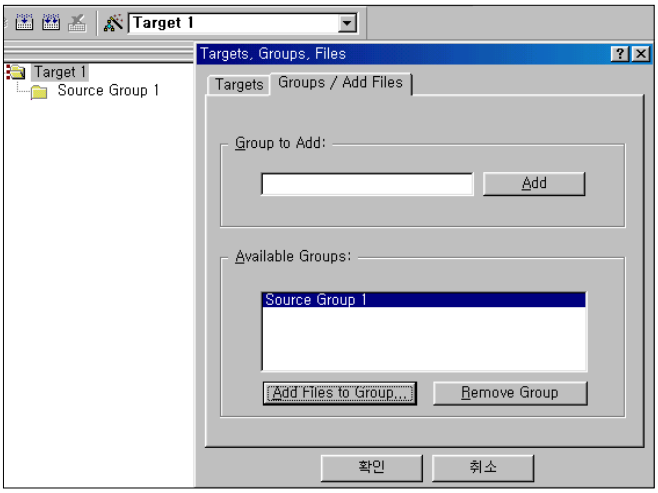

그림 22. Target Groups, Files...

'Target Groups, Files...'에서는 컴파일 하고자 하는 소스 파일을 등록하면 된다. 소스 파일의 기준은 *.C, *.LIB, *.A51 등의 파일인데 확장자는 옵션에서 언제든지 변경이 가능하다. 기존에 이미 작성한 소스 파일이 없을 경우는 워드에서처럼 파일메뉴에서 새로운 이름으로 만든 후 작성하자.

'Project/Targets, Groups, Files...'의 'Target/Target to Add'는 타겟과 해당 파일을 등록하는 것으로 타겟의 이름은 소스 파일에 타겟을 여러 개 둘 경우에(보통 이런 경우는 거의 없다) 해당되며 타겟의 표시는 'Target1'이라고 표시(Default 일 경우)되어 있는 우측 중간 정도에 표시된다.

'Target/Groups_Add Files'는 소스 파일 개발과 함께 해야 할 파일을 등록하는 곳으로 개발 일기도 쓸 수 있고 Documentation 등의 파일을 만들 수도 있는 항목이다. 참고적으로 카일 통합 개발 환경(IDE)의 'Project Window'에서는 저자가 확인하지는 않았지만 소스 파일을 255개까지 등록할 수 있다. 'Edit Window'에서 적절히 소스 파일을 수정한 뒤에는 타겟에 관련된 정보를 수정하면 된다.

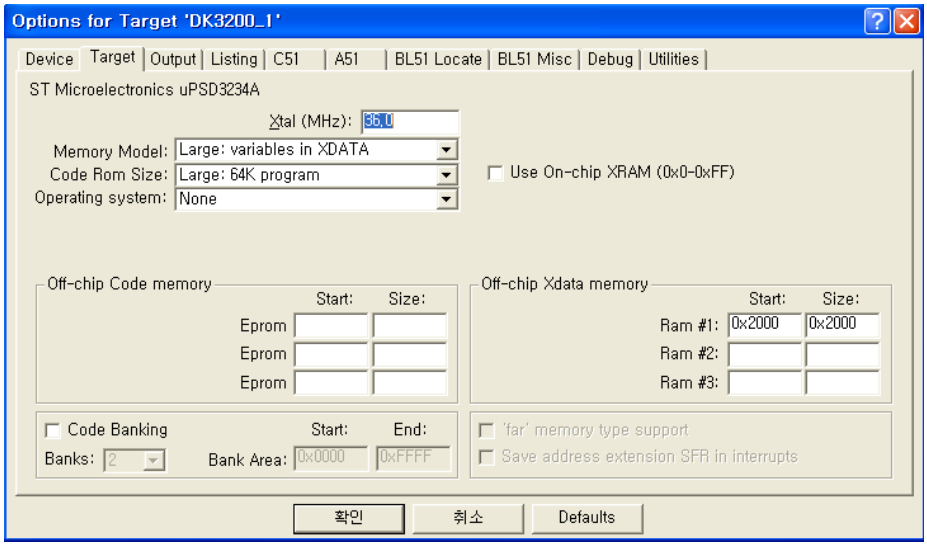

그림 23. Options for Target

소스 등록과 프로그램 입력, 수정을 다 했다면 컴파일과 관련된 정보와 Target에 관련된 사항을 수정하는 'Options for Target' 으로 들어가자. V7.0을 기준으로 두 번째 탭(Tab)에 있는 'Target'은 타겟 보드의 Xtal 값과 메모리 모델 그리고 외부 메모리의 시작번지를 기록하는 매우 중요한 사항을 설정하는 항목이다. 시작은 'Project/Option for Target…….'을 선택하거나 만화영화에서 나오는 요술지팡이처럼 생긴 아이콘()을 선택하면 된다. 이곳에서는 시스템의 크리스탈(Crystal) 진동수 값과 ROM과 RAM의 시작 번지와 크기 그리고 메모리 모델을 입력하면 된다. 저자의 DK3200보드에는 System Clock이 36㎒의 크리스탈과 μPSD3234A 80핀 칩이 장착되어 있다. 메모리 영역은 PSDsoft에서 설정된 어드레스를 기입해야 한다. 만약 이 값이 틀리면 하드웨어 따로 프로그램 따로 노는 격이 된다. ROM 영역의 어드레스는 아무것도 표시하지 않았는데 시작 어드레스가 0번지이고 크기는 설정하지 않았으므로 최대를 의미한다.

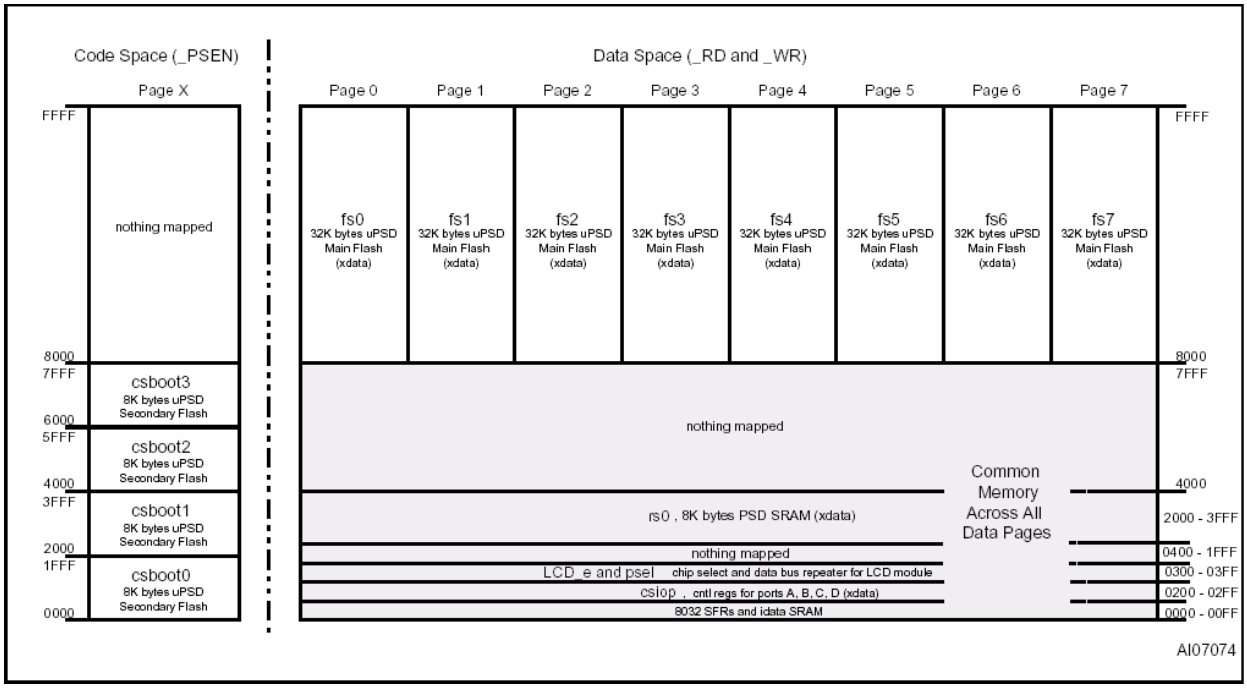

그림 24. DK3200의 Memory Map

8051의 메모리에는 크게 두 가지가 있다. RAM에 해당하는 메모리 부분과 ROM에 해당하는 코드 영역이다. 코드 영역은 다른 말로 Program 영역이라고도 한다. 8051의 메모리 영역은 어떤 메모리든지 내부만 사용할 경우에는 'Small', 외부를 사용할 경우는 'Large'라고 한다. 보기에는 메모리 모델을 코드영역과 데이터영역 모두를 외부에서 사용하는 경우로 Large로 했고 코드 영역은 0번지부터 시작, xdata 영역은 0x2000 번지부터 길이가 0X2000(8K)의 크기를 가짐으로 0x3FFF 번지까지 사용하는 예를 보였다. μPSD3234A에는 8K바이트의 SRAM이 있으므로 RAM 영역의 XDATA(원래는 KEIL에서 칩 외부에 있을 경우에만 사용되는 키워드이지만 μPSD는 내부에 있는 xdata 공간이다) 어드레스의 시작 번지와 크기를 기록해 주어야 한다. 번지는 여기 값을 변경할 경우 사용자가 'Chip Select Equation'의 'rs0'의 어드레스를 변경해서 다른 번지로 할당해도 무방하다.

알나간 8051 KEIL 변수의 선언

8051 프로세서 안에서 사용되는 기본적인 변수는 비트, 바이트, 워드, 롱 워드 등이 있는데 각각 1, 8, 16, 32비트로 설명될 수 있다. 궁극적으로는 비트들의 묶음으로 관리되기 때문에 변수를 정의할 때는 다음처럼 하면 된다. 그러나 컴파일마다 약간씩 다른 변수 정의 키워드가 있는데 이것은 컴파일러 제작 업체가 특정 마이크로프로세서의 메모리 영역을 잘 이용하기 위해 내부적으로 만들어 놓은 것이라고 이해하면 된다.

```
bit test;                // 변수를 1비트 할당, 0과 1의 값을 가진다. 8051은 20h~2Fh 범위
unsigned char test1; // 부호 없는 1 바이트를 어떤 영역에 배치
char test1;              // 컴파일러가 정한 메모리의 어떤 영역에 배치된다. (옵션의 영향)
                         // signed char : -128 ~ +127, unsigned char : 0 ~ 255
```

```
signed int test2;       // 부호 있는 1 바이트를 컴파일러가 정한 어떤 영역에 배치
int test2;              // signed int test2 = signed short test2 동일
                        // signed int : -32768 ~ +32767, unsigned int : 0 ~ 65535
char xdata test3;       // xdata 영역에 1바이트 변수가 배치된다.
xdata char test3;       // 결과는 위와 같다. 민감하다고 판단되면 컴파일러의 매뉴얼을 숙지.
long data test4;        // 내부메모리 영역에 4바이트의 크기로 공간을 할당한다.
unsigned long test4; // signed long : -2147483648 ~2147483647
                        // unsigned long : 0 ~4294967295
float xdata test5;      // ±1.175494E-38 to ±1.402823E+38
```

여기서 잠깐 메모리 구성에 대해 알아보자. 메모리는 내부메모리와 외부메모리로 나눌 수 있는데 8051 코어를 가진 마이컴의 내부메모리는 레지스터 메모리가 128바이트(8032는 256바이트) SFR 128바이트 합해서 256바이트로 구성되어있다. 요즈음에는 512바이트 용량의 8051 코어도 나올 정도로 선택의 폭이 다양하다. 8051의 기본 메모리 구성은 다음 그림 8051 메모리 맵의 회색부분이 없는 상태, 즉 128 바이트와 SFR로 되어 있다. µPSD는 이것에 비해 상당히 진화되어 복잡하게 느껴진다.

외부메모리는 Code(Program)와 Data로 나누는데 어드레스 능력 때문에 최대 각각 64K(0~0xFFFF)로 구성할 수 있다. C 프로그램을 만들고 컴파일 하면 Hex 파일로 만들어진 기계어가 놓일 자리가 Code(Program) 메모리 부분이 되고 마이컴이나 사용자의 연산 명령에 의해 저장된 데이터는 xdata 메모리 부분이 된다. 그래서 개발자가 Source 프로그램을 컴파일하고 난 후에는 이론적으로 Code 크기가 64K, 사용한 xdata 메모리가 64K 이상 되지 않게 하고 Target System에 맞게 Coding을 해야 한다. µPSD는 코드, Program 부분은 모두 뱅크의 Paging 기능을 이용할 수 있다.

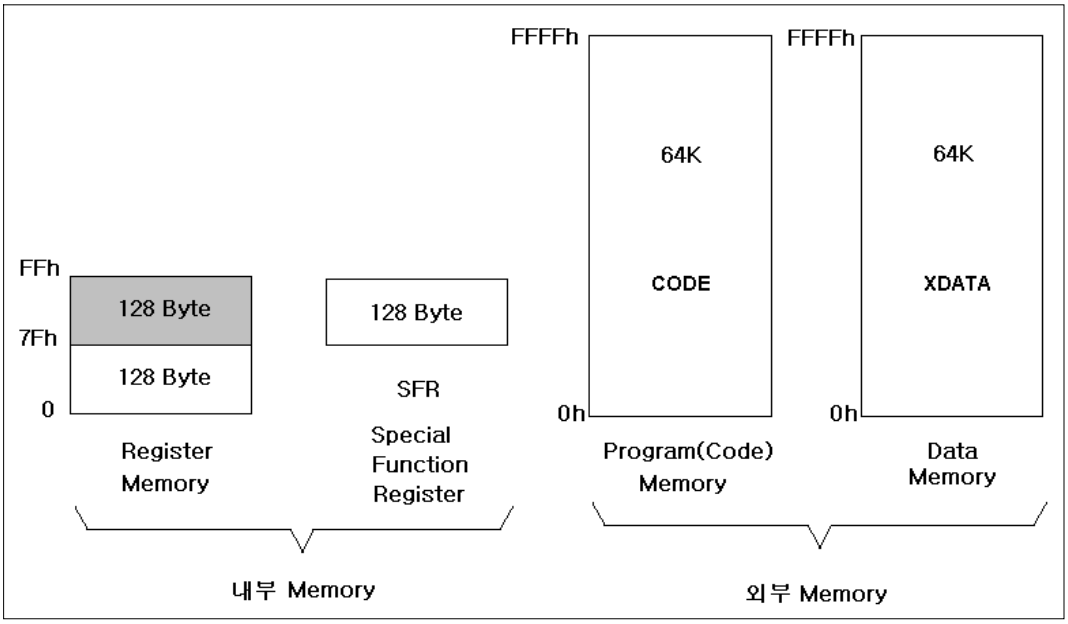

그림 25. 8051 기본 메모리 맵

내부메모리에서 128바이트는(혼동을 피하기 위해 Register Memory라고도 한다) 4개의 범용 레지스터뱅크(0~7, 8~F, 10~17, 18~1F)로 이루어지고 20~2F까지는 비트 어드레스 영역으로 사용할 수 있는 공간, 그리고 보통은 스택이나 사용자 공간으로 이용하는 30~7F까지의 data 영역이 있다. 또 128 바이트로 되어 있는 SFR(Special Function Register)은 21개(μPSD는 추가 되어 더욱 많다)의 레지스트로 구성되어 있는데 C 언어로 프로그램을 하더라도 자세히 알아야 한다.

μPSD는 SFR 영역이 멀티플렉스(Multiplex) 기능을 구현하기 위해 추가 되었고 PA, PB, PC, PD 등의 PLD 영역을 제어하기 위해 csiop 부분이 따로 있다. 추가된 레지스터는 내부적으로 크기가 고정되어 있는 반면에 어드레스의 시작 번지는 칩 내부에 Decode가 있어 범위 안에서 변경이 가능하다.

다음은 KEIL에서 사용된 csiop의 구조체 정의와 사용 예를 보인다.

```
typedef xdata struct REG_PSD_struct {// 반드시 xdata 영역 할당
        unsigned char DATAIN_A;          // PSD_REG_BASE +0x00
        unsigned char DATAIN_B;          //              +0x01
        unsigned char CONTROL_A;         //              +0x02
        unsigned char CONTROL_B;         //              +0x03
         중략....
        unsigned char PAGE;              //              +0xE0
        unsigned char res1e;             //      spacer
        unsigned char VM;                //              +0xE2
     unsigned char res29[0x1d];      //       spacer
} PSD_REGS;

PSD_REGS upsd _at_ PSD_REG_ADDR;  // Define PSD registers at address "csiop"(300번지)

void write_data(unsigned char val){// 의미 없는 프로그램
unsigned char pass=0x30;
   upsd.CONTROL_A=val;   // 300+2=302번지
   pass+=val;
   upsd.DATAOUT_B=pass;
}
```

SFR은 크게 소프트웨어 제어, 연산과 I/O를 제어하는 부분의 두 가지로 나눌 수 있는데 우선 P0~P3은 I/O 포트와 연결되어 있는 레지스터이다. ACC(Accumulator)는 모든 연산의 중심이 되며 산술, 논리에 이용된다. B는 ACC와 더불어 곱셈에서는 상위값을 저장하고 나눗셈에서는 나머지를 저장한다. PSW는(Program Status Word) 프로그램 수행 때 상태 결과를 나타내는 역할을 하며 CY, AC, F, RS1, RS2, OV, P 등이 있다. SP(Stack Point)는 리셋 되었을 때 초기값은 07h이며 스택의

최하위 번지를 지정해서 PUSH 되면 자동으로 증가하고 POP 되면 감소한다. 최대 증가 번지가 내부 메모리를 초과하면 시스템이 정지되므로 넘치지 않도록 주의가 필요하다. C 언어로 KEIL을 이용해서 컴파일 할 경우 소스 안에서 내부 메모리를 과다하게 사용하지 말아야 한다. 스택 Over의 맹점은 컴파일 할 때는 에러 발생을 알 수가 없고 Target을 동작 해봐야 한다는 것이다.

DPTR(Data Pointer Register)은 8비트의 DPH와 DPL로 구성되어 외부 데이터 메모리의 16비트 번지를 지시할 때 사용한다. SBUF는 직렬통신에서 송신, 수신버퍼로 이용된다. PC(Program Count)는 프로그램의 실행 번지를 지시할 때 사용하며 SFR이나 내부 RAM 레지스터 메모리와 독립적으로 움직이는 16비트 레지스터다. 이밖에 제어레지스터로는 IP, IE, TMOD, TCON, SCON, PCON 그리고 타이머 레지스터가 있다. 본인이 미처 설명하지 못한 내용은 인텔 웹사이트(http://www.intel.com/design/product.htm)의 Microcontrollers/MCS 51 microcontroller family를 참조하면 된다. 이곳에는 마이컴의 본거지답게 여러 종류의 8051 관련 Data Sheet가 있으므로 더 나은 지식을 얻을 수 있으리라 본다.

또한 ST사의 μPSD에 대한 통합 정보는 'http://www.st.com/psd/'에 있으므로 최신의 데이터시트, Application 그리고 주요사항들을 참고 하면 된다. 칩에 대한 문의는 대리점 '비에스아이코리아(02-2067-0700:www.bsisemicon.com) 또는 ST 한국 본사(02-3489-0114)로 연락하면 도움을 받을 수 있다.

메모리 모델을 설정한 뒤에는 프로그램 상에서 사용할 때 컴파일러에서 제공되는 키워드를 사용해야 한다. 8051은 특히 메모리가 여러 종류가 있는데 메모리 모델의 크기에 따라, 위치에 따라 각각 다른 곳에서 동작한다. 다음은 실제로 사용된 예를 나타낸다.

```
sfr WDCON  = 0xD8;        // SFR 영역 설정
sbit SMOD_1 = WDCON^7; // 7번째 비트 정의
bdata testbyte;
sbit byte0=testbyte^0;        // 설정한 변수의 testbyte LSB
sbit byte7=testbyte^7;        // 설정한 변수의 testbyte MSB
bit timeoutcheckbit;          // 0, 1의 두 가지 값만을 가지는 변수 선언

unsigned int xdata secutimerinterval;     // 부호 없는 16비트 변수를 xdata 영역에 설정
unsigned int sirentime;                   // 옵션에 따라 메모리 설정이 변경된다.
char xdata ramdata[6][6];         // 2차 배열 선언, 옵션을 'Large'로 했다면 자동으로 xdata
unsigned char data test1;  // 내부 메모리 영역의 부호 없는 1 바이트 설정
char code test2=3;        // 1 바이트의 코드 영역을 설정한 초기값으로 3을 넣는다.
unsigned code segment[4]={0xbe,0xe0,0xfe,0xf6};  // 코드 영역에 배열 설정
SAVE xdata timer_check;// SAVE Struct를 xdata 영역에 구조체 선언
char test3;                // 1 바이트를 컴파일러가 알아서 설정, Default는 data영역에 위치한다.
```

Type	설 명	예
code	프로그램(코드)메모리 최대 64K바이트 사용	char code test;
data	직접 내부 데이터메모리에 번지 지정, 128바이트 사용, 변수형 키워드를 사용하지 않으면 기본설정(Default)	char date test; char test;
idata	간접 내부 데이터메모리 256바이트 사용	char idata test;
bdata	비트 지정 내부 데이터메모리 16바이트 사용	char bdata test;
xdata	외부 데이터메모리 최대 64K바이트 사용	char xdata test;
pdata	page된 외부 데이터메모리 256바이트 사용	char pdata test;

표 3. KEIL의 메모리 타입

KEIL 8051 C 컴파일러에서 메모리와 관련되어 사용하는 명령어 키워드(Keyword)는 code, xdata, pdata, data, idata, bdata가 있다. 먼저 외부메모리에 있는 Code 메모리는 사용자에 의해 'code' 키워드를 사용하여 어드레스를 지정하고 Firmware에 이식 후 기계어를 로딩하기도 하지만 대부분 컴파일러에게 Code 배치를 알아서 하도록 위임한다. Code는 일단 운영(Run)중에는 읽기만 가능하고 쓰기를 할 수 없으므로 사용자 지정은 상수값이나 Font 등의 정보저장 장치로 이용한다.

'xdata', 'pdata' 명령어를 사용하는 RAM 메모리는 마이컴의 운영중에 발생하는 정보저장 장치로 사용하고 읽고 쓰기가 자유롭다. C 언어로 보면 프로그램 전체에서 사용하는 글로벌 변수와 함수 내에서 사용하는 로컬 변수 그리고 배열 등의 변수 저장 영역으로 'xdata'가 사용될 수 있으며 내부 RAM에 비해 쓰고 읽는 속도가 느리다.

내부 메모리는 레지스터 메모리의 크기에 따라 사용하는 명령어가 다르다. 8051은 내부 레지스터 크기가 128바이트에서는 'data', 256(레지스터메모리의 128K바이트 어드레스 맵에서 회색부분)바이트에서는 'idata', 그 이상은 다시 'xdata' 그리고 16바이트 영역의 0x20~0x2F(128K바이트 어드레스 구성도의 00~7F)까지만 사용할 수 있는 'bit', 'bdata' 명령어로 나눌 수 있다. bdata는 뒤에 비트로 조정할 수 키워드가 별도로 있다. 다음은 개발할 때 참고가 될 수 있도록 가장 필요한 변수의 일부분을 요약 정리했다.

용 어	비트/바이트	범 위	보 기

bit
 	
sbit
	1 / -	0 ~ 1	bit test(void); // 함수의 리턴값
			char bdata d;// bdata 선언
			sbit d0=d^0;// d 변수 LSB
			sbit d7=d^7;// d 변수 MSB
			sbit EA=0xAF; // 비트 선언

signed char	8 / 1	-128 ~ +127	signed char xdata test;
sfr unsigned char	8 / 1	0 ~ 255	sfr P0=0x08; unsigned char xdata test;
enum signed int signed shot	16 / 2	-32,768 ~ +32,767	signed int code test=-1234; ※ signed는 생략해도 된다.
sfr16 unsigned int unsigned shot	16 / 2	0 ~ 65,535	sfr16 test=0xCC; // 16비트 선언 unsigned int xdata d=0xFFFF;
signed long	32 / 4	-2,147,483,648 ~ 2,147,483,647	signed long xdata test;
unsigned long	32 / 4	0 ~ 4,294,967,295	unsigned long xdata test;
float	32 / 4	±1.175494E-38 ~ ±3.402823E+38	float xdata test;

표 4. KEIL C51의 데이터 타입

알나간 C51 키워드(Keyword)

C51에서 미리 등록해 놓고 사용하는 단어로 프로그램 작성할 때에는 변수이름이나 루틴이름을 키워드에 있는 단어와 동일하게 사용할 수 없다. C51은 대문자, 소문자를 구분하는데 키워드는 소문자로 되어 있다. 아래에 KEIL C51에서 사용하는 키워드의 일부를 나타내었다.

_at_	alien	bdata	bit	code	data	idata
large	pdata	sbit	sfr	sfr16	small	_task_
using	xdata	_priority_	reentrant	compact	interrupt	

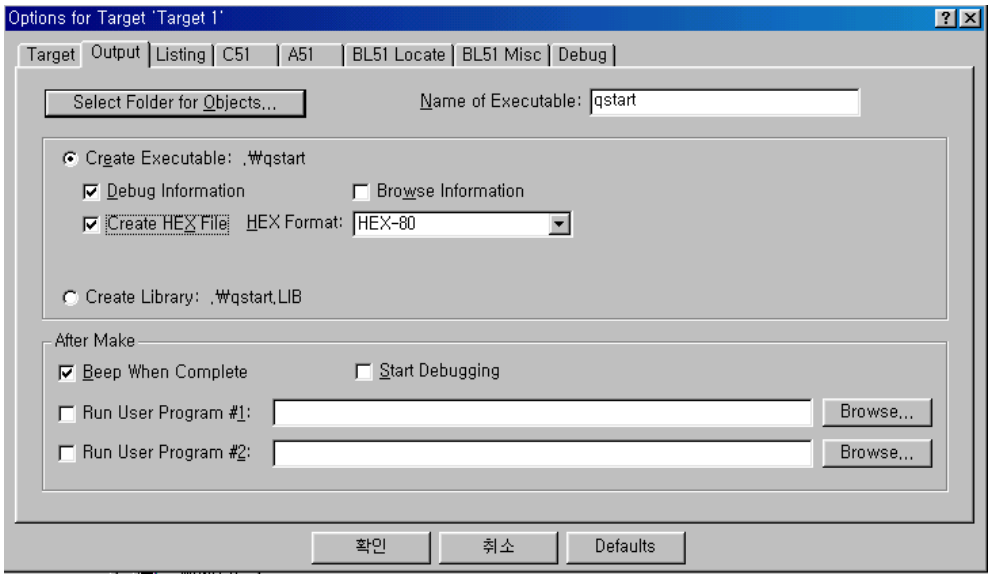

그림 27. Options for Target 출력 파일 설정

다음 Option 항목은 8051의 기계어 코드를 롬에 이식하기 위해서 Hex 코드를 만들어 주는 곳인데 어떤 이름으로 어디에('Select Folder for Objects...') 출력할 것인가를 설정하는 항목이다. 엔지니어가 컴파일하고 디버깅을 한 다음 ROM에 이식(Porting) 하게 되는데 목표에 도달하기까지 많은 컴파일을 시도하게 된다. 이때 반복적인 쓰기 동작으로부터 하드디스크의 섹터와 트랙을 보호하기 위한 것으로 'Create HEX File' 선택 항목을 위의 그림처럼 체크하지 않으면 컴파일 한 뒤(에러가 발생하지 않아도) 보고서는 만들어지지만 HEX 코드는 절대 만들어지지 않는다.

오른쪽 상단의 'Name of Executable'는 핵사 코드의 파일과 Report의 이름을 나타내는데 보통은 프로젝트 이름과 동일하다. 만약 HEX 파일이름을 프로젝트와 동일하게 하는 것을 원하지 않는다면 다른 이름으로 변경하면 된다. 보통은 컴파일이 완료되면 프로젝트가 있는 경로에 프로젝트 이름으로 핵사 코드가 생성된다. 8051은 지적 재산을 Intel이 가지고 있고 명령어 처리도 인텔의 구조를 따라가므로 Intel 핵사(HEX-80) 형식만 KEIL에서 지원한다. 그리고 Source 개념으로 디버깅을 할 수 있는 옵션인 'Debug Information'과 프로그램을 편집할 때 사용하기 편한 'Browser Information'등은 선택해 두는 것이 여러모로 편하다. 다음의 화면은 'Browser Information'의 Check Box를 선택했을 경우 편집 화면의 소스 파일 위에서 마우스의 오른쪽 버튼을 찾아가고자 하는 단어 위에서 누른 직후의 모습이다.

'Create Library'는 Source File을 Library로 만들어 애써 만든 C 언어 소스를 보이는 것으로부터 보호하기 위해 사용하는 옵션이다. 만드는 방법은 "How to use KEIL 8051 C Compiler", 양서각출판. 단행본을 참고하면 된다. 'After Make' 항목은 컴파일과 링크를 마친 뒤 별도로 실행하는 사항을 지정 할 수 있다. μPSD가 아닐 경우는 보통 롬 에뮬레이터(ROM Emulator) 실행 파일과 경로를 연결한다.

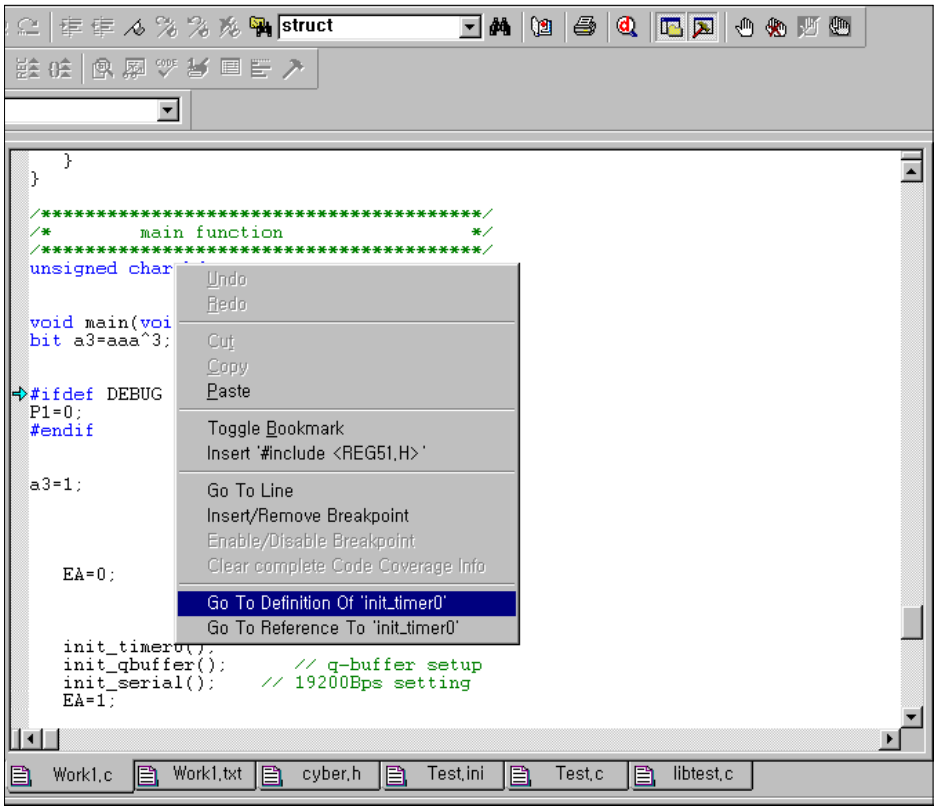

그림 28. 'Browser Information'의 Check Box를 선택했을 경우

알나간(알고 나면 간단) 인텔 HEX Format

인텔 HEX File Format은 바이너리 코드를 ROM Writer등으로 다운로드하기 위해 만들어진 형식으로 데이터를 바이트 단위로 한 뒤 ASCII 코드로 표현한다. 예를 들어 코드의 16진수 '09'는 0x30('0'), 0x39('9')의 상태로 전송한다. 다음에 제시된 핵사코드는 이해를 돕기 위해 만들어진 Sample인데 실제로 첫줄을 분석해보도록 하자.

```
:09000F00D2A0C2A0B28080F82248
:03000000020003F8
:0C000300787FE4F6D8FD75810702000F3D
:00000001FF
```

:-> 인텔 Hex Format의 레코드(Record) 시작

09-> Record의 데이터 바이트의 수, 밑줄 부분의 Hex의 개수를 나타낸다. 밑줄 친 부분은 어셈블리 소스의 니모닉 코드에 해당한다.

000F-> 데이터 바이트를 저장할 ROM의 어드레스 위치 000Fh번지를 나타낸다.

00-> Record 형식, 01이면 마지막 Record를 나타낸다.

D2~22-> 컴파일 후의 Hex Data, 실제로 이 부분을 Code Memory 영역에 기록한다.

48-> 09+00+F0+00+D2+A0+~+F8+22의 2의 보수(합을 구한 후 반전하고 다시 1을 더함) 값을 나타낸다.

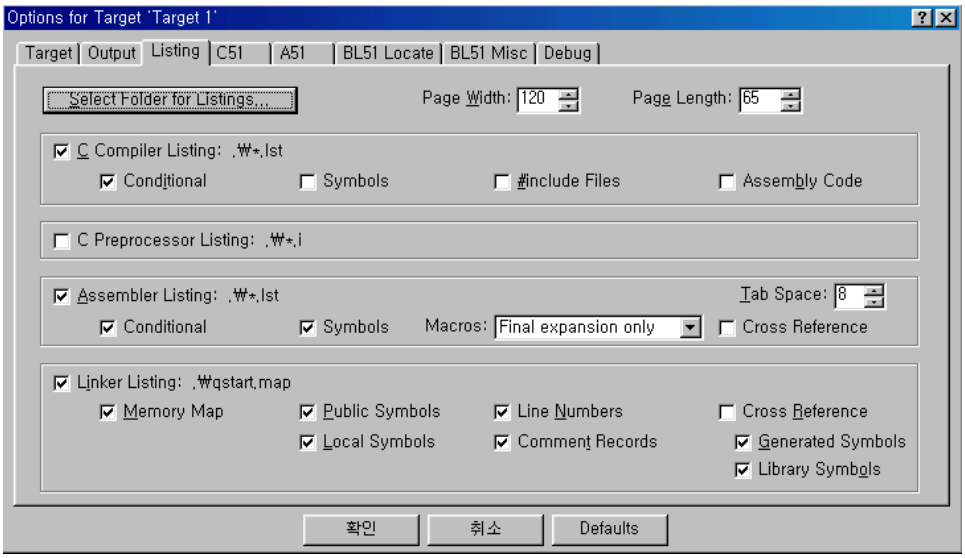

그림 29. Options for Target Listing

C 소스 코드를 컴파일하고 나면 *.lst 파일이 생성된다. List 파일은 C 소스를 어셈블리로 만든 리스트와 메모리 사용의 결과 보고서를 나타낸다. 본 항목에서는 디폴트 설정 이외에 추천해 주고 싶은 것이 있다면 'Assembly Code' 항목을 선택하는 것이다. 불과 5년 전만 해도 시스템의 단가를 떨어트리기 위해 컴파일러를 사용하지 않고 어셈블리 언어로 프로그램을 하는 개발실이 많았지만 현재는 알고리즘이 보다 복잡하고 어셈블리만으로는 관리나 개발 기간을 맞추기가 어렵기 때문에 컴파일러를 많이 사용한다. 하지만 어셈블리 언어를 잃어버리면 안 된다고 생각하는데 8051은 제어를 하기 위해 제작된 MPU이므로 Firmware 엔지니어가 변환된 기계어를 보고 버그를 잡을 필요 혹은 효율에 대해 냉철하게 확인할 필요가 있기 때문이다. *.lst File은 프로젝트 윈도우에 등록된 C 소스 숫자만큼 생성된다.

다음은 Hello 프로젝트의 Hello.lst 파일의 예를 보인 것이다. List 파일의 처음에는 사용한 컴파일러의 버전 번호 그리고 생성된 시간 날짜를 기록해 놓고 그들의 경로 그리고 실행파일의 위치 등에 대해 나열한 것을 볼 수 있다. 소스 코드와 행 번호가 나타난 이유는 'Listing/Conditional'의 항목을 선택했기 때문이다. 이렇게 리스트 파일은 컴파일 한 뒤의 소스 파일에 상응한 어셈블리를 확인하고 에러 여부와 사용된 Resource를 점검할 수 있도록 고안된 보고서에 해당한다. List의 출력파일 안의 추가 설명을 주석문으로 표기했다.

```
C51 COMPILER V6.12  HELLO                    PAGE 1   // 원래는 날짜, 시간도 있다.
C51 COMPILER V6.12, COMPILATION OF MODULE HELLO
OBJECT MODULE PLACED IN .₩HELLO.OBJ
COMPILER INVOKED BY: C:₩HARDWARE₩KEIL₩C51₩BIN₩C51.EXE .₩HELLO.C DEBUG OBJECTEXTEND CODE
stmt level     source    // Hello.C의 소스 코드
   1           /*-------------------------------------------------------------------------
   2           HELLO.C
소스 코드 중략...
  23           void main (void) {
소스 코드 중략...
  41    1        while (1) {
  42    2          P1 ^= 0x01;                /* Toggle P1.0 each time we print */
  43    2          printf ("Hello World₩n");    /* Print "Hello World" */
  44    2        }
  45    1      }
ASSEMBLY LISTING OF GENERATED OBJECT CODE      // Assembly 코드 명시
             ; FUNCTION main (BEGIN)                  // 함수의 시작 명기
                                          ; SOURCE LINE # 23    // 소스 코드의 행 번호
                                          ; SOURCE LINE # 29
0000 759850              MOV      SCON,#050H
                                          ; SOURCE LINE # 30
0003 438920              ORL      TMOD,#020H
                                          ; SOURCE LINE # 31
중략...
0016 120000       E      LCALL    _printf      // 왼쪽 숫자는 코드 메모리의 절대번지, 120000은 니모닉 코드
                                          ; SOURCE LINE # 44
0019 80F2                SJMP     ?C0001
             ; FUNCTION main (END)          // 함수의 끝 알림
MODULE INFORMATION:    STATIC OVERLAYABLE     // 메모리 사용 정보
   CODE SIZE         =     27     ----              // 크기는 십진수이며 바이트
   CONSTANT SIZE     =     13     ----
   XDATA SIZE        =    ----    ----
C51 COMPILATION COMPLETE.  0 WARNING(S),  0 ERROR(S)   // 워닝과 에러 안내
```

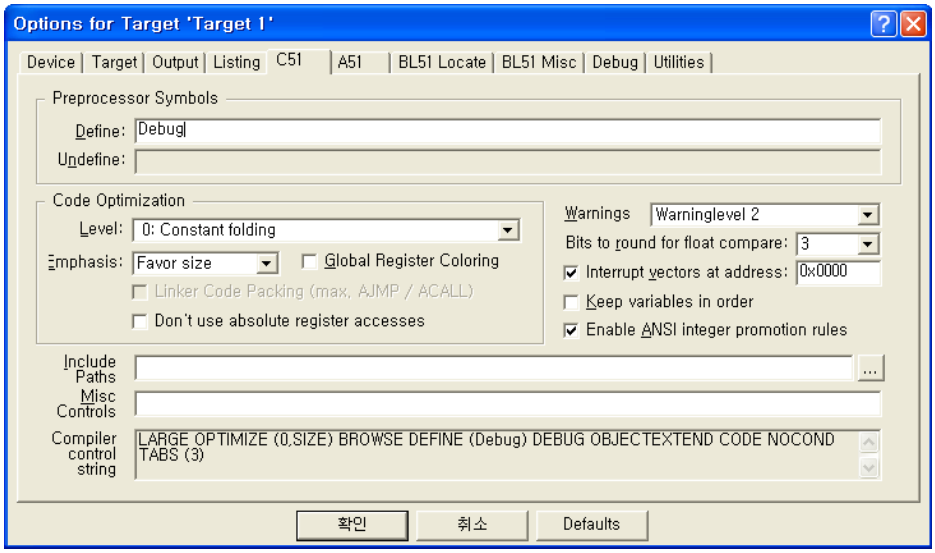

그림 30. Options for Target C51

'C51'은 C 언어를 위한 항목으로 구성되어 있다. 전처리기에서 운용되는 'Define' 항목과 코드의 효율을 높이기 위한 'Code Optimization' 항목과 'Emphasis'로 구성되어 있는데 원하는 항목으로 선택하면 된다. 특히 'Code Optimization'의 'Level' 항목 번호는 높을수록 요약이 잘 되는 것은 아니지만 해당 번호의 특징이 나름대로 정의되어 있으므로 관련 매뉴얼(Getting Start)을 보면 된다. 이중에서 레벨을 0으로 선택하면 C 소스 프로그램에서 정의한 대로 메모리에 할당되므로 프로그램에 자신이 있고 수동으로 처리하기를 원하는 엔지니어는 0번으로 선택하는 것이 좋다. 하지만 최적화는 Compiler가 하지 않는다. 사실 많은 엔지니어들이 8051에 대해 잘 알고 있으므로 메모리 할당을 본인의 프로그램대로 컴파일하고 싶다면 이렇게 하면 된다. 저자는 메모리 용량에 구애받지 않는 프로젝트라면 기꺼이 0번을 선호한다.

전처리기(Preprocessor)의 사용 예제를 다음에 보였는데 전처리기에 둘러싸인 문장을 컴파일하기 위해서는 'C51/Define' 항목에 'Debug'라고 입력해야 한다. 대/소문자 구분되므로 주의가 필요하다.

```
#ifdef Debug
   P1=0xFA;    // 'Debug' 단어를 만나면 컴파일 된다. 그렇지 않으면 Skip
#endif
```

전처리기는 main() 내부나 외부 그리고 다른 함수 내/외부, 프로그램의 시작 등에 자유롭게 사용할 수 있는 C 언어의 사전 처리임으로 카일 컴파일러에만 특별히 있는 사항은 아니다. '#ifdef'의 마지막은 '#endif'으로 if 문의 마지막을 알려주기 위해 사용된다. '#ifndef'도 있는데 C 언어 매뉴얼의 전처리기 항목을 참고하자. 오른쪽의 'Warning' 항목은 컴파일 할 때의 경고 메시지를 얼마만큼 자세히 나타내는가에 대한 항목으로 번호가 높을수록 보다 꼼꼼히 검사 후 자세히 메시지를 보여준다.

지금까지 Target의 옵션을 설명했는데 항목을 설정 후 컴파일과 디버그를 하고 HEX 코드를 Porting하면 된다. KEIL의 프로젝트는 종료 된 뒤 다시 프로그램을 실행하면 가장 최근의 환경과 화면 상태

를 다시 제공해 준다. 현재 활성화 되어 있는 편집윈도우의 소스 파일만 컴파일 할 경우는 'Translate' 아이콘()을, 그리고 C 소스가 여러 개 있을 경우 하나의 소스만 수정했다면 파일 전부를 컴파일 할 필요가 없음으로 변경된 파일만 컴파일하고 다른 컴파일 된 것과 링크 하는 'Project/Build Target'()을 선택하거나 소스 파일 전부 컴파일하고 링크 하는 'Project/Rebuild all Target Files'()를 클릭하면 된다. 추천해 주고 싶은 것은 'F7' 혹은 '' 아이콘을 눌러 수정된 파일의 컴파일과 링크를 하는 것을 권하고 싶다. 'Project/Build Target'과 'Project/Rebuild all Target Files'는 소스코드가 하나일 경우는 서로 동일하게 무조건 컴파일 한다.

프로젝트 윈도우에 등록된 소스코드에 에러가 없으면 컴파일 뒤 OBJ 파일이 생성되고 OH51.EXE 파일이 호출되어 HEX 코드가 만들어진다. 최종적으로 컴파일과 링크를 에러 없이 거친 뒤에는 프로젝트 이름과 소스 이름으로 *.lst 컴파일 보고서가 만들어지고 Project 이름으로 *.M51 Link 보고서가 만들어진다. HEX 코드 생성 옵션을 선택했다면 Project 이름으로 *.HEX 코드가 생성되고 확장자가 없는 OMF51(MDS 장비에서 사용) 파일도 생성된다.

다른 Application에서는 HEX 파일을 ROM에 Writer를 이용하여 구운 다음 Target에 넣고 실행하는데 특별한 장치(8051 Emulator, ROM Emulator, ICE)가 없다면 이런 과정을 셀 수 없이 많이 하게 될 것이다. 하다 보면 μPSD의 JTAG Cable 'Flashlink'를 통해 로딩하고 플래시 메모리에 기록 하는 것이 개발에 좀더 효율적이라는 것을 느낄 것이다.

4. dScope

소스 코드를 에러 없이 컴파일 했다면 디스코프(dScope)를 운영해서 μPSD 보드와 양방향 통신하며 변수의 값을 읽고 Break를 걸어 검증할 수 있는 Simulator[1]을 해보자. 제품이나 상품을 개발할 때는 하드웨어 설계와 소프트웨어 설계(어플리케이션 제작)에 시간적 차이가 발생하여 소프트웨어를 먼저 검증할 필요가 생길 때 Simulator를 이용한다. 또 현재의 Application 개발은 여러 사람이 개발하고 한 사람이 Marge 하는 경향이 많아 하드웨어로 동시에 검증이 어려울 경우에 각자가 작성한 모듈을 통합개발 환경의 디스코프로 사전 점검할 때 이용한다. 카일 컴파일러 IDE(Integrated Development Environment)의 한 부분인 디스코프는 내부 운영을 볼 수 있다고 해서 Simulator를 Debug+Scope=dScope 라고 한다. 여기서는 작성한 Application을 디스코프를 이용하여 μPSD의 Target과 어떻게 인터페이스 해야 하고 검증하는 방법에 대해 나열했다.

디스코프는 V7.0 버전부터 'Keil ISD51 In-System Debugger' 항목이 포함되어 있어 μPSD와 동작 중에 UART(Universal Asynchronous Receiver Transmitter) 포트를 통해 디버깅을 할 수 있는 기능이 있다. 물론 플래시에 모니터를 할 수 있는 함수를 탑재해야 하고 시리얼에 관련된 정보를 설정해 주어야 한다. 먼저 디스코프를 운영하기 위해서는 'Option for Target'()의 'Debug' 항목에서 다음의 그림처럼 설정해야 한다.

밑에 있는 'Load Application at Startup' 항목은 디스코프를 시작할 때 자신의 어플리케이션 (프로그

1) Simulator는 소프트웨어 적으로 동작을 흉내 내는 용어로 많이 사용되고 Emulator는 하드웨어적으로 흉내 내는 장치의 용어로 사용된다.

램이 정상적으로 컴파일 되었을 경우) OMF 파일을 불러 올 수 있도록 선택하는 사항이므로 'V'자를 표시하자. 선택하지 않고 디스코프를 실행했다면 수동으로 Load를 취하면 된다.

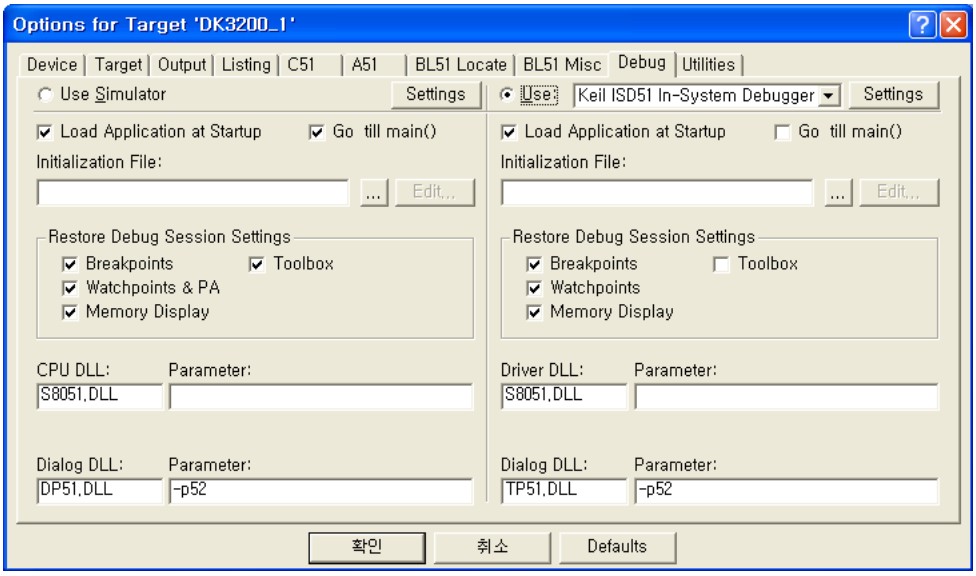

그림 31. μPSD의 디스코프 실행 위한 설정

'Go till main()'은 dScope를 시작한 뒤 C Source의 main() 함수에서 시작하는 것이므로 선택해 두는 것이 편하지만 μPSD에서는 Start-up Code 때문에 만족할 만큼 Main 함수로 들어가지 않는다. 또 컴파일하고 디스코프를 실행할 때 일련의 과정을 도스의 배치파일처럼 하여 편리하게 환경을 구축하고 싶다면 'Initialization File' 항목에서 일련의 명령어가 들어가 있는 파일을 브라우저를 이용해서 등록하면 된다. 대부분 버튼을 등록해서 사용하게 되는데 'Toolbox' 창으로 표시된다. 버튼을 만드는 문법은 컴파일러를 설치하면 자동 생성되는 파일(*.ini)을 참고하고 여기서는 별다른 기술이 아니므로 언급하지 않았다.

또 오른쪽의 'Setting' 버튼을 누르면 디스코프와 통신하기 위한 환경을 기록하는 항목이 나오는데 통신 포트와 RTS, DTR 항목을 올바르게 넣어 주어야 한다. 특히 통신 속도는 19200Bps로 함과 동시에 인터페이스 하고자 하는 Targe의 클락도 Bps와 정수배가 될 수 있도록 보다 정확한 값으로 디자인 되어 있어야 한다. Setting 환경을 바꾸길 원한다면 디버그에 관련된 함수도 변경해야 되는데 몇 개를 제외하고는 내부 함수로 이루어져있다.

19200Bps로 설정하기 위한 프로그램의 일부

```
T2CON    = 0x34;     // Use Timer 2 as baud rate generator
PCON    |= 0x0C;     // Set UART2 to user timer2 baud rate
timer2_baud = (65536L - ( (FREQ_OSC * 125L) / (4L * 19200L))); // FREQ_OSC=24576㎑
RCAP2L = (timer2_baud & 0x00FF); int 변수
RCAP2H = (timer2_baud >> 8);
```

```
SCON    = 0x50;         // enable first serial UART & receiver
SCON2   = 0x50;        // enable 2nd UART
EA = 1;                 // Enable global interrupt flag
// ================================================
```

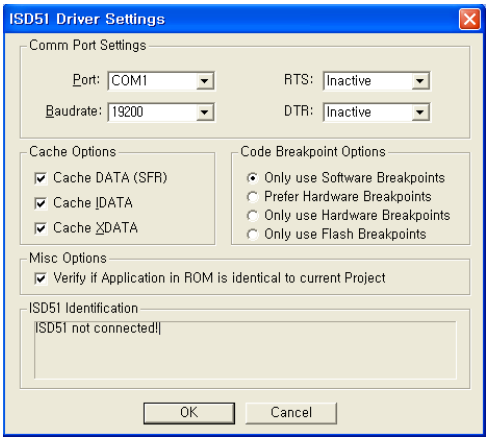

그림 32. ISD51 설정 값

알나간 EIA/TIA-232E, RS-232C

EIA(Electronic Industries Association), TIA(Telecommunications Industry Association)의 약어로써 1991년에 개정되었다(RS-232C의 규격은 1969년). 표지에는 'Interface Between Data Terminal Equipment and Data Circuit Terminating Equipment Employing Serial Binary Data Interface'로 우리말로 하면 '시리얼이진 데이터에 의한 DTE와 DCE 사이의 인터페이스'를 말한다. 가장 널리 퍼진 1969년의 표준 EIA RS-232C(이후 1986년의 RS-232D, 1991년의 RS-232E)에 의해 보레이트를 변조하는 걸로 대체되었다. 표준 RS-232는 양쪽 송신기(Driver)와 수신기(Receiver)의 특징을 하고 있는데 현재가장 널리 보급된 중/저속 인터페이스 사양이며 권장 케이블 길이는 15m 이하이다. CCITT 권고사항의 V.24/V.28과 일본 JIS X5101과는 기능적으로 호환성이 있다. 시리얼 통신은 232외에도 20KBps 이하 속도를 가진 통신을 통칭하며 그 이상의 데이터전송 속도는 EIA/TIA-530, EIA/TIA-560, EIA/TIA-574를 권장한다.

컴파일을 마친 뒤 기계어를 타겟 롬에 이식하는 것을 Porting 이라고 하는데 이때 쓰고 혹은 지우는 작업이 빈번하면 할수록 개발 속도를 떨어뜨리는 이유가 된다.

디스코프의 실행은 소스코드를 컴파일 할 때 경고는 상관없지만 에러가 없을 경우만 실행할 수 있으며 반드시 프린터 포트에 락키가 있어야 한다. 그렇지 않을 경우는 소스코드가 2K 한도의 평가 버전으로 자동 전환되므로 자신의 컴파일러의 정품여부 확인을 먼저 하고 시작하도록 하자. 디스코프는 디버그와 스코프의 합성어로 돋보기 모양 안에 적색으로 'd'라고 적혀 있는 아이콘()을 실행하면 된다. 디스코프를 실행한 뒤에는 아이콘의 활성화 된 부분이 KEIL의 실행 메뉴와 다른 것을 느낄 것

이다.

활성화 된 디스코프의 아이콘 중에서 왼쪽의 적색 화살표와 같이 있는 'RST'()는 소프트웨어적으로 Reset을 처리하는 버튼이고 다음은 'Run'() 버튼, 그리고 그림은 활성화되어 있지 않지만 실행 중에는 적색으로 나타나는 X 표시의 'Stop'() 버튼이 있다. 이어서 파란색 화살표가 안쪽에 표시되어 있는 'One Step'()은 C의 한 문장 또는 어셈블리 하나의 라인을 실행한다. 그리고 화살표가 외부에 있고 한 블록씩 실행할 수 있는 'Step Over'(), C 함수의 마지막까지 실행하는 'Step Out'()이 있다. 마지막 디버그 실행 아이콘은 커서가 위치 한 곳까지 실행하는 'Run to Course Line'()이다. 디버그는 눈에 보이지 않는 부분의 알고리즘 오류를 검증하는 것이므로 꼼꼼하게 대처해야 한다. 사실 개발하는 것 보다 버그퇴치를 올바르게 해야 하는데 적재적소에 잘 사용할 수 있도록 숙련되게 익혀야 하며 보통 개발은 디버그 하는 기간을 설계하는 기간과 같은 정도로 두고 스케줄을 짜는 것이 바람직하다.

디스코를 실행하면 편집모드에서 보인 Project Window가 Register Window로 변경되는 다음과 같은 화면이 나오게 된다. 화면의 상단에는 풀다운 메뉴와 아이콘이 있고 화면의 왼쪽에는 레지스터의 상태를 알 수 있는 'Project Window'와 오른쪽에는 소스코드인 어셈블리 언어와 C언어 등을 볼 수 있는 화면이 있다. 왼쪽 하단에는 명령어를 입력하고 결과를 출력할 수 있는 'Output Window'가 있는데 Debug Command 코드를 Text로 입력하면 실행하고 처리 결과를 화면에 디스플레이 한다. 그리고 하단 가운데 있는 'Watch/Call Stack'은 소스 안에 있는 변수나 스택 영역을 살펴 볼 수 있다. 메모리 윈도우는 data, code, xdata, idata, pdata 메모리 영역을 볼 수 있는 환경을 제공한다.

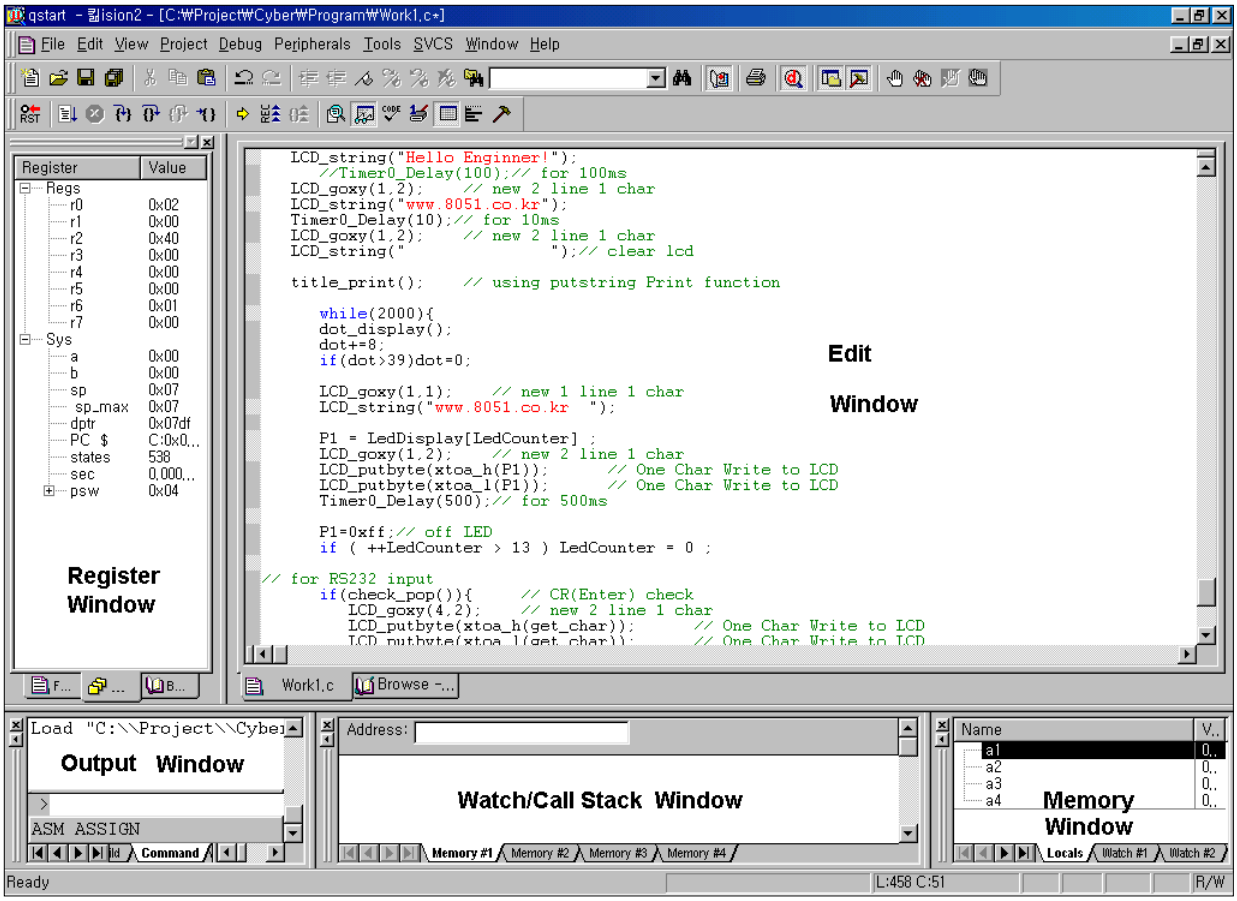

그림 33. 디스코프 실행 화면

μPSD가 탑재 되어 있는 타겟보드와 시리얼케이블을 통해 인터페이스 한 뒤 시뮬레이터를 하기위해 디스코를 운영하면 하단에 %값으로 진행되는 막대기가 현재의 접속 진행 상태를 알려 주고 접속이 양호하게 되면 디스코프를 운영하게 된다. 그러나 정상접속이 되지 않을 경우에는 다음의 적색 메시지 원도우가 뜨면서 더 이상의 작업을 할 수 없게 한다. 원인은 제일 많은 통신 속도의 불일치, μPSD에 시리얼 운영프로그램(Isd51_U1.obj)이 탑재되지 않았을 경우, 또는 헤드파일(Isd51_U1.h)이 정상적으로 연결되지 않았을 경우라고 할 수 있다.

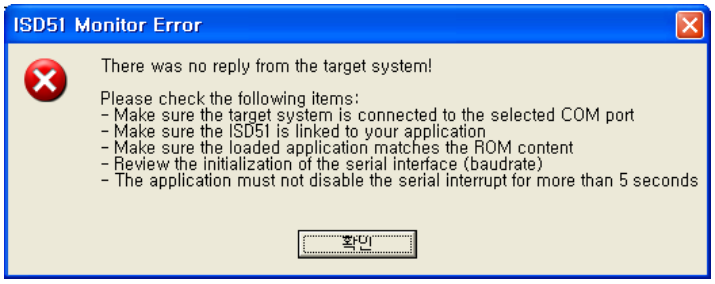

그림 34. 인터페이스 에러

디스코프의 모니터 프로그램은 소스를 하드웨어와 연계하여 실시간으로 운영하면서 디버그를 할 수

있는 환경을 제공해 주며 여러 가지 부가적으로 개발의 편리를 주기도 한다. 하지만 하다 보면 MDS(Microprocessor Development System)가 있는 것 보다 디스코프를 사용하여 개발할 때 불편한 점이 많이 있다. 그것은 코드를 수정하기 위해 디스코프를 정지해서 다시 편집환경으로 나가야 하고 수정 뒤에는 컴파일과 링크를 한 다음 다시 μPSD에 로딩해서 새로 변경된 HEX를 포팅 해야 하기 때문이다. 이런 것을 수도 없이 반복 할 것인데 인내가 필요하다. 그렇지 않고 디스코프를 운용하면 다음과 같은 메시지가 나오고 더 이상 타겟 보드와 접속이 되지 않는다.

그림 35. HEX 코드와 소스가 틀린 경우 디스플레이 문구

디스코프에서 'View/Disassembly Window' 환경은 C 언어를 컴파일 한 뒤 어셈블리 언어로 보여주는 윈도우 기능이며 어셈블리 환경에서 실행, 정지, Breakpoint 등을 설정할 수도 있다. 8051에서는 C 언어를 이용해서 알고리즘을 설계하지만 경우에 따라서는 어셈블리로 확인할 필요가 있는데 이때 이용하면 된다.

'View/Code Coverage'는 함수의 사용빈도와 호출 빈도의 Time을 이용하여 효율을 측정할 수 있다. 100%일 경우는 함수를 전부 사용한 것이 되고 그렇지 않을 경우는 효율이 떨어진다고 할 수 있다. C로 만든 소스는 함수가 코드로 되므로 코드 사이즈를 줄여야 하는 귀로에 놓이면 걱정하지 말고 이 기능을 이용하여 점검을 한 뒤 숫자 값이 적은 함수를 집중적으로 줄여나가면 목적한 다운사이징을 실현할 수 있다. 마이크로프로세서의 코드는 함수의 양이 코드의 양과 일치한다. 비공식적이지만 C 언어로 작성하여 A4 용지로 출력한 량이 300장 정도가 될 때 약 64K바이트의 용량이 된다.

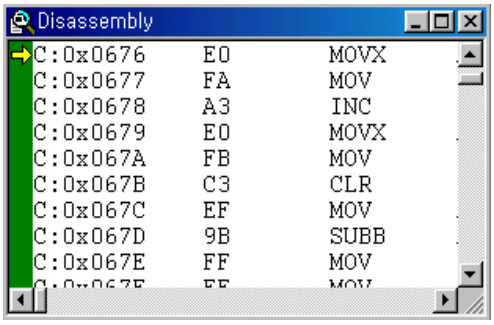

그림 36. Disassembly

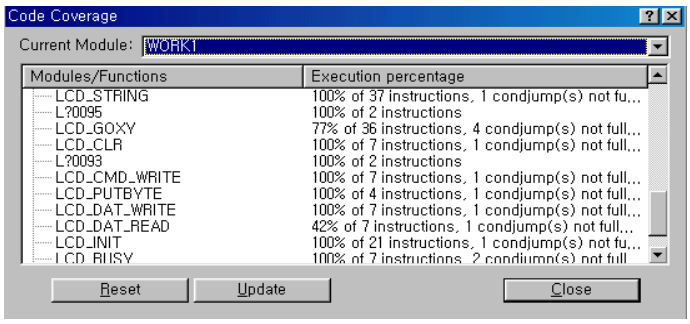

그림 37. Code Coverage

다음은 'View/Performance Analyzer Window'인데 작성된 함수를 윈도우에 등록한 뒤에(마우스의 오른쪽 버튼으로 함수 등록) 실행의 경과 측면에서 평가를 도표로 확인하는 것이다. 시간의 단위는 초이며 프로그램을 실행할 때 클릭한 함수의 경과된 시간을 나타낸다. 4개의 등록된 함수 중에서 프로그램을 실행하면 할수록 Timer0_Check 함수가 다른 함수에 비해 상대적으로 많이 사용된 것을 볼 수 있는데 8051 Timer Interrupt를 10㎳마다 한번씩 System Tick으로 동작되게 했기 때문이다. 함수의 해석을 하기위해 등록하고자 할 경우 마우스의 오른쪽 버튼을 누르면 된다.

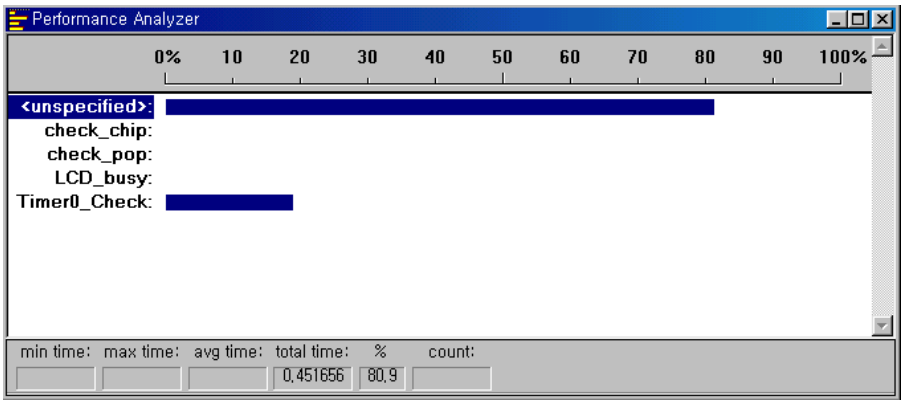

그림 38. Performance Analyzer Window

이어지는 'View/Symbol Window'는 변수, 함수 등의 심벌을 볼 수 있는 것으로 정의된 모든 소스 코드에서 사용된 Symbol을 확인할 수 있다. 'X'는 xdata 영역, 'C'는 code 영역을 나타내고 'Name' 항목은 함수나 변수의 이름, 그리고 'Type'는 변수 Type 또는 함수를 나타낼 때 사용된다. 그리고 시리얼 통신을 할 때 입/출력으로 할 수 있는 윈도우를 제공하기도 한다.

알나간　RS-232 통신할 때 주의할 점

RS-232 포트는 손쉽게 윈도우의 하이퍼터미널과 연결하여 PC와 Target Board, 혹은 PC와 PC끼리 심지어 Target Board 끼리도 통신한다. 그라운드 전위를 기준으로 RX, TX의 상대 전압의 차이를 이용 통신하므로 선로의 임피던스(교류저항) 문제로 인하여 먼 거리 통신에는 적합하지 못하다. 통신 속도를 높이기 위해 통신 케이블은 상대적으로 짧아야 하며 저속일수록 길게 할 수 있다. 특히 콘넥터를 체결할 경우 주의해야 하는데 5번핀에 전원의 그라운드가 연결되어 있어 상대측 접지(frame Ground)가 불안하여 전압 차이로 인해 PC나 Target보드의 232 드라이버 Chip이 파괴될 수도 있기 때문이다. 그래서 연결하고 결합을 해제 할 경우는 한쪽 또는 양쪽 보드의 전원을 반드시 끄고 진행해야 한다. 그리고 노이즈 문제는 통신 속도가 빠르면 빠를수록 선로가 불안할수록 증가하므로 실드를 취하는 것이 바람직하다. 이때 실드는 232 콘넥터의 양측 모두에서 프레임 그라운드 처리하지 말고 한쪽에만 새시나 전기적 그라운드에 연결하는 것이 바람직한 방법이다.

디스코프에는 모니터와 연결 되었을 때 변수와 스택의 영역도 실시간으로 볼 수 있다. 실행은 'View/Watch & Call Window'를 선택하면 Local, Watch#1, Watch#2, Call Stack의 변수를 관찰할 수 있는 윈도우가 나오게 된다. Local(지역변수)은 함수 내에서 정의된 변수를 실시간으로 관찰할 때 사용되고 Watch1, 2 윈도우는 변수이름으로 직접 등록하면 관찰이 가능하므로 전역변수를 등록하는 것이 바람직하다. 스택은 PUSH, POP으로 명령 처리된 영역을 볼 수 있다. 여기서는 인터럽트 처리에서 발생한 스택도 무리 없이 관찰이 가능하므로 Timer 등의 오류를 발견할 때 좋은 도구가 된다.

'View/Memory Window'를 선택하면 4개의 세부 창으로 data, idata, xdata, code 등의 메모리 영역

을 관찰할 수 있음으로 Address 항목에 보고자 하는 메모리의 머리글자를 입력한 뒤 콜론(':')으로 구분하고 어드레스를 넣으면 된다. 이때 Hex 값으로 입력을 원한다면 '0X'를 먼저 입력 한 뒤 4자리의 숫자를 이어서 넣으면 해당 어드레스 이후의 영역이 화면 가득 차게 나오게 된다. 디스플레이 단위를 변경하고자 할 경우는 메모리 윈도우에서 마우스의 오른쪽 버튼을 눌러 Char, ASCII, Double 등으로 선택하면 Display 형식이 변경된다. Memory#1~#4까지는 서로 다른 메모리 영역의 값을 표시할 수 있으며 예로 윈도우의 Address 항목에 C:0x300 했다면 코드 영역의 300번지 이후를 볼 수 있다. C는 코드영역, X는 xdata, I는 idata 영역을 나타내며 프로그램 실행 중에 실시간으로 변경되어 특정 영역에서 변경되는지의 여부도 알 수 있다.

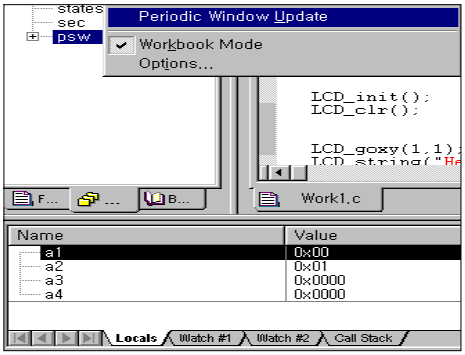

그림 39. Watch Window

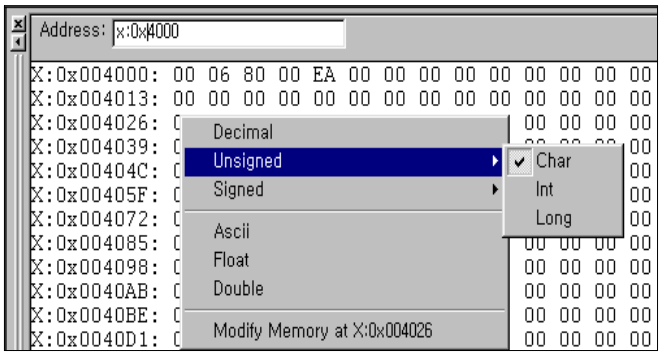

그림 40. Memory Window

디스코프는 외부 장치의 설정상태와 결과값을 볼 수 있고 시뮬레이터 중간에 특정 값을 변경할 수 있는 기능이 있다. 메뉴 'Peripheral' 부분에 보면 인터럽트의 실행, 우선순위 설정, 벡터 등을 볼 수 있는 'Interrupt' 항목과 Port0~3까지의 포트의 입력/출력 값과 비트 선택이 가능한 'IO-Ports'가 있다. 포트확인에서 'V' 표시는 1을 나타내고 없으면 0을 나타낸다. 'Serial'은 Serial0과 Serial1의 설정상태를 나타냄으로 Serial Mode와 SCON, SMOD 등의 값을 확인 할 수 있음으로 한눈에 보는 편리함이 있다. 'Timer/Count'는 타이머 혹은 카운터로 사용하는 Multiplex 개념인 8051의 포트 설정상태와 해당되는 레지스터의 설정값을 보여주며 프로그램에서 제어되어 있지 않더라도 디버그 상황에서 직접 값을 변경할 수 있다. 또 마이크로프로세서의 경우에 따라 있을 수 있는 'Watchdog' Timer 설정도 볼 수 있다. 마지막의 'Clock Control'은 8051 MPU의 특별한 모드를 반영한 것인데 외부 클락의 분주 설정을 확인해 볼 수 있다.

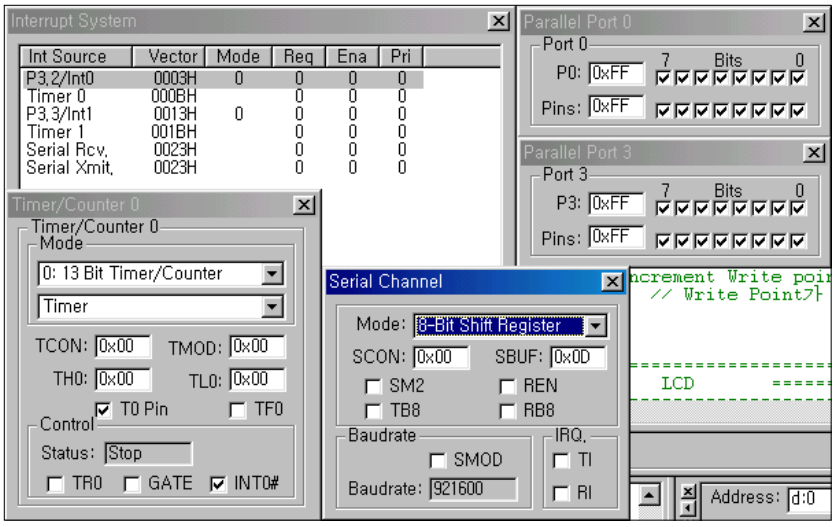

그림 41. 8051 외부 장치의 설정 윈도우

알나간 PAL, GAL, PLA

PAL(Programmable Array Logic)은 미국 AMD사(옛날에는 MMI)의 PLD에 붙인 상표 이름으로 여러 개의 로직이 AND-OR Array로 구성되어 있는 구조로 사용자가 필요에 의해 로직의 배선을 변경하여 사용한다. GAL(Generic Array Logic)은 미국 Lattice사가 개발한 전기적으로 지우고 쓰기가 자유로운 PLD(Programmable Logic Device)로 PAL과 호환성이 대부분 고려되어 있다. 또 OR-AND Array로 AND 어레이를 미리 고정한 구조로 되어 있는 PAL을 PLA(Programmable Logic Array)라고 한다.

---

5. μPSD 응용

다음은 앞에서 익힌 하나하나의 정리를 설계와 실제의 Application에 적용하기 위해 실제로 설계와 코딩을 해 보는 순서다. 그러기 위해서는 μPSD에 대해 알고 PSDsoft 그리고 컴파일러를 익힌 다음 디자인 하고 코딩하여 동작 결과를 얻는 숙달의 지식 습득이 있어야 한다. 이렇게 하기 위해서는 우선 구현하고자 하는 보드를 다음의 주어진 회로처럼 설계하고 제작하자. μPSD의 핀에 대한 이름과 기능을 할당하고 알고리즘을 C 언어를 이용하여 프로그램 컴파일을 한 다음 HEX 코드를 만들고 나서 타겟 보드에 로딩, 실행하고 점검하여 동작이 안나오는 부분은 하드웨어 부분과 Application 부분 둘 다를 디버깅해야 한다.

이런 일련의 절차와 개발 Tip을 예제로 들어 설명하고자 했는데 회로는 복잡하지 않다. μPSD3233(5V,128K Main, 32K Secondary Memory)을 80핀을 이용하여 DIP 스위치를 Port4에 연결하고 Graphic LCD를 PA, PB 포트에 그리고 LED를 P1에 인터페이스 한 다음 동작 알고리즘을 만들어 Target Board에 이식하고 dScope와 시리얼 케이블로 연결, 실시간으로 디버깅했다. 그리고 사용하지 않는 나머지 핀은 Test Point로 두어 나중에 연결하거나 계측기를 사용하여 파형으로 동작을 관찰 할 수 있도록 했다.

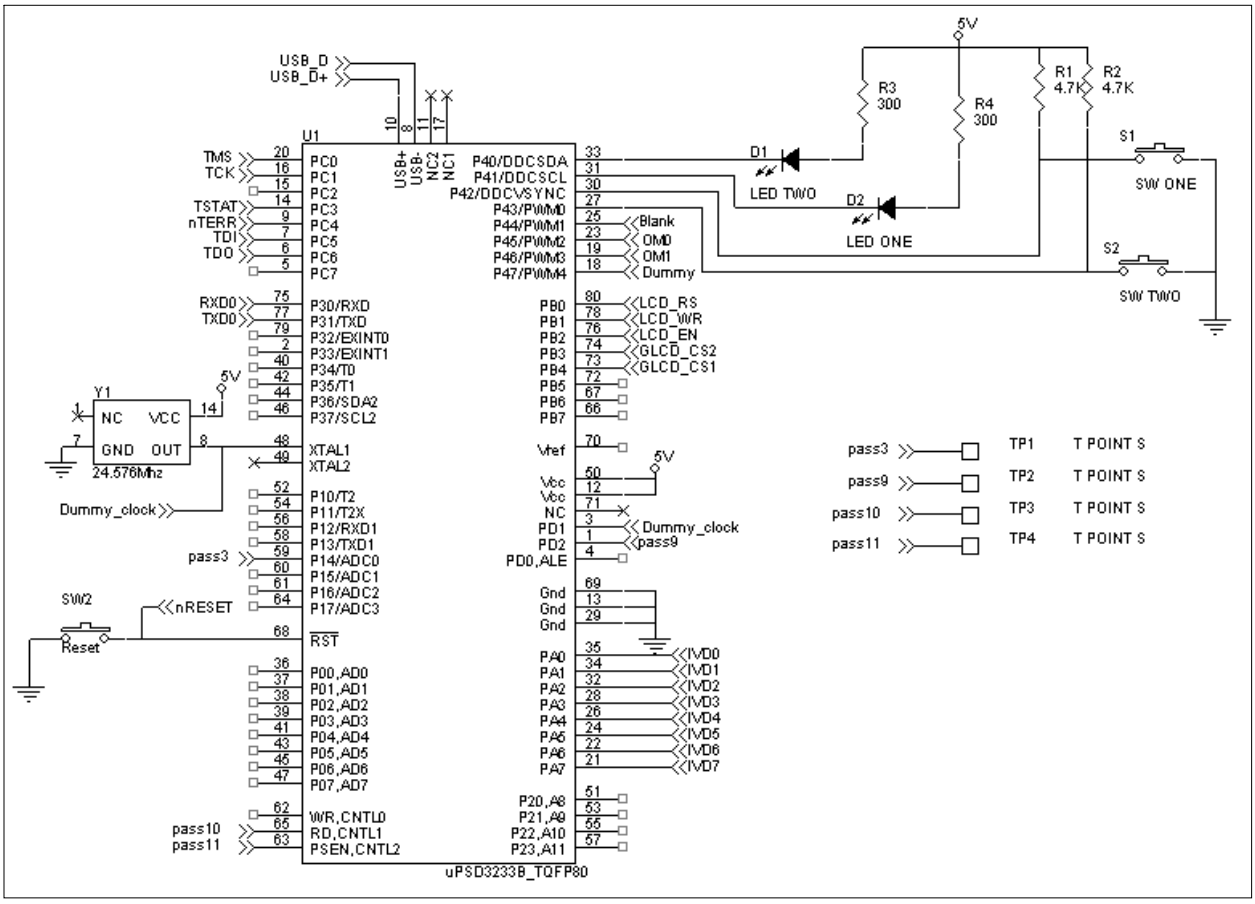

그림 42. μPSD 회로도1

알다시피 μPSD는 내부에 메모리와 하드웨어에 관련된 PLD 영역이 있고 디코드를 구성하게 되어 있으므로 소프트웨어를 Link 시키기 위해서는 어드레스를 프로그램에 등록해 주어야 한다. 하드웨어에 관련된 부분은 PSDsoft와 함께 PLD 항목에서 설명했고 LED/LCD를 구동하는 알고리즘은 CODE 항목에서 소스와 함께 설명했다. 그리고 Target Board와 KEIL Compiler의 실시간 Interface는 Debug에서 그 상태와 결과를 사진과 화면 캡처를 통해 보여 주었다. 아무쪼록 Tool을 익히고 μPSD를 배우려고 하는 관심 있는 엔지니어에게 도움이 있기를 바란다.

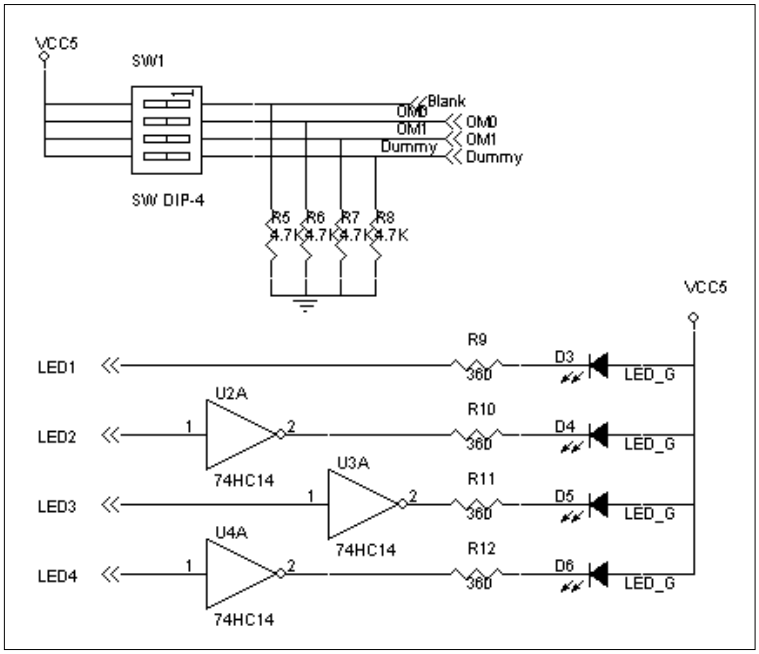

그림 43. I/O 장치 회로도2

프로세서의 동작 상태를 알기 위해 G-LCD 이외에 LED 출력과 딥스위치 입력을 연결했다. 자체적으로는 아무 의미 없지만 Firmware를 작성하다 보면 필요할 때가 있다. DIP 스위치의 Pull Down 저항은 4.7K로 했는데 입력 전류가 5V에서 0.6㎃ 이상 되어야 하기 때문이다.

5.1 PLD

PSDsoft 프로그램을 Quick Start에서 다룬 대로 프로젝트를 만들고 회로도를 보고 핀의 이름을 할당하자. 여기서는 프로젝트 이름을 'scopepld'라고 했는데 'Define PSD Pin/Node Function' 과정을 에러 없이 처리하면 scopepld.sum 파일이 생성된다. 실험한 것과 같은 결과를 얻기 위해서는 다음과 같이 sum파일의 내용처럼 핀과 Chip 환경을 설정하면 된다. 중복을 피하기 위해서 당연한 출력결과 값은 빼고 꼭 있어야 할 Text만 보였다.

PSDsoft Express Version 7.91
PROJECT : scopepld DEVICE : uPSD3233B MCU/DSP : uPSD32XX
Pin Definitions:; I/O 모드로 대부분 설정했다.
==========

Pin	Signal	Pin Type
pa7	LCD_d7	Peripheral I/O mode
중략...		
pa0	LCD_d0	Peripheral I/O mode
pb4	GLCD_CS2	GP I/O mode
pb3	GLCD_CS1	GP I/O mode
pb2	LCD_EN	GP I/O mode
pb1	LCD_WR	GP I/O mode
pb0	LCD_RS	GP I/O mode
tdo	tdo	Dedicated JTAG - TDO
tdi	tdi	Dedicated JTAG - TDI
pc4	_terr	Dedicated JTAG - /TERR
pc3	tstat	Dedicated JTAG - TSTAT
pc2	vstby	SRAM standby voltage input
tck	tck	Dedicated JTAG - TCK
tms	tms	Dedicated JTAG - TMS
p3.1	USART1_Txd	UART1 TxD
p3.0	USART1_Rxd	UART1 RxD

```
User defined nodes:; PLD부분에서 추가했다.
====================
특별히 정의된 것 없음

Page Register settings:
=======================
pgr0 is not used
중략...
pgr7 is not used

Equations:; 임의대로 변경해도 되지만 C Source에서도 동일하게 해야 한다.
==========
rs0 = ((address >= ^h2000) & (address <= ^h3FFF));
csiop = ((address >= ^h0200) & (address <= ^h02FF));
fs0 = ((address >= ^h8000) & (address <= ^hAFFF));
csboot0 = ((address >= ^h0000) & (address <= ^h1FFF));
csboot1 = ((address >= ^h2000) & (address <= ^h3FFF));
csboot2 = ((address >= ^h4000) & (address <= ^h5FFF));
csboot3 = ((address >= ^h6000) & (address <= ^h7FFF));
psel0 = ((address >= ^h0300) & (address <= ^h03FF) & (_psen));
```

여기까지는 평상시 프로그램 사용법대로 하면 되는데 PLD 부분을 아벨 언어를 이용하여 하드웨어를 구현하기 위해서는 PSDsoft 메뉴의 Project/Preference의 ABEL 언어 부분을 'V' 하여 활성화 시킨 다음 편집 항목이 보일 수 있도록 해야 한다. 또는 프로젝트 파일이름과 확장자 abl을 집적 수정해도 된다. 목적은 PLD의 로직을 이용하여 24.576㎒ 입력의 크리스탈 정형파 카운터를 사용, 12.288㎒ 구형파로 출력하기 위한 것이다. 이럴 때 예전 같으면 카운트를 이용하거나 CPLD, 또는 타이머를 이용해서 구현했다. μPSD의 핀에(여기서는 PD1) 하드웨어적으로 클럭을 입력하고 주파수를 절반으로 한 다음 출력(여기서는 PD2)할 수 있는 핀을 설정한 후 소프트웨어적으로 아벨 언어로 코딩해야 1/2 Count가 된다.

아벨 언어로 되어 있는 파일은 전부 설계자가 프로그램 할 수 있는 것은 아니다 PSDsoft 프로그램은 이 파일을 이용하여 핀 정의와 PLD 부분을 설정하게 되어 있는데 설계자가 마음대로 할 수 있는 부분 이외는 변경하지 말고 사용자 부분만 다음과 같이 입력한다. 기능은 언제 이용할지는 모르지만 저자는 이런 기능이 꼭 필요해서 ST사의 엔지니어에게 도움을 받아 다음과 같이 정의하여 카운트를 구현했다.

```
// Begin user preserved declarations ====================
clk_24m_i PIN 3; "MCU CLK 24.576㎒ input, Oscillator 클럭 입력/이름은 알아서
clk_12m_o PIN 1; "divided CLK 12.288㎒ output, 나중에 다른 I/O에서 이용하기 위해
div0 node istype 'reg_t'; "노드 정의
// End user preserved declarations ====================
// Begin user preserved equations ====================
div0.clk = clk_24m_i; "Clock을 나타낸다.
div0.t = 1; "T F/F를 나타낸다. 도움말에 있음
clk_12m_o = div0.fb; "Feed Back
// End user preserved equations ====================
```

TEXT : PLD Programming

다음은 'Fit Design to Silicon' 항목을 클릭하여 에러 점검과 최종 디자인된 하드웨어 칩에 이식할 준비를 하자. 에러가 발생하거나 관련 정보의 리스트는 하단의 윈도우 또는 *.PLG 파일에 기록 되고 핀의 구성, 방정식, Cell 사용 등에 대한 Fit 정보는 *.FRP 파일에 Text로 기입된다.

구성된 하드웨어로 만들어진 Application 프로그램 HEX 파일을 완성 했다면 'Merge MCU Firmware with PSD' 항목을 클릭하여 CSBOOT0 영역에 HEX 파일이름과 경로를 입력한 다음 'OK' 하자. 다음 PC의 프린터 포트와 Flashlink를 연결한 뒤 JTAG 14핀 끝을 Target과 올바르게 연결하고 Target의 전원을 ON 'Execute' 버튼을 클릭하면 된다. 여기까지 말과 글은 쉽지만 나름대로 잘 안되는 부분은 열심히 수정해보고 그래도 안 되면 저자에게 메일을 보내거나 ST사의 엔지니어에게 문의 하면 되겠다. 다음의 사진은 실험에 이용되었던 Target의 동작 상태와 출력 파형을 나타낸다.

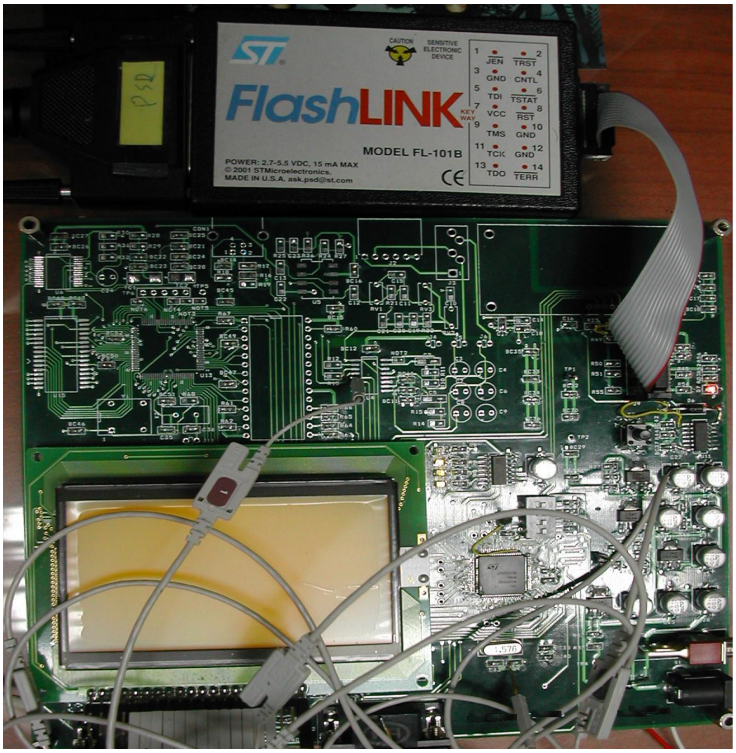

그림 46. 실험 Target Board

사진의 상단에 JTAG 케이블이 보이고 로직 계측기에 연결된 Target보드가 보인다. LCD의 오른쪽 상단에 SMD LED가 4개(작아서 보이지 않음) 회로도처럼 연결되어 있고 그 바로 밑에 μPSD가 붙어 있다. 수집된 파형은 실험의 결과를 알기 위해 Agilent Logic Analyzer 1672G 모델을 사용하여 화면 카피한 것이다. [그림:Target Board 출력파형] 상단의 신호는 System Clock(24.576㎒)이고 밑에는 2분주(12.288㎒) 한 결과이다. 주기는 약 80㎱이며 성공적인 결과가 나왔다. P1_3은 계측기의 트리거를 잡기위한 기준이 되기도 하는데 메인 프로그램에서 ON 상태로 한 다음 Port A(PA_POT all 신호)에 A1, B2, C3의 16진 값을 차례로 입력했다. 나머지는 G-LCD의 신호를 나타낸다.

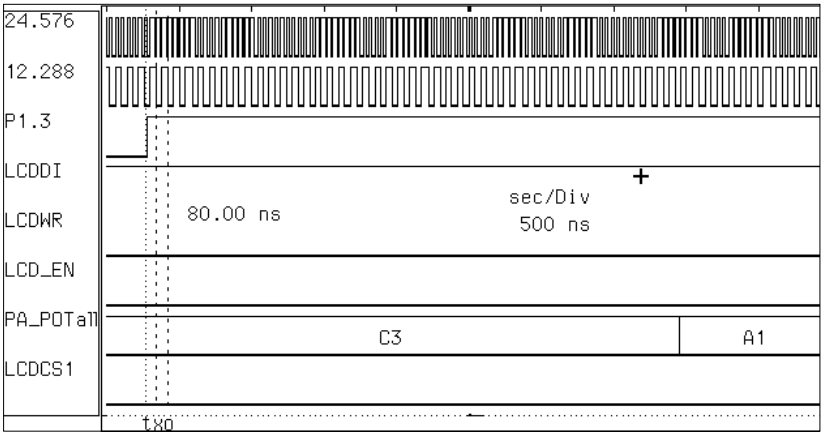

그림 47. Target Board 출력파형

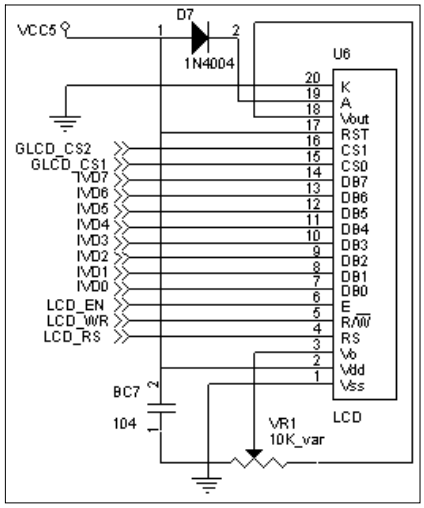

그림 48. G-LCD 회로도5

5.2 Code

Application 코딩을 위해 KEIL 컴파일러 V7.0을 이용하여 이미 구성된 PLD의 하드웨어 구성으로 설정한 뒤 Port 제어와 200~2FF(CSIOP 영역) xdata 영역에 구조체로 정의한 레지스터에 변수값을 출력 하는 Application을 만들었다. 여기서는 Basic 설계 방법과 Source Code 그리고 디버깅 할 때 아주 유용한 Target과 dScope가 인터페이스 하는 방법에 대해 학습한다.

카일(www.keil.com) 컴파일러 통합 환경에서 프로젝트의 이름을 부여하여 새로 만들도록 하자. 특히 μPSD 또는 하드웨어적으로 사전에 설정된 값을 'Option for target' 항목에서 올바로 입력하자. 타겟의 클럭값과 Csboot 영역(Code 영역)의 플래시 메모리 시작 번지와 크기 그리고 Csiop 영역의 memory mapped I/O xdata 공간을 정확히 입력한다. Code 영역은 빈 공란으로 두면 0번지부터 FFFFh 번지까지 자동 설정된다.

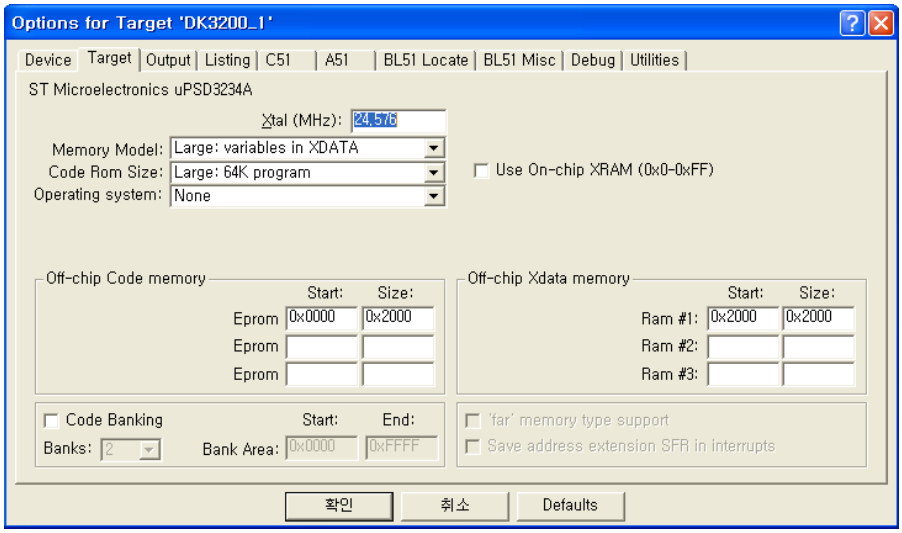

그림 49. Option for Target

'Memory Model' 항목은 좀 조심해야 되는데 이곳을 그림처럼 Large로 했을 경우에 변수정의를 하면 자동으로 컴파일러가 알아서 xdata 메모리 영역으로 할당하게 된다. 만약 외부 메모리를 연결하지 않았을 경우는 'C51/Code Optimization'에서 설정값을 0으로 해야 한다. 이렇게 되면 소스 코드에서 xdata 키워드를 사용했을 경우만 그곳으로 배치하게 된다. 반대로 'Memory Model'을 Small로 하면 Optimization을 0으로 두지 않아도 자동으로 외부 xdata 영역에 변수를 위치시키지 않게 되어 프로그래머로 하여금 덜 신경 쓰게 만들어 준다. 이때도 xdata 키워드를 사용했을 경우는 예외가 된다.

그리고 만약 Flashlink를 이용하여 Target Board와 실시간으로 정보를 주고받으면서 디버깅하고자 한다면 'Option for Target/Debug' 항목에서 다음처럼 설정해야 하고 함수와 타이머에 관련된 소스를 프로그램에 추가해야 한다. 'KEIL ISD51 In-System Debugger'로 선택하고 Startup Code를 불러오게 하면 준비는 다 끝난 것이다. 옆에 있는 'Go till main' 은 디스코프를 시작 할 때 메인 프로그램을 바로 실행 하므로 Break를 걸어 놓지 않으면 자동으로 열심히 실행하게 된다. 'Settings'는 Default 기본값으로 COM1 Port 그리고 통신 속도는 19,200Bps로 설정 되었는데 Com Port 이외에는 초보일 경우 변경하지 말자.

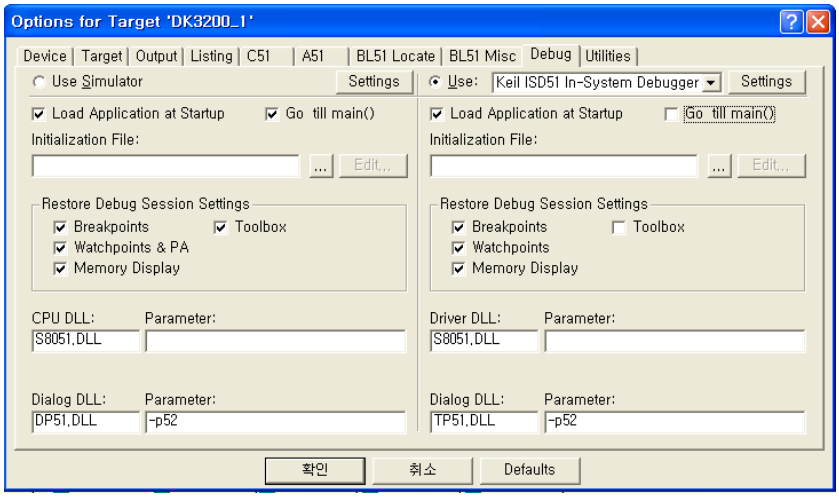

그림 50. Flashlink를 이용한 실시간 디버그 설정

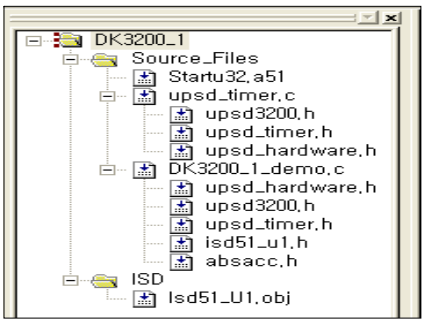

그림 51. 프로젝트 구성

이제는 소스 코드로 들어가 보자. 우선 프로젝트 구성은 만들어서 등록 시킨 C 소스 코드 두 개와 컴파일러에서 만들어준 어셈블리 코드 startu32.a51 파일과 그리고 또 컴파일러에서 제공된 오브젝트 코드 isd51_u1.obj(In-System 디버깅)로 구성되어 있다. 이 모든 프로젝트에 등록된 파일은 컴파일러 측면에서는 모두 소스코드가 된다. 헤드파일 한번 눈여겨보자. 이미 제공된 것과 추가로 저자가 작성한 것이 있다.

```
upsd_timer.c
/* ST사에서 Demo로 제공해준 파일의 일부를 저자의 Target Board에 맞게 수정했다.*/
/* Timer 설정과 인터럽트에 대한 사항이 들어있다. */
#include "upsd3200.h"
#include "upsd_timer.h"
#include "upsd_hardware.h"

static unsigned int idata timer0_tick;  // 2바이트 크기 변수를 7F~FF 내부 RAM 영역 배치
unsigned int timer0_value; // 컴파일러가 알아서 배치 하지만 주로 0~7F 영역에 놓임
static void timer0_isr (void) interrupt 1 using 1{// Bank 1을 사용한 인터럽트 정의
    TR0 = 0;             /* stop timer 0 */
    TL0 = (timer0_value & 0x00FF);   // 10ms 타이머 값, 하위, Auto reload 안됨
    TH0 = (timer0_value >> 8);       // 상위
    TR0 = 1;                            // 타이머 동작 시작
    timer0_tick++;       // Increment global var timer_tick (number of 10ms ticks)
}

void timer0_init (void){  // timer0을 인터럽트로 사용하기 위한 초기 루틴
    EA = 0;          /* disable interrupts */
    timer0_tick = 0;
    TR0 = 0;         /* stop timer 0 */
    TMOD &= 0xF0;         /* clear timer 0 mode bits - bottom 4 bits */
    TMOD |= 0x01;         /* put timer 0 into 16-bit no Prescaler */
    timer0_value = 0x10000 - ( ((FREQ_OSC * 5L) / 6L) - 17L);// 10ms(100hz)주기
    // FREQ_OSC는 24576㎑
    TL0 = (timer0_value & 0x00FF);   TH0 = (timer0_value >> 8);
    PT0 = 1;         // 우선순위 높다
    ET0 = 1;         /* enable timer 0 interrupt */
    TR0 = 1;         /* start timer 0 */
    EA = 1;          /* enable interrupts */
}
```

Source : upsd_time.c

회로도는 시스템 클락과 JTAG 포트의 구성을 나타낸다. Flashlink를 통해 프로그램을 Target에 로딩할 때 D8의 LED가 ON 된다. 저항값에 주의하자.

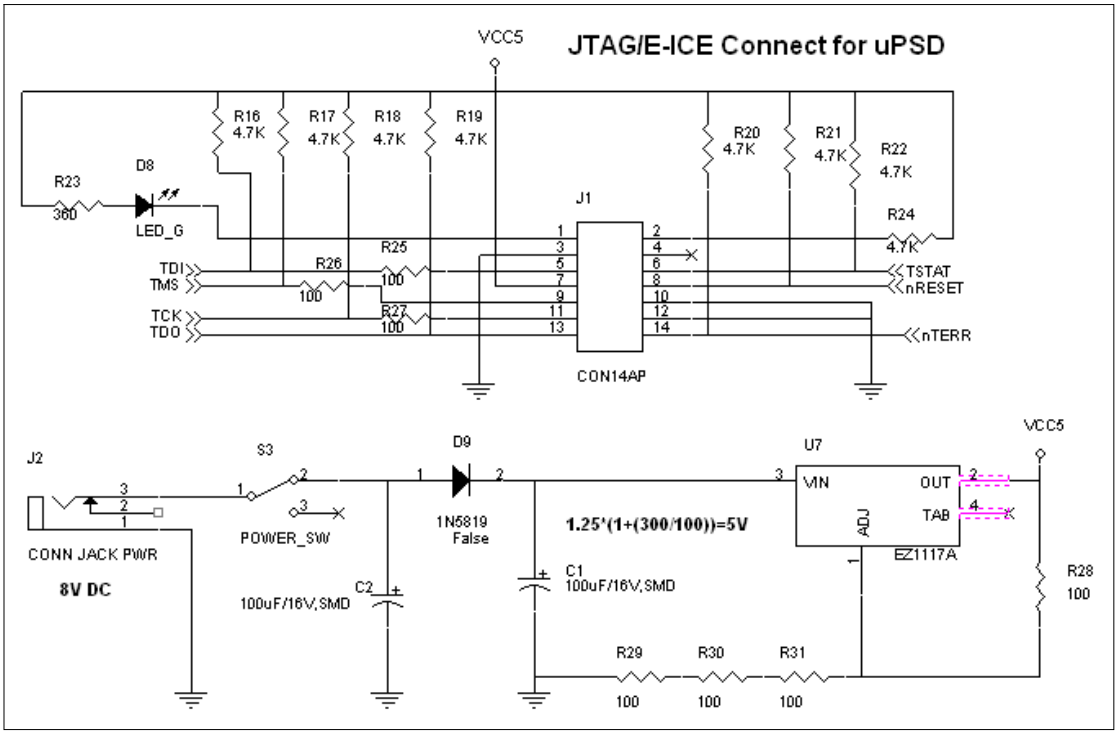

그림 52. JTAG/Power 회로도4

```
unsigned int timer0_count (void){// 후반에서 Delay로 이용하기 위한 함수
  unsigned int t;

  EA = 0;       // 인터럽트를 잠시 멈춤
  t = timer0_tick;
  EA = 1;
  return(t);
}

void timer0_delay (unsigned int count){// 보다 정확한 Delay 이지만 다른 루틴이 멈추는 것이
흠
unsigned int start_count;
start_count = timer0_count(); /* get the start count */
while ((timer0_count() - start_count) <= count);    /* wait for count "ticks" */
}

void delay_10ms() {
    timer0_delay(1);
}

void delay_1sec(void){// 다른 시간은 만들기 나름
    timer0_delay(100);
}
```

Source : Utility

Main 함수는 DK3200의 Demo 예제에서 불러와 수정을 거친 뒤 사용했다. 그래서 파일이름도 아 3200_1_demo.C이다. 이 프로그램의 기능은 별것 아니지만 초기화 하고 설정하는 기본 방법에 대해서는 잘 알려 준다. 특히 타겟과 디버깅하기 위해서는 어떤 함수로 어떻게 사용되는 지를 잘 설명해 주므로 MDS가 없거나 초기 개발에 임하고 시행착오를 줄이고자 하는 엔지니어라면 반드시 숙지가 필요하다.

```
#include "upsd_hardware.h"    // environment hardware specific defines
#include "upsd3200.h"         // ST사에서 제공된다. 기본 설정값 이외에 추가된 것이 있음
#include "upsd_timer.h"
#include "ISD51_U1.h"       // 타겟과 디버깅하기 위해 반드시 첨부

#define LCD1IR   PSD8xx_reg.DATAOUT_B =0x08 // 0000 1000
#define LCDEN    PSD8xx_reg.DATAOUT_B |=0x04  //µPSD323x Data sheet 38쪽 참조
#define LCDDIS   PSD8xx_reg.DATAOUT_B &= 0xFB    //1111 1011

#define LCD2IR PSD8xx_reg.DATAOUT_B=0x10// 0001 0000
#define LCD1DR PSD8xx_reg.DATAOUT_B=0x09// 0000 1001
#define LCD2DR PSD8xx_reg.DATAOUT_B=0x11// 0001 0001

#define uchar unsigned char  // byte와 동일

#define XDOT  64 // for LCD
#define YDOT  128   // for LCD
```

Source : main

```
void delay(unsigned int i);
void write_data1(unsigned char val);
void write_data2(unsigned char val);
void pixel_x_y(unsigned char x, unsigned char y, unsigned char val);
void lcd_clear(void);
void display_on(void);
void Zaddress(unsigned char dsl);
void put_char16(unsigned char x, unsigned char y, unsigned char value);
void put_string16(unsigned char x, unsigned char y, unsigned char *str);
void put_char8(unsigned char x, unsigned char y, unsigned char value);
void put_string8(unsigned char x, unsigned char y, unsigned char *str);

PSD_REGS xdata PSD8xx_reg _at_ PSD_REG_ADDR;  //"csiop" 영역의 구조체 선언, xdata 영역
unsigned char idata  msg_buff[20]; // 여기서는 의미 없음
unsigned char idata i;
unsigned char dip;

unsigned char idata d;

char code font123_16[10][22] = {// Font
0x0f,0xe0,0x3f,0xf8,0x70,0x1c,0xc0,0x06,0x80,0x02,0x80,//0
0x02,0xc0,0x06,0x70,0x1c,0x3f,0xf8,0x0f,0xe0,0x00,0x00,
0x00,0x00,0x00,0x00,0x20,0x00,0x20,0x00,0xff,0xfe,0xff,//1
중략...
0x3e,0x18,0x7f,0x1c,0xc1,0x86,0x80,0x82,0x80,0x82,0x80,//9
0x82,0xc1,0x86,0x7f,0xfc,0x3f,0xf8,0x00,0x00,0x00,0x00
};
```

Source : main_계속

```
char code font_ABC16[26][22]={
0x00, 0x0e, 0x00, 0xfe, 0x07, 0xf0, 0x3f, 0x40, 0xf8, 0x40, 0xf8, //A
0x40, 0x3f, 0x40, 0x07, 0xf0, 0x00, 0xfe, 0x00, 0x0e, 0x00, 0x00,
중략...
0x80, 0x02, 0x80, 0x0e, 0x80, 0x3e, 0x80, 0xf2, 0x83, 0xc2, 0x8f, //Z
0x02, 0xbc, 0x02, 0xf0, 0x02, 0xc0, 0x02, 0x80, 0x02, 0x00, 0x00
};
char code font8[100][7]={
0x00,0x00,0x00,0x00,0x00,0x00,0x00,
중략...
0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x7c,0x82,0x82,0x82,0x7c,0x00,//0
중략...
0x82,0x86,0x8a,0x92,0xa2,0xc2,0x00,//Z
};
```

Source : main_계속

```
void main (void)  {
unsigned int timer2_baud;
unsigned char idata idataval; // idata 실험용
unsigned char data dataval;   // 내부 RAM 영역 실험용
unsigned char xdata xdataval; // xdata 실험용

// UART 설정, Target과 디버깅 통신 속도 설정
    T2CON   = 0x34;       // Use Timer 2 as baud rate generator
   PCON  |= 0x0C;         // Set UART2 to user timer2 baud rate
   timer2_baud = (65536L - ( (FREQ_OSC * 125L) / (4L * 19200L)));
   RCAP2L = (timer2_baud & 0x00FF);// 통신 속도 19200bps
    RCAP2H = (timer2_baud >> 8);
    SCON   = 0x50;        // enable first serial UART & receiver
   SCON2  = 0x50;         // enable 2nd UARTt
    EA = 1;               // Enable global interrupt flag

   timer0_init();         // initialize timer0 interrupt

PSD8xx_reg.OUTENABLE_A=0xFF;// enable PA;
PSD8xx_reg.DIRECTION_A=0xFF;// output PA
PSD8xx_reg.OUTENABLE_B=0xFF;// enable PB;
PSD8xx_reg.DIRECTION_B=0xFF;// output PA

Zaddress(0);// G-LCD 초기화
WDKEY=0x55;// Disable WDT
display_on();// On Display
lcd_clear();// must be initial
PSD8xx_reg.OMCMASK_AB = 0xF0;  // Mask off upper nibble of Output MacroCell register.
                               // This allows writing a byte to OMC register to load
                               // 4-bit initial count to down-counter in PLD without
                               // disturbing the upper 4-bits of OMC register
PSD8xx_reg.OMC_AB = 0x08;  // Load initial count of eight into down-counter in PLD.
                           // This 4-bit counter will pulse pin PB4 each time 8 counts of
                           // 8032 ALE pulses occur per logic equations. See App Note AN1560.
while (TRUE){  // Main demo loop, 무한 루프
    ISDcheck();       // 이 루틴이 있어야 In-system 디버깅이 가능하다.
    dip=P4;     // DIP 스위치 검사
    dip &=0x07;    // 마스크 처리

    P1_3=OFF;// for trigger
    idataval=0xA1;
    dataval=0xB2;
    xdataval=0xC3;
    P1_3=ON;// for trigger
    PSD8xx_reg.DATAOUT_A=idataval;
    PSD8xx_reg.DATAOUT_A=dataval;
    PSD8xx_reg.DATAOUT_A=xdataval;
}
}  // End main
```

Source : main_계속

```
void delay(unsigned int i) {//시간 지연
    while(i--);
}

void write_data1(unsigned char val){
    LCD1DR;
    LCDEN;
    delay(1);
    PSD8xx_reg.DATAOUT_A=val;
    LCDDIS;
}

void write_data2(uchar val){
    LCD2DR;
    LCDEN;
    delay(1);
    PSD8xx_reg.DATAOUT_A=val;
    LCDDIS;
}
```

Source : main_계속

```
void pixel_x_y(unsigned char x, unsigned char y, uchar val){
    if(y>127) y = 127;
    if(x>7) x = 7;
    if (y <= 63)    {
        LCD1IR;    //CS1영역에서 x좌표 설정
        LCDEN;
        delay(1);
        PSD8xx_reg.DATAOUT_A=(0xb8 + x);
        LCDDIS;
        LCD1IR;    //CS1영역에서 y좌표 설정
        LCDEN;
        delay(1);
        PSD8xx_reg.DATAOUT_A=(0x40 + y);
        LCDDIS;
        write_data1(val);        //CS1영역의 설정된 좌표에 DATA쓰기
    }
    if (y >63)  {
        LCD2IR;    //CS2영역에서 x좌표 설정
        LCDEN;
        delay(1);
        PSD8xx_reg.DATAOUT_A=(0xb8 + x);
        LCDDIS;
        LCD2IR;    //CS2영역에서 y좌표 설정
        LCDEN;
        delay(1);
        PSD8xx_reg.DATAOUT_A=(0x40 + y-64);
        LCDDIS;
        write_data2(val);        //CS1영역의 설정된 좌표에 DATA쓰기
    }
}

void display_on(void){// for initial
    LCD2IR;    //CS2영역을 Display ON
    LCDEN;
    delay(1);
    PSD8xx_reg.DATAOUT_A=0x3f;
    LCDDIS;
    delay(1);        //명령처리 시간동안 대기
    LCD1IR;    //CS1영역을 Display ON
    LCDEN;
    delay(1);
    PSD8xx_reg.DATAOUT_A=0x3f;
    LCDDIS;
    delay(1);        //명령처리 시간동안 대기
}
```

Source : main_계속

```
void Zaddress(uchar dsl){// for initial, typical 0, other value is 0,1,2,3
    char i;
    i = 0xc0 + dsl;
    LCD2IR;       //CS2영역의 Zaddress설정
    LCDEN;
    delay(1);
    PSD8xx_reg.DATAOUT_A=i;
    LCDDIS;
    delay(1);       //명령처리 시간동안 대기
    LCD1IR;       //CS1영역의 Zaddress설정
    LCDEN;
    delay(1);
    PSD8xx_reg.DATAOUT_A=i;
    LCDDIS;
    delay(1);       //명령처리 시간동안 대기
}

void lcd_clear(void){// must be initial
    int i, j;
    for (i=0;i<YDOT;i++) //i->y좌표
        for (j=0;j<8;j++) pixel_x_y(j,i,0x00); //j->x좌표 ,DATA는 모두 0
}

void put_char8(uchar x, uchar y, uchar value){
    char i;             //for 문을 위한 변수
    char *tmp_ptr;        //font의 data가 있는 곳을 알려주기 위한 변수

    x = 127-x;        //font의 좌표는 LCD내부의 좌표와는 다르기 때문에 변환
    value -= 0x20;       //font data의 index를 위한 변환
    tmp_ptr = font8[value];

    for(i=0; i<7; i++) //설정된 좌표에 font를 출력
        pixel_x_y(y , x-i,*tmp_ptr++);
}

void put_string8(uchar x, uchar y, uchar *str){
    while(*str){    //문자열에 NULL값이 있을때 까지 한 문자씩 출력
        put_char8(x ,y,*str++);
        x+=7;
    }
}
```

Source : main_계속

5.3 Debug

저자는 개발자의 한사람으로 설계보다 힘든 것이 디버깅 이였다. 처음 설계 계획을 내 자신의 과신으로 시간에 맞추다 보니 디버깅에서는 시간이 부족하여 밤늦게까지 하는 경우가 많았고 촉박한 마음에 졸작으로 건너갈 때가 있었다. 설계한 만큼의 시간과 동일하게 디버깅에 할당하는 요즈음에는 이런 벼락치기는 좀 줄어들었지만 새로운 프로그램이나 환경을 익히는데 예전보다 좀더 많은 시간을 두게 된다. PC의 발달과 소프트웨어 그리고 마이크로프로세서의 업그레이드는 Tool을 간단하고 자동으로 알아서 해주는 것으로 진화되지 못했고 익히는데 쉽지 않은 환경을 개발자에게 안겨 주기 때문이다.

µPSD 3233이 탑재된 실험용 PCB를 저자가 처음 만들어 프로그램을 이식했을 때 정상 동작이 되다가 보드가 갑자기 동작을 멈추는 일이 생겼었다. 이때 피시비 기판의 패턴을 터치해야 정상 동작이 되는 마치 냉땜이 되었을 때의 오동작을 되풀이 했는데 디버깅이 무척 힘이든 기억이 있다. ST사의 엔지니어에게 묻고 매뉴얼을 확인해 보니 USB의 '-' 단자에 4.7K 저항으로 Full up을 해 주어야 Floating 되지 않아 정상동작 된다는 것을 알았는데 USB를 실제로 사용하지 않아도 회로도처럼 연결해 줘야 함을 잊지 말자. 그리고 WDT(Watch Dog Timer)를 설정해 주지 않아 되풀이되는 Reset으로 처음에는 어리둥절하기도 했다.

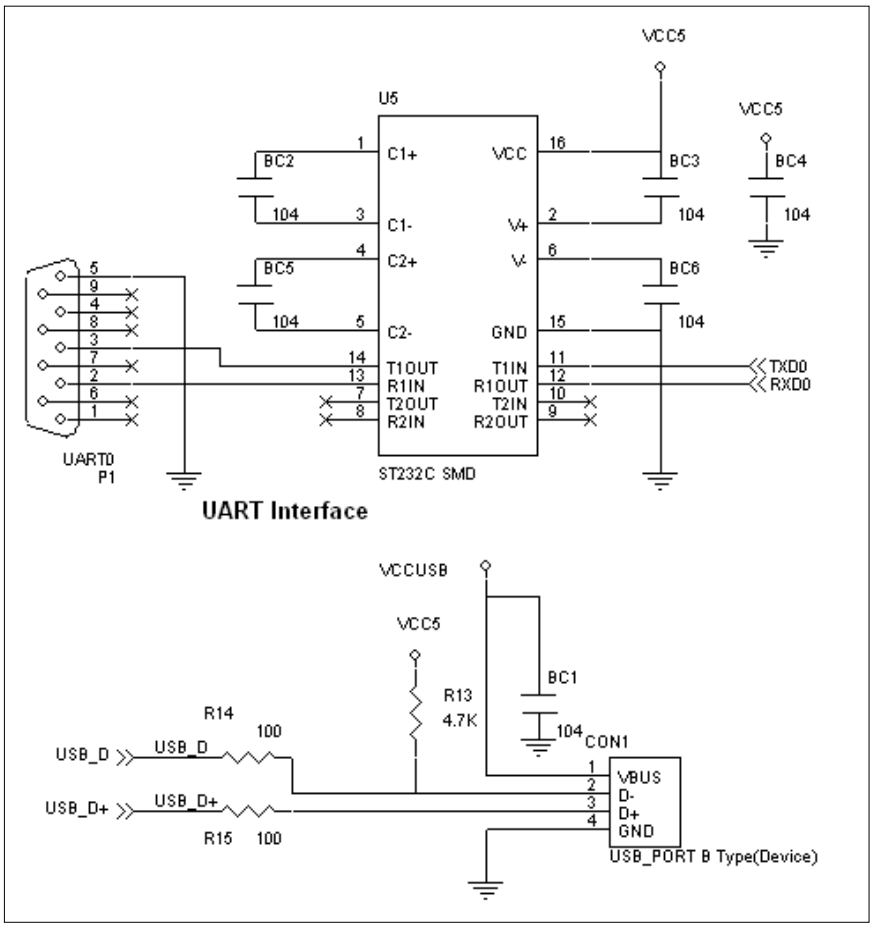

그림 53. UART/USB 회로도3

주어진 조건이 정해진 시간 내에 만족되는지를 판별하는 타이머 장치. 예를 들어 프로그램의 이상을 감지하기 위해 타이머를 설치하고 정상적인 경우에 프로그램이 주기적으로 설치된 타이머에 일정한 신호를 가하여 타이머에서 리셋 신호가 발생하지 않도록 한다. 프로그램(보통 무한루프를 가지고 있다)에서 주기적으로 신호를 발생해 주지 않으면 Watchdog Timer는 오류로 판단하여 리셋 신호를 발생, 마이크로프로세서 등의 시스템을 초기화한다. Watchdog은 집을 지키는 뜻을 내포하고 있는 단어의 조합을 감안한다면 개에게 신호라는 밥을 주지 않으면 프로그램을 관장하는 프로세서를 Dog라는 Timer가 물어버린다고 생각하면 된다. 여기서 신호는 주기적으로 Timer에게 제공할 수 있고 비주기적으로 공급할 수 있지만 타이머가 행동을 취하기 전에 신호(보통 충/방전 회로로 구성되어 있으므로 신호는 방전이 되지 전에)를 가해야 한다. Firmware에서는 시스템이 비정상적으로 무한 루프에 빠지거나 홀트(Holt) 되었을 때 Watchdog를 이용, Reset을 스스로 눌러 시스템을 자동으로 초기화 할 때 사용한다. 마이크로프로세서는 내부에 수많은 D F/F 등으로 이루어져 클락에 의해 동기화하는 구조로 되어 있다. 외부의 노이즈나 프로그램 오류로 인하여 Program Count 같은 포인터 지정자가 정해진 부분을 넘어 다른 것을 가리키면 Reset이 되기 전에는 빠져 나오지 않는다. 하드웨어 방식과 소프트웨어 방식으로 구분 할 수 있다.

또 하나의 디버깅을 어렵게 만든 것은 128*64 Dots의 그래픽 LCD(Powertip사의 PG-12864A)를 구동 할 때 Chip Select의 지연시간이 짧아 정상적인 출력을 얻을 수 없었는데 실제로는 High Holding Time이 450ns 이상이여야 한다. 다음의 화면은 Agilent Logic Analyzer 1672G로 Target 보드에서 추출한 신호이다. 여기서 왼쪽의 신호는 LCD에 입력되는 포트로써 μPSD의 PB Port에 연결해 놓았다. LCD_EN 신호는 PB2 포트에 연결해 놓고 High 일 때 Enable이 되도록 설정했으며 선택되는 번지는 0x400~0x40FF, 그리고 LCD_RSDI, LCDR-W, LCDCS1, LCDCS2 신호는 각각 A0, A1, A2, A3로 했다. LCD_EN 신호로 GLCD를 구동하기 위해서 최소 450ns는 되어야 하지만 측정된 값은 246ns이다. 이 값을 늘리기 위해서 Address 방법에서 포트로 지정하여 새로이 코딩 했다. 하강 시간은 10ns 이상이면 되므로 스펙에는 무난하다. LCD에 출력한 값은 0x41이며 ASCII의 영문자 'A'에 해당된다.

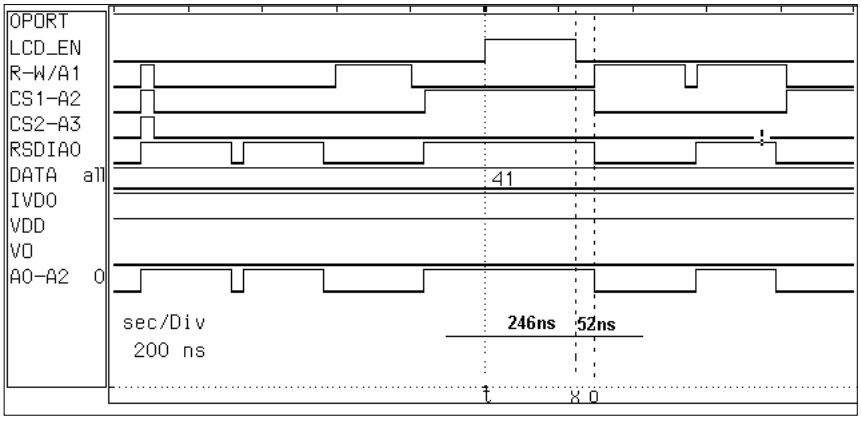

그림 54. LCD 신호 Timing 측정

이 파형을 관찰한 뒤에 G-LCD의 제어 신호를 어드레스 디코드에서 I/O 포트로 변경했다. 어드레스에 비해 프로그램의 양이 많아지고 복잡해지지만 특별한 대안이 현재 까지는 없었다.

알나간 if, while, do, for문(文)

---

if(조건1){루틴1}
else{루틴2}
if:괄호 안에 있는 조건1이 참(Zero가 아니면)이면 루틴1을 실행하고 조건1이 거짓(Zero)이면 else에 있는 루틴2를 실행한다. else를 생략하면 조건1이 참일 때 루틴1을 실행하고 또 루틴2를 실행한다. elseif도 있다.

while(조건2){루틴3}
while:조건2가 참이면 루틴3을 실행하고 거짓이면 한번도 실행하지 않는다. 무한 루프로 실행하기 위해 'while(1){……}'을 이용한다.

do{루틴4}while(조건3);
do:일단은 루틴4를 실행하고 조건3이 참이면 계속 실행, 거짓이면 do를 빠져 나온다. 적어도 한번은 실행하는 조건문이며 while 괄호 뒤에 세미콜론(';')을 붙이는 것에 주의해야 한다.

for(식1;조건4;식2){루틴5}
for:while 문을 변형한 것으로 셀 수 있는 루틴에 주로 사용한다. 식1은 변수의 초기화에 해당하고 식2는 증감(+, -)을 나타낸다. 조건4의 값이 참이면 루틴5를 실행하고 식2를 변경 후 다시 조건4를 검사한다. 조건4가 거짓이면 루틴5를 실행하지 않고 빠져나간다. 조건4가 초기에 0(Zero)이면 한번도 실행되지 않는다. 무한 루프로 실행하기 위해서는 'for(;;){……}'를 이용한다.

조건식에 들어가는 연산 기호와 우선순위에 대해 요약하면 다음과 같다.

연산자의 구분	기 호	통용되는 우선순위
괄호, 포인터	(),[],'.'	우선순위 높다
Type,증가,감소,반전,Not	(type),++,--,~,!	
산술 연산자	*,/,%,+,-	
비트 쉬프트	<<,>>	
비교 연산자	<,<=,>,>=,!=,==	
비트 연산자	&,^,|	
논리 연산자	&&,||	우선순위 낮다
대입 연산자	?:,=,+=,-=,*=,/=,<<=,>>=,&=,%=,^=,|=	

μPSD에 상품의 가치가 있는 원하는 프로그램을 탑재하기 위해 PSDsoft를 이용하여 로직을 구성하고 컴파일러나 어셈블리를 이용하여 Application을 작성한 뒤에 Target Board에 Porting, 수정하고 오류를 발견하여 다시 하드웨어나 프로그램을 수정하여 이식하고 다시 되풀이 되는 플래시메모리에 코드를 기록하는 일을 얼마나 많이 할까? 아마 데이터시트를 외울 만큼은 하리라고 생각한다. 그런데 이 이식하는 시간이 길면 개발자로 하여금 상당한 인내를 필요로 하고 또한 소스 상태에서 디버깅이 어려울 경우는 좀더 많은 절대적인 시간이 필요하게 된다. 이를 좀더 개선한 것이 있다면 Target Board와 시리얼 케이블로 연결되어 μPSD의 내부 상태를 관찰하면서 소스에 집적 Break를 걸어 정지와 실행을 반복하면서 알고리즘과 메모리/변수의 값으로 디버깅을 할 수 있는데 이것이 In-System Debugging이라고 한다. 잘만 활용하면 8032의 MDS(Micro Development System)나 ICE(In-Circuit Emulator) 만큼의 효과를 누릴 수 있다.

이렇게 인터페이스가 되기 위해서는 프로젝트에 In-System Object를 넣고 소스 프로그램에는 관련 헤드 파일과 함수를 연결하고 메인 함수에는 주기적으로 Loop를 실행하여 ISDcheck(); 함수를 넣어 주어야 한다. 단점으로는 UART 포트 한 개를 디버깅에 써야 하므로 동시에 사용하는 보드에는 불가능하다.

소스코드를 Target보드에 포팅한 뒤 KEIL 컴파일러의 dScope(적색 돋보기모양) 아이콘을 누르면 왼쪽 하단의 진행 막대가 증가 하면서 붙기를 시도 하는데 잘 안될 경우는 물리적으로 그리고 프로그램에서 먼저 오류를 검토하길 바란다. 시리얼 케이블은 타겟보드와 19200bps의 속도로 인터페이스 하므로 케이블 길이는 그리 민감하지 않으며 신호의 연결은 9핀일 경우에 2-3, 3-2, 5-5번의 서로 엇갈린 포트로 되어 있어야 한다. 그리고 저자의 실험용 보드와 프로그램은 24.576㎒의 클락으로 되어 있으므로 만약 다르다면 환경과 프로그램도 같이 변경해 주면된다.

다음의 화면은 정상적으로 Target과 연결 한 뒤에 소스코드 dip=P4;에 정지를 걸어 놓고 'Run' 한 다음 한 개의 명령어를 실행 한 뒤 화면을 잡은 것이다. 왼쪽에 CPU의 내부 상태와 오른쪽에 소스코드 그리고 타이머와 인터럽트 모니터 윈도우 하단에 메모리와 변수 윈도우가 보인다. dip 변수는 DPTR 번지로 확인하면 203d 번지인데 하단의 메모리 윈도우에서 동일하게 불러 오면 P4에서 불러

온 값 0xFF가 저장 된 것을 알 수 있다. 하드웨어의 딥스위치 값과 동일하지 않아서 xdata 영역 설정이 잘못된 것을 알 수 있다.

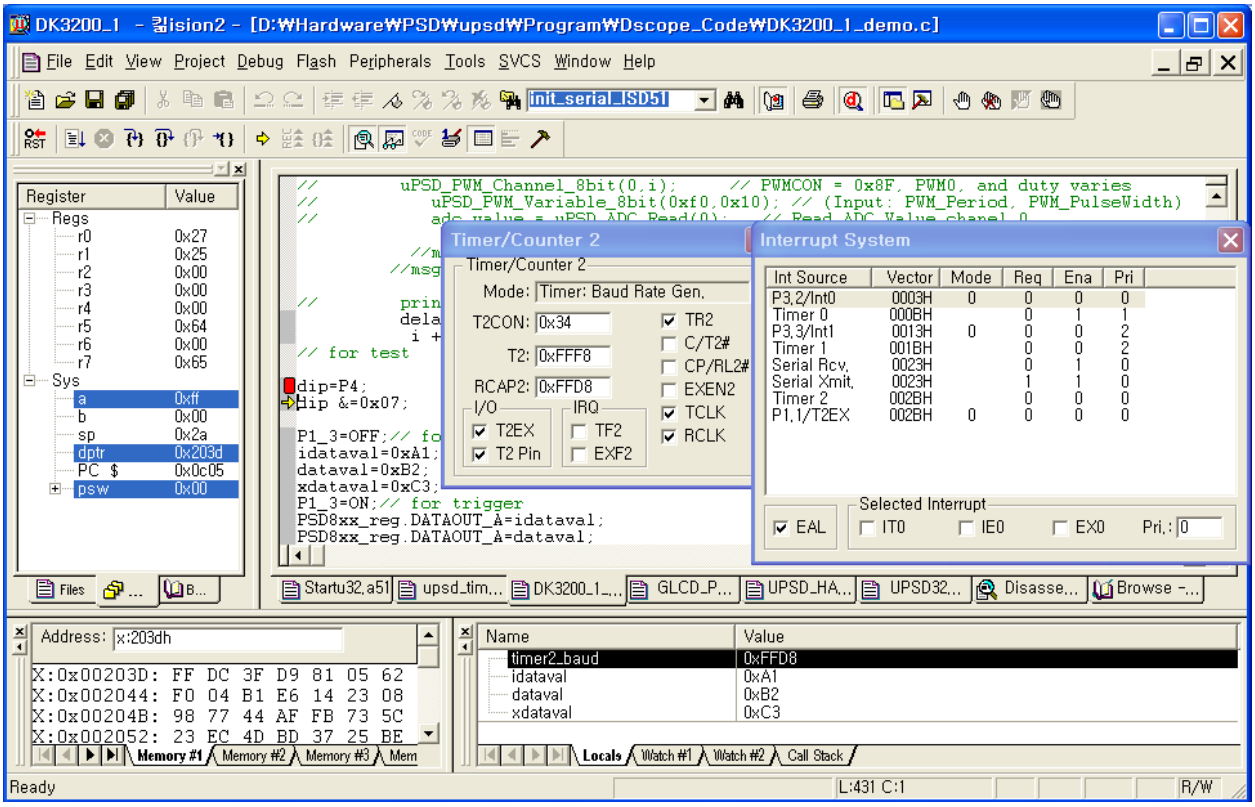

그림 55. dScope와 Target Board 통신 In-System 실행 화면

PSD8xx_reg.DATAOUT_A=idataval;// idata 변수를 204h 번지에 기록
PSD8xx_reg.DATAOUT_A=dataval;// data 변수를 204h 번지에 기록
PSD8xx_reg.DATAOUT_A=xdataval;// xdata 변수를 204h 번지에 기록

PSD8xx_reg.DATAOUT_A 변수는 'PSD Module Register', Csiop레지스터 0~FF의 메모리 영역을 구조체 Offset으로 선언한 뒤 이용된다. PSDsoft에서 Csiop의 시작 번지를 200으로 했기 때문에 A포트 Data out 레지스터는 4번째 이므로 200+4가 된다. 어드레스 윈도우에서 X:0x204한 뒤 실행화면 정상 동작되는지를 관찰 할 수 있다. 이상과 같이 간단하게 디스코프와 타겟보드와의 벌레잡이를 마치는데 역시 엔지니어가 실행하고 숙달하면서 숨겨진 Know How를 발견해야 할 것이다.

6. μPSD323x

8032 코어에 기반을 둔 8비트 프로세서에 PLD(Programmable Logic Device)와 Flesh메모리를 합한 것 같은 μPSD는 Package와 전압 그리고 메모리 사이즈에 따라 다양한 종류가 있다. 여기서는 Data Sheet에 있는 내용 중에 익히기 힘들고 난해한 부분에 대해서, 특히 8032 코어와 다른 부분에 대해서 설명하도록 하겠다.

μPSD323x 시리즈의 가장 큰 특징은 JTAG을 이용하여 프로그램을 로딩 할 수 있는 것일 것이다. 여기에 USB와 3.3V, 5V에 동작이 가능하고 I^2C, PWM, AD변환기 등이 있다는 것은 저자와 같은 Firmware 엔지니어에게는 그만이라고 할 수 있다.

제 품	Main/ Sec Flash	SRAM	Cell	USB	Vcc	최대 클락	핀 수
μPSD3234A-40	2Mb/256Kb	64Kb	16	O	5V	40㎒	52/80
μPSD3234BV-24				X	3V	24㎒	80
μPSD3233B-40	1Mb/256Kb			X	5V	40㎒	52/80
μPSD3233BV-24				X	3V	24㎒	52/80

표 9. μPSD 323x 시리즈 디바이스 종류(2003년 중반)

메모리 맵과 어드레스 공간의 영역도 기존의 8051과는 다른 것을 보인다. 왼쪽의 32K바이트 Second 플래시메모리는 8K씩 csboot0~3로 나누어져 있고 Main 플래시의 최대 256K는 fs0~7의 8개의 Paging으로 32K씩 8개로 나누어져 있다. 가운데 블록의 내부 메모리 0~7F번지까지는 KEIL에서 data 키워드를 사용하는 영역이고 80~FF까지는 idata, 그리고 80~FF의 SFR(Special Function Register)의 영역으로 나룰 수 있다. SFR 영역은 프로그램 할 때 굿이 만들지 말고 KEIL 컴파일러에서 제공되는 헤드파일 'KEIL설치경로＼C51＼INC＼ST＼upsd.h'에 이미 구성되어 있으므로 그대로 사용하면 된다. SFR은 기존의 8032 코어에 비해 ST사에서 추가로 한 부분이 상당수 있으므로 시간적 여유를 두고 검토가 필요하다. 오른쪽의 나머지 RAM 메모리 영역은 KEIL의 키워드 xdata를 사용하며 경우에 따라는 메모리이외의 I/O(Memory Mapped I/O) 장치를 연결해도 된다. 그림에 있는 메모리의 크기는 전부 Byte 단위이다.

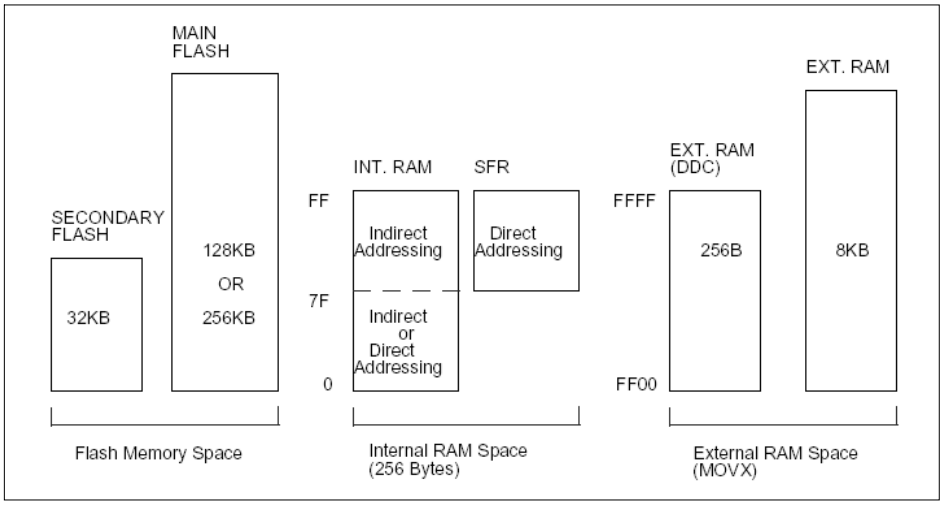

그림 56. μPSD 323x Address, Memory map

323x의 내부 블록도를 보면 인텔이 개발한 작은 프로세서 코어 8051이 진화와 합체로 성장한 것 같은 느낌이 든다. 이들 레지스터를 제어하기 위해서는 머리도 아프지만 그만큼 설계하기 위한 선택의

폭이 넓어 졌다고 할 수 있다. 포트는 기존의 8032관련된 Port0~4까지, PSD에 있는 PortA~D까지 해서 포트만 약 72개의 핀으로 구성된다. 포트는 Multiplex I/O로 한 개의 핀에 여러 가지의 기능을 선택적으로 사용할 수 있도록 하여 프로그림에 의해 다양한 동작을 선택/수행할 수 있다.

내부 블록도의 위쪽은 기존의 51 코어이지만 하단에는 PSD 부분이다. ST에서는 이를 다시 두개로 나누긴 했지만 PLD(Programmable Logic Device)부분과 USB 등의 디바이스에 대해 학습하면 8051 엔지니어라면 무난할 것이라 판단된다.

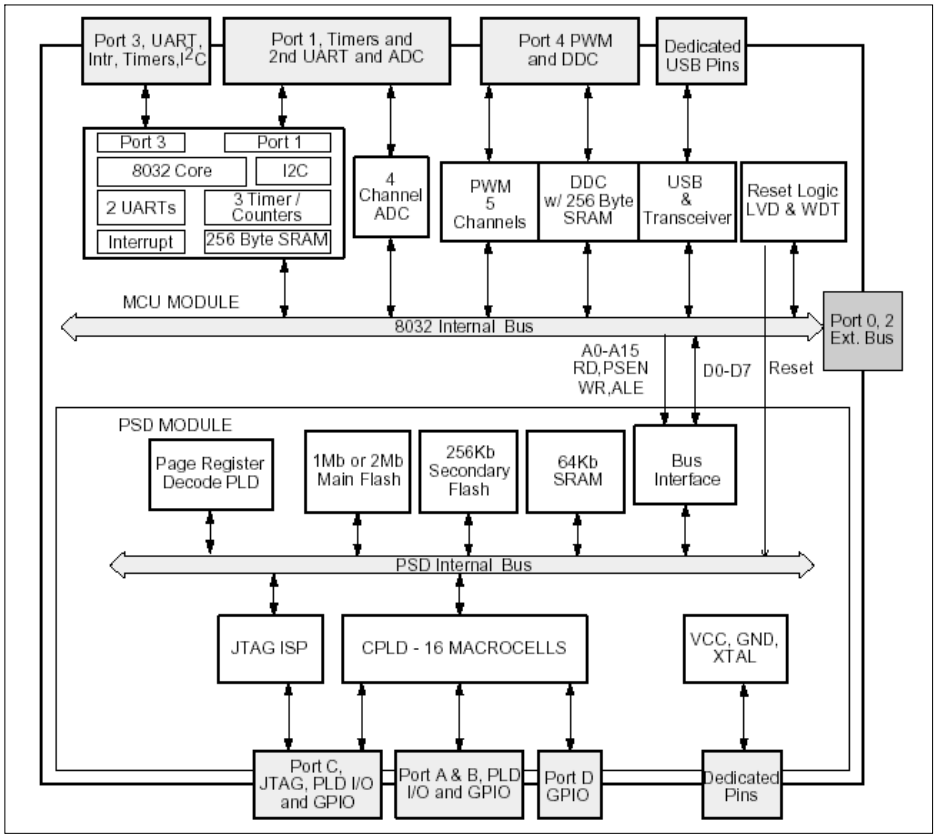

그림 57. 323x 80핀 내부 Block

16개의 Macrocell은 F/F에 해당되는 것으로 16개가 있다. 저자는 비트 단위의 메모리 Cell로 인식하고 사용한다.

다음은 인터럽트에 관한 사항인데 무지하게 추가되어 우선순위와 점프 번지가 늘어났으며 IP(Interrupt Priority)와 IE(Interrupt Enable)가 IPA, IEA로 새로이 설정 되었다. 이밖에 제자리걸음(?)을 하는 Idle 모드와 내부적으로 정지한 Power-Down 모드가 있다.

INT 이름	우선순위	IE, IEA(Enable)	IP, IPA(순위)	Vector(h)	비 고
INT 0	0, 높음	IE:0, EX0	IP:0, PX0	03	IE:A8h IEA:A7h IP:B8h IPA:B7h
USART(2nd)	1	IEA:4, ES2	IPA:4, PS2	4B	
Timer 0	2	IE:1, ET0	IP:1, PT0	B	
I^2C	3	IEA:1, EI2C	IPA:1, PI2C	43	
INT 1	4	IE:2, EX1	IP:2, PX1	13	
DDC	5	IEA:7, EDDC	IPA:7, PDDC	3B	
Timer 1	6	IE:3, ET1	IP:3, PT1	1B	
USB	7	IEA:0, EUSB	IPA:0, PUSB	33	
USART(1st)	8	IE:4, ES	IP:4, PS	23	
Timer 2+EXF2	9, 낮음	IE:5, ET2	IP:5, PT2	2B	

표 10. µPSD Interrupt

UART는 하나가 추가 되어 두개가 있는데 기본 동작은 8032와 같지만 Bps(Bit per Sec)는 약간의 문제가 있다. 이유는 최대 시스템 클락 40㎒를 사용할 경우 원하는 통신 속도 또는 정수배를 기대하기 어렵기 때문이다. USB에서 사용하는 36㎒를 사용할 경우는 예상치 못한 통신 에러(Spacing Error)가 생길수도 있으므로 신중히 고려해야 할 것이다.

통신모드 1은 많은 시스템에서 사용하게 되므로 이를 바탕으로 다음과 같은 통신 속도 계산식과 시스템 클락을 구하는 것을 예로 보였다. USB를 고려하지 않았을 경우에는 최대 주파수를 사용하는 것이 일반적이므로 이를 구했다. 참고적으로 클락은 OSC(Oscillator) 또는 Crystal 모두 소수점 이하의 수가 많을수록 정밀도 때문에 고가일수 밖에 없으며 시스템의 소비 전력도 고속일수록 증가한다.

$$TH값 = 256\,(FFh) - \frac{2^{MOD}}{32^{*}\,12^{*}\,시스템\ \ 클락^{*}\,원하는 Bps}$$

제시된 TH레지스터의 값을 구하는 공식에서 TH의 8비트 값이 FFh가 된 뒤에 0으로 될 때 인터럽트가 발생하게 되므로 MOD를 포함한 분수의 값이 정수가 되어야 한다. 주어진 최대의 시스템 클락에서는 통신 속도가 지장을 받게 된다. 다음은 각각의 통신 속도에서 정수값으로 나올 수 있는 시스템 클락을 구한 것이다. 그리고 나머지 통신 모드는 데이터 시트에 블록도와 함께 자세히 설명되어 있다.

9600Bps : 11.0592㎒, 22.1184㎒, 25.8048㎒, 29.4912㎒, 33.1776㎒, 36.864㎒, 40.5504㎒

19200Bps : 22.1184㎒, 29.4912㎒, 36.864㎒

38400Bps : 29.4912㎒, 44.2368㎒

57600Bps : 22.1184㎒

ADC(Analog Digital Covert)는 ACH0~3까지 4개의 채널이 있고 MUX에 의해 처리된다. AD 변환기

는 보통 6㎒로 AD 변환기에 클락을 주는 것으로 계산했을 때 10.67㎲의 변환시간을 가질 수 있다. 이것보다 더 바르게 했을 경우는 Sample & Hold 회로에 아날로그 전압이 충분히 충전되지 않으므로 정확한 변환을 보장 할 수 없다. 시스템 클락은 최대 40㎒이하에서 개발품의 환경에 따라 여러 가지 크리스탈을 가질 수 있으므로 8Bit pre-scaler를 이용하여 과용하게 설계하지 않도록 해야 한다.

$$8Bit\ \ \mathrm{Pre}-Scal=\frac{MCU\text{시스템클락}}{AD\text{변환}Clock(6Mhz)\ *\ 2}-1$$

그리고 AD 변환은 ACON:5, ADEN 비트를 '1'로 해야 변환이 가능하고 시작 신호(ACON:1, ADST)와 변환(ACON:0, ADSF)중인지를 점검하는 비트를 프로그램으로 제어해야 한다. 결국 변환된 8비트 데이터 값은 96h번지의 ADAT 레지스터에 놓이게 된다.

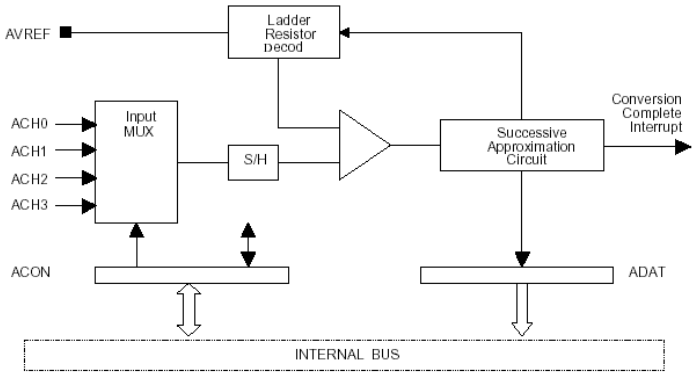

그림 58. AD 변환 블록도

PSD에는 크게 CPLD와 DPLD 부분으로 나눌 수 있는데 CPLD(Complex Programmable Logic Device)는 16개의 OMC(Output Micro Cell)와 24개의 IMC(Input Micro Cell)이 있으며 DPLD(Decode Programmable Logic Device)는 어드레스를 디코드 하는 부분이 들어 있다. 수명은 플래시메모리는 10만 번, PLD 부분은 천 번 쓰고 지우기가 가능하고 데이터의 저장은 15년간 할 수 있다.

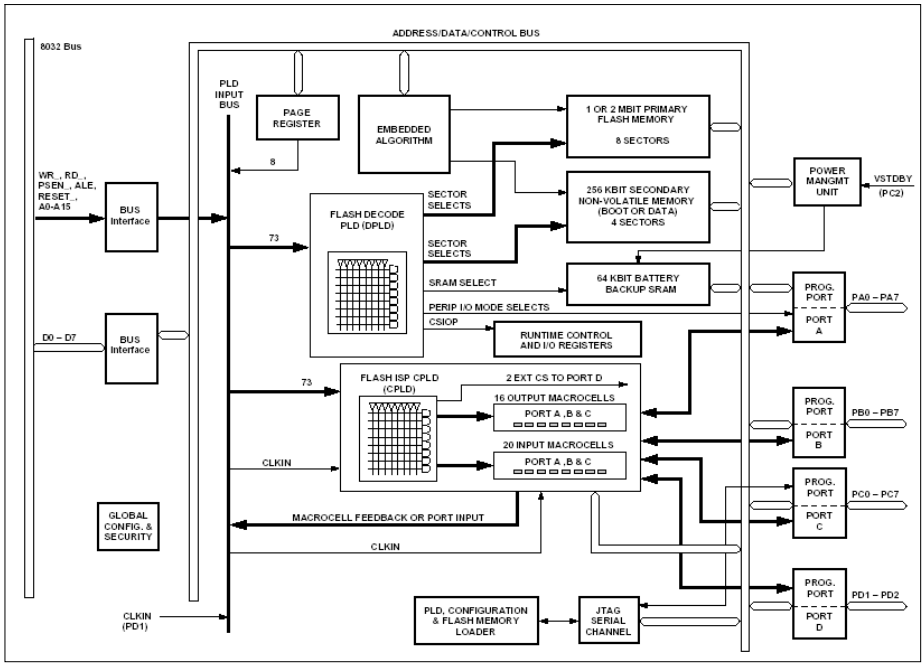

그림 59. PSD 내부 Block

7. JTAG Cable

PSD, μPSD에 Hex 파일과 이미지 정보를 다운로드 하고 통신하기 위해서 필요한 케이블을 Flashlink, 이때 PC에 설치하여 사용되는 소프트웨어를 'PSDsoft Express'라고 'ST Microelectronics'사에서는 부른다. 물론 JTAG과 관련 된 표준 스펙에 따라 개발된 것이지만 통상 접하는 JTAG 포트와는 약간 다른 구조를 가지고 있으므로 개발자들은 ST사를 통해 구입하거나 올바르게 자작해야 한다. 본 장에서는 이들의 구조와 제작하기 위한 기술에 대해 언급하고자 한다.

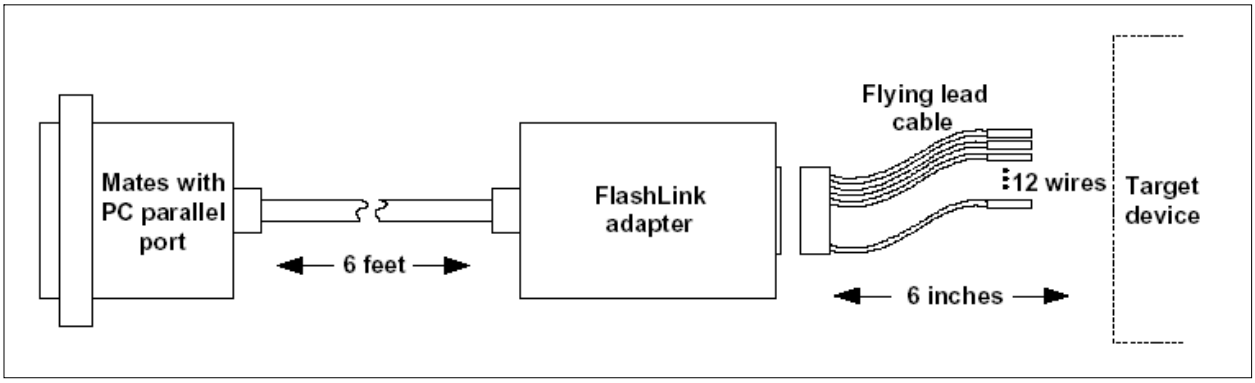

그림 60. Flashlink 구성

Flashlink는 프린터포트와 1.8m 길이의 케이블과 어댑터 그리고 타겟보드와 연결되는 14핀의 소켓으로 연결되어 있다. 프린터포트는 KEIL 컴파일을 사용할 경우 PC에 설치된 락키 다음에 설치하고 반대쪽은 1번핀에 주의하여 Target Board에 연결한다. 어댑터는 타겟보드로부터 공급 받은 전원

(2.7~5.5V)으로 구동되고 PC와 버퍼를 통해서 연결된다.

핀	신호이름	설 명	비 고
1	/JEN	Enable JTAG	Option
2	/TRST	Reset	Option, Target과 연결
3	GND	그라운드	
4	CNTL	제어	Option
5	TDI	JTAG Serial Data Input(Flashlink 측면은 출력)	PSD에 입력
6	TSTAT	EJTAG Program State	Option
7	Vcc	DC 2.7~5.5V(최대 15㎃@5.5V)	Target으로부터 공급
8	/RST	시스템 리셋	Target과 연결
9	TMS	JTAG Mode Select	
10	GND	그라운드	
11	TCK	JTAG Clock	
12	GND	그라운드	
13	TDO	JTAG Serial Data Output	
14	/TERR	JTAG Programming Error	Option

표 11. Flashlink 핀 설명

14개의 핀에서 반드시 JTAG을 위해 설계해야 하는 핀이라면 Vcc와 GND를 포함하여 표에서 회색으로 표시되어 있는 6개의 신호들이다. 자작하거나 설계할 때 공간적으로 필요에 의해 설계 한다면 Core 핀이므로 염두에 두자. Option은 ISP와 Multiplexing를 위한 신호이다. JTAG 신호 케이블은 Flashlink 프린터포트와는 동선을 사용하지만 Flashlink에서 Target Board와는 리본케이블을 사용한다. 이때 리본케이블이 길어진다면 신호케이블의 양면에 그라운드 처리('TCK'가 다른 신호에 비해 빠른(약 400㎑) 주파수를 띄고 있으므로)를 하여 감싸 주어야 전자파 장애로부터 보호를 받을 수 있다.

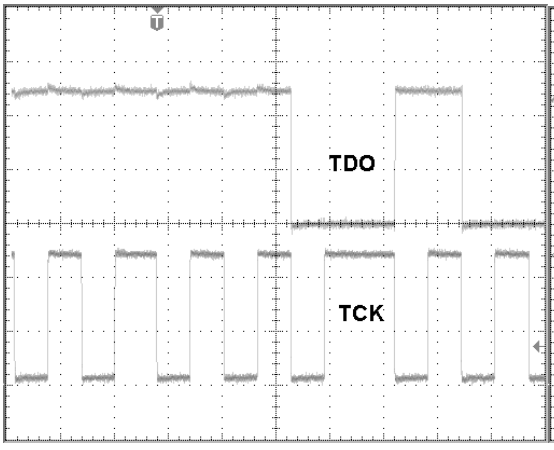

그림 61. TDO 신호

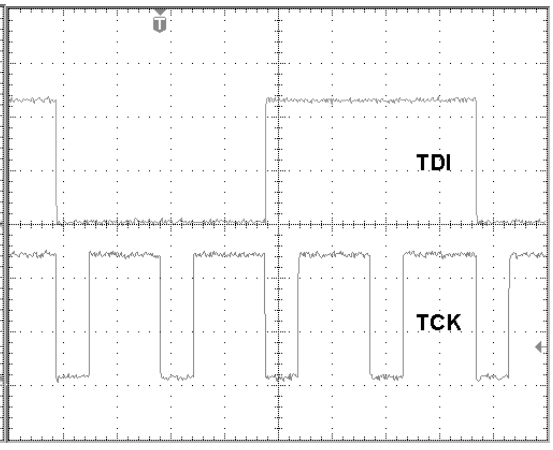

그림 62. TDI 신호

다음의 그림은 1개 이상의 칩을 JTAG를 이용하여 제어하기 위한 예제를 보인 것이다. 한 개의

Target에 서로 다른 μPSD나 혹은 전압을 가지고 있어도 데이지 체인으로 연결할 수 있다.

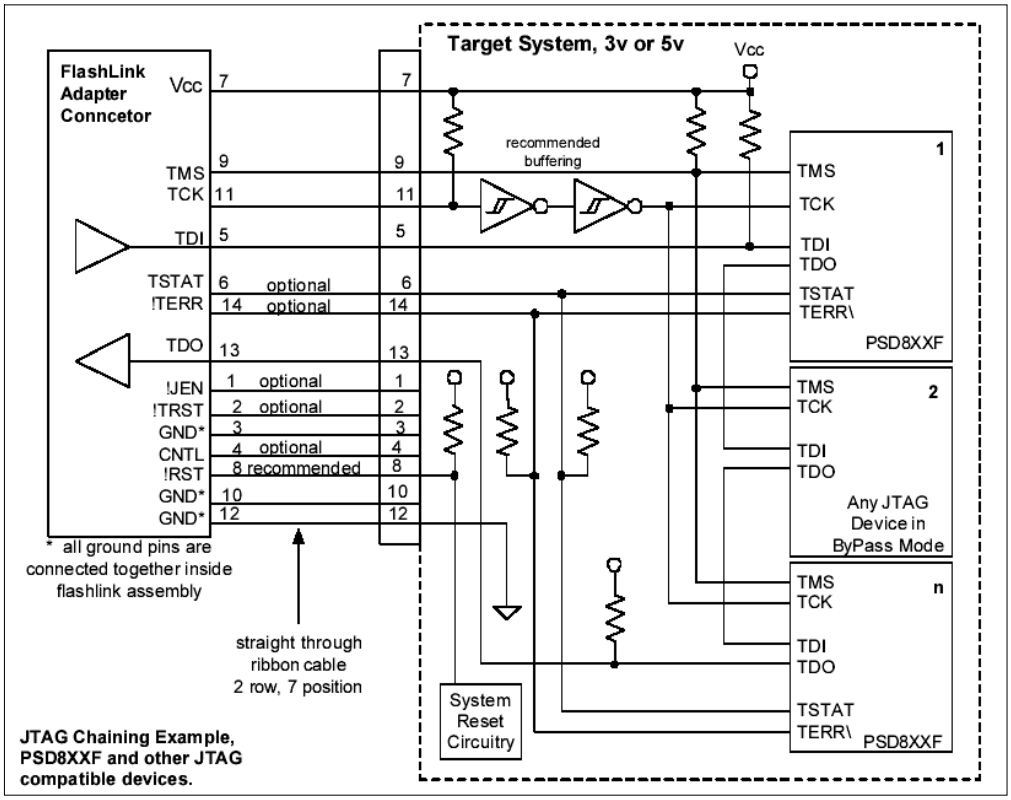

그림 63. PSD, μPSD의 Multiplex 연결 방법

다음은 프린터 포트와 JTAG를 연결하기 위한 Flashlink의 회로도와 실제의 시판되는 제품의 PCB를 촬영한 모습이다. 대량 생산으로 갈 경우나 신뢰성에 의심이 갈 경우는 ST사로부터 구입($59)해야 하겠지만 실험실에서 혹은 취미로 구성하기에는 무리가 없을 것으로 판단된다. 회로도는 ST사에서 제공되는 DK900보드의 사용설명서 PDF 파일에서 참고하여 다시 캐드를 이용하여 그린 것으로 왼쪽은 프린터 포트와 연결되는 부분이고 오른쪽의 Port는 Target과 연결되는 부분이다. 부품의 번호는 무작위로 만들고 중복이 되었다. PC의 프린터 포트와 Flashlink와의 연결은 1~25번까지 일대일로 연결된 케이블을 사용하면 된다.

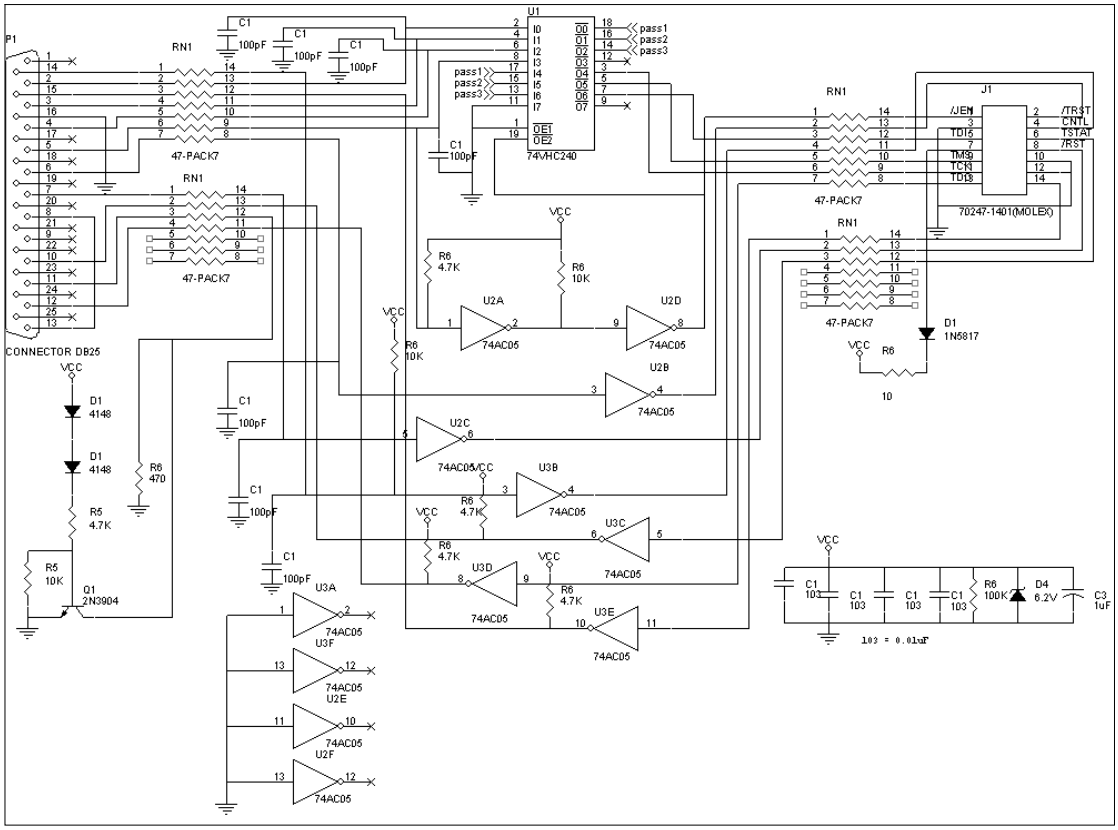

그림 64. Flashlink 회로도

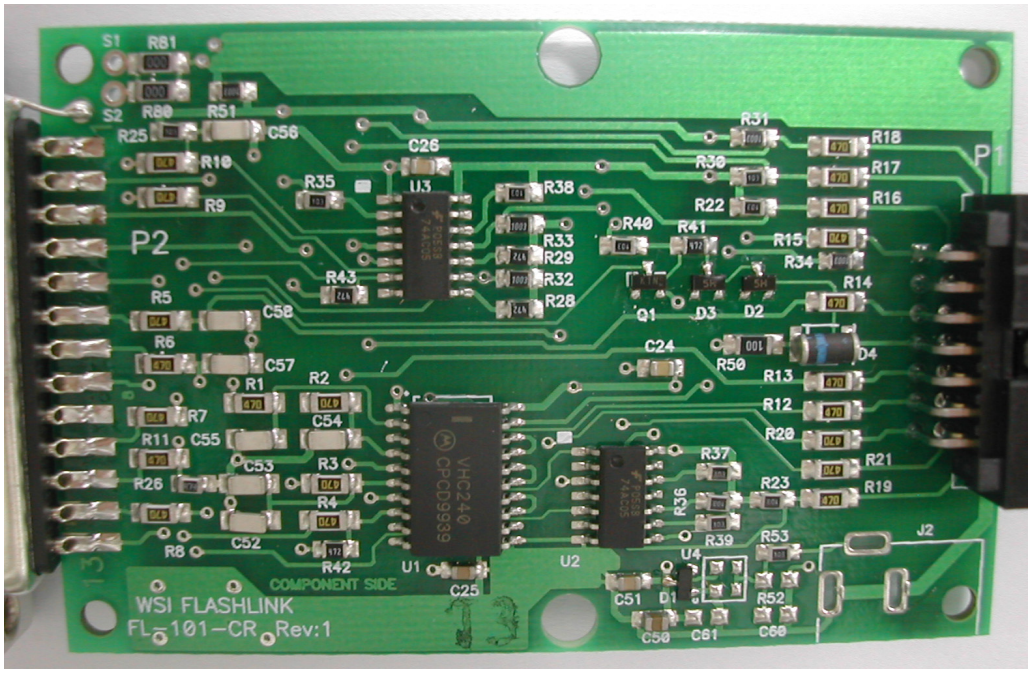

그림 65. Flashlink PCB

8. 참고자료

DK900 Demo Board 사용자 매뉴얼 V1.10
Optimizing 8051 C Compiler and Library Reference, KEIL, 01.97
8051/251 user's Manual, KEIL, 11.97
dScope for Windows, KEIL, 01.97
Getting Started with µVision2, KEIL, 02.2001
Microprocessor Principles and Application, 권장우 외6명, 사이텍미디어, 2001
C and The 8051, Thomas W.Schultz, Prentice-Hall, 1999
8051 코어 DS5002FP를 이용한 지킴이 제작, 황희융/김형태, 교학사, 1999
How to use KEIL 8051 C Compiler, 김형태, 양서각, 2002
New C 언어 입문-중급편-, 기획2교재팀, 영진.com, 2000
uPSD323X Data sheet, ST사, 2003

9. 찾아보기

