Trace32 Manual

ARM 사업부 기술지원

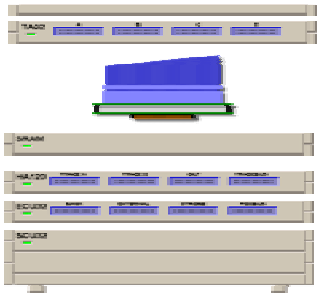
Contents

1. 기본 환경 설정

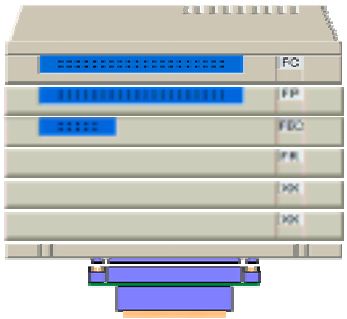
1.1 Emulator의 종류

Emulaor 혹은 debugger는 크게 세가지 범주로 나눌 수 있는데 CPU emulator, JTAG & BDM 방식의 emulator, ROM monitor 방식의 emulator가 이에 속한다. 각각의 emulator의 장©단점은 다음과 같다.

1.1.1 CPU emulator(Trace32 ICE/FIRE) : target CPU와 동일하거나 상위의 CPU를 내장하고 있는 emulator로서 emulation memory 사용, Trace & Trigger 기능, Code coverage & Performance기능 등의 강력한 디버깅 기능을 가지고 있는 emulator이다.
단점으로는 가격이 고가이며, cpu clock이 100Mhz이상의 경우는 제조하기도 어렵다. 게다가 사용하기 위해서는 CPU의 모든 pin들을 장비와 인터페이싱 시켜야 하기 때문에 다소 불편하다. 게다가 ARM의 경우 core level로 ASIC에 내장되기 때문에 더욱 인터페이싱하기 어려워진다.



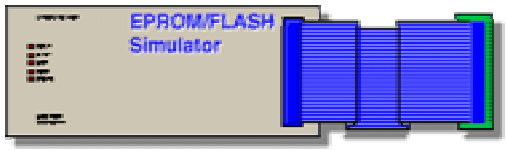
< Trace32 ICE >



< Trace32 FIRE >

1.1.2 JTAG & BDM 방식 emulator(Trace32 ICD) : Boundary Scan Tester을 기반으로 해서 나온 JTAG 또는 BDM port을 이용하여 디버깅하는 방식으로 몇 개의 기본 핀(TDI, TDO, TMS, TCK, TRST, VCC, GROUND등)만을 연결하면 됨으로 인터페이싱하기 용이하며, 타겟 자원을 그대로 사용하기 때문에 거의 realtime으로 타겟을 monitoring할 수 있는 emulator이다.
단점으로는 CPU emulator의 강력한 기능인 Trace & Trigger기능, Code Coverage & Performance기능을 사용할 수 없다는 것이다. 참고로, 이에 대한 보완으로 ETM/NEXUS 방식을 이용한 emulator가 나오고 있다.

1.1.3 ROM monitor 방식 emulator(Trace32 EEPROM Simulator) : JTAG & BDM port가 내장되어 있지 않는 68k, C196과 같은 chip의 경우는 ROM monitor방식을 이용하여 디버깅 할 수 있다. 이 경우 emulator와 통신에 필요한 최소한의 monitor program이 사용자 소스와 integration되어야 한다.
단점으로는 타겟 메모리의 일부를 emulator를 위한 monitor program에서 사용되어야 한다. 참고로 RTOS debugger도 일종의 ROM monitor 방식 emulator라고 말 할 수 있다.



1.2 Host Environments

Trace32은 다양한 호스트 개발 환경에서 사용할 수 있다.
Windows기반의 PC일 경우, 현재 Windows 95/98, Windows NT, Windows2000, Windows XP등 모든 windows platform에서 사용할 수 있다.
Solaris, HP와 같은 Unix 기반의 platform뿐만 아니라 요즘 널리 애용되고 있는 Linux환경에서도 Windows와 똑 같은 환경에서 Trace32 PowerView 소프트웨어를 사용 가능하다. 단, 사용자가 Windows외의 platform에서 Trace32를 사용하고자 할 경우는 반드시 Ethernet방식의 인터페이스 모듈을 사용하여야 한다.

1.3 Supporting High Level Language

Source level debugging이 가능한 program languages로는 Assembly, C/C++, JAVA등 범용적인 모든 embedded languages가 지원된다.
(source level debugging이란 사용자가 작성한 소스형태로 디버깅할 수 있다는 것을 의미한다. 예전에는 단지 binary code를 disassemble하여 디버깅하였다.)

1.4 Supporting Compilers' Generating Formats

Compiler가 만들어 내는 범용적인 포맷으로는 binary code, hex code, absolute format code등을 들 수 있다. 각각의 code의 특징은 다음과 같다.

1.4.1 Binary code : 타겟을 구동시키는 실제적인 코드로 address정보를 가지고 있지 않아 debugger나 ROM writer등을 이용하여 타겟에 다운로드 할 경우 반드시 번지를

지정해야 한다.

1.4.2 Hex format code : binary code, address information, checksum으로 구성된 file format으로 address information과 checksum은 타겟을 구동시키는 real code에 포함되지 않으며, 단지 ROM writer와 같은 tool을 위한 정보임. Hex format의 경우는 checksum이 들어 있기 때문에 memory에 다운로드하는 속도는 다소 느리나 정확하게 다운로드 할 수 있다는 장점이 있다.

1.4.3 Absolute format code : binary code, address & symbols' information으로 구성되어 있으며, debugger를 위한 format임. Debugger의 경우는 symbol information을 이용하여 source level debugging이 가능하게 만들어 준다.

1.5 RTOS Kernel Awareness

기존의 CPU emulator, JTAG-BDM emulator의 경우는 software뿐만 아니라 hardware까지도 디버깅하는 일이 가능하다. 왜냐하면, 타겟의 register, memory, external I/O port등 세부적으로 하드웨어를 제어하는 일이 가능하기 때문이다. 즉, 타겟의 전반적인 모든 사항을 체크할 수 있다는 점을 장점으로 볼 수 있다. 그러나, RTOS kernel에 대한 정보는 전혀 볼 수 없다는 단점이 있다. 즉, microscopic view는 가능하나, 전체적인 프로그램 task의 흐름에 대한 macroscopic view는 할 수 없다는 점이다. 보통 이러한 경우에는 RTOS kernel debugger를 사용해야 한다.
다행히도, Trace32의 경우는 Kernel awareness기능을 가지고 있어서, RTOS kernel debugger없이도 디버깅하면서 kernel을 정보를 읽어 올 수 있다.
현재, 범용적인 모든 RTOS에 대해서 이러한 작업이 가능하다.

예를 들면, Vxworks, Nucleus, AMX, pSOS, Vrtxsa, Osebasic, Rtc EMBOS, ThreadX, RTXC, OSEDELTA, OSEEPSILON 등의 RTOS등이 Kernel awareness가 지원 된다.
물론, 각각의 RTOS를 사용할 경우 다음의 설정이 필요하다.
이러한 명령은 Macro file내에 정의 할 수 있으나, command line에 바로 수행할 수도 있다. 만약 Nucleus를 사용하고 있다면, nuc.t32파일과 nuc.men파일이 필요한데, 이 파일들은 demo₩arm₩kernel₩nucleus 디렉토리 내에 존재한다.
이 파일을 이용하여 다음과 같은 명령을 수행하면 된다.

```
Task.config nuc.t32
Menu.rp nuc.men
```

1.6 사용자가 수정 가능한 파일

Trace32 PowerView software에는 사용자가 수정 가능한 파일들이 있다.
그 중에서 대표적인 파일로는 config.t32, t32.cmm, t32.men파일이 있다.
각 파일의 특징은 다음과 같다.

1.6.1 Config.t32 : host pc와 ICD와 인터페이스 설정(여기서 PBI란 POD BUS INTERFACE)

Parallel의 경우:

```
PBI=
LPT1
```

USB의 경우:

```
PBI=
USB
```

Simulator 경우:

```
PBI=sim
```

1.6.2 T32.cmm : Trace32 PowerView에 default로 포함되어 있는 매크로 파일로서 PowerView 구동시 항상 초기에 읽어오는 파일이다. 그래서, startup파일로서 사용이 가능. 예를 들면, 디버깅시 필요한 일반적인 사항을 이 파일 내에 그대로 기술하면, PowerView 구동시 이러한 내용을 배치파일처럼 일괄 처리할 수 있다.

EX1.6.2-1 프로그램 실행시 셋팅한 환경을 항상 불러 들이는 명령.

```
Do win.cmm ; open the stored windows
---------------------------------------
System.up
Data.load.elf *.elf /nocode
Map.bonchip 0x0--0x1fffff
```

위와 같이 t32.cmm파일 내에 삽입하면, PowerView이 구동되면서 윈도우 환경, 파일 다운로드, 브레이크포인트 설정 등을 자동적으로 수행할 수 있다.

1.6.3 T32.men : 메뉴 파일의 일종으로 PowerView의 default main menu로서 윈도우 환경을 사용자의 취향에 맞게 수정할 때 사용한다.

EX1.6.3-1 Toolitem을 추가하여 메뉴 편집

```
Toolitem  "파일 다운로드"   "DN"          "do download.cmm"
          (tooltip)      (icon shape)   (icon를 click할 때 수행할 명령)
```

Command line에 (B:: menu)라고 입력하고 toolitem을 작성한 후 compile하면 메뉴가 추가된다.

1.7 하드웨어 연결시 주의 사항

1.7.1 jtag cable의 1번 pin과 타겟 jtag의 1번 pin을 항상 확인하고 연결한다.

1.7.2 Power on/off 순서 Power on시 : icd /target 순서

Power off시 : target/icd 순서

1.8 Debugging 시작 절차

Trace32에서 가장 중요한 명령은 system.up이라는 명령을 것이다. 타겟과 icd를 연결한 상태에서 전원을 인가하고 PowerView를 구동시킨다하더라도 아직 icd에서 타겟을 제어하고 있는 것은 아니다. System.up명령을 수행하는 순간부터 제어권을 icd가 갖게 된다.
이 명령 수행시 다음과 같은 일련의 과정이 수행된다.

System.up – a. debug port test
b. resetting the target board
c. locating PC(program counter) into reset vector

1.9 디버깅 환경 설정

디버깅할 프로그램을 다운로드하는 위치(RAM/FLASH)에 따라 디버깅 환경을 다르게 꾸밀 수 있다.

EX1.9-1 RAM에 다운로드 하여 디버깅하는 경우

```
System.up
Data.load.elf filename
```

EX1.9-2 Flash/ROM에 다운로드 하여 디버깅하는 경우

이 경우는 두 가지 과정을 나누어서 생각해야 한다.

a. flash memory programming
b. flash memory debugging

만약 두 번째의 경우처럼 flash memory를 debugging하는 경우에는 RAM 다운로드에

비해서 다음과 같은 절차가 필요하다.

```
System.up
Data.load.elf filename /nocode ;symbol 정보만 다운로드
Map.bonchip 0x0—0x1fffff ; flash/ROM영역에 breakpoint를 걸기 위해 필요
```

1.10 Macro 기능

매크로 기능은 크게 두 가지로 나눌 수 있다.
배치 파일 기능과 테스트 프로그래밍 기능이다.
배치 파일 기능은 일련의 작업 과정을 매크로로 만들어 단순화 시키는 작업을 말한다.
그런데, Trace32의 가장 우수한 기능은 바로 테스트 프로그래밍을 할 수 있다는 것이다.
지금부터는 가장 간편하게 사용할 수 있는 매크로 명령들과 예제를 살펴 보기로 하자

EX1.10-1 기본 명령어

```
label => start:
goto문 => goto label명
wait문 => wait 1.s : 1초간 기다린다.  Wait 100.ms :100 milisecond간 기다린다.
Print문 => print “ “        문자열, 메모리, 레지스터 값을 area에 표시해 준다.
변수 print문 => v.print 변수이름
go, step, step. over, break와 같은 Trace32 일반 command.
화면 update문 : screen
```

EX1.10-2 자동 step하면서 메모리/레지스터/변수 값 print하기
(label의 경우 이름 뒤에 “:”을 붙히면 됨.)

```
start:
step   ; 한 스텝 수행
wait 100.ms ; 100 millisecond 멈춤
screen ; 화면 업데이트 명령
goto start
```

EX1.10-3 일정한 시간 간격으로 go/break하면서 중요한 윈도우정보 저장하기

```
printer.filetype ascii ; 저장할 데이터의 포맷
printer.open k.txt ; 데이터를 저장할 파일,나중에 반드시 printer.close해야 함
start:
go
wait 1.s ; 1초 간격으로 go/break (millisecond의 경우는 1.us)
break
wp.v.w ; 저장할 윈도우이름앞에 wp를 붙혀준다.
goto start
```

EX1.10-4 반복적으로 수행할 명령어

```
(go/break를 10번 반복하고자 할 때)

repeat 10.
(
        g
        wait 1.s
        break
)
```

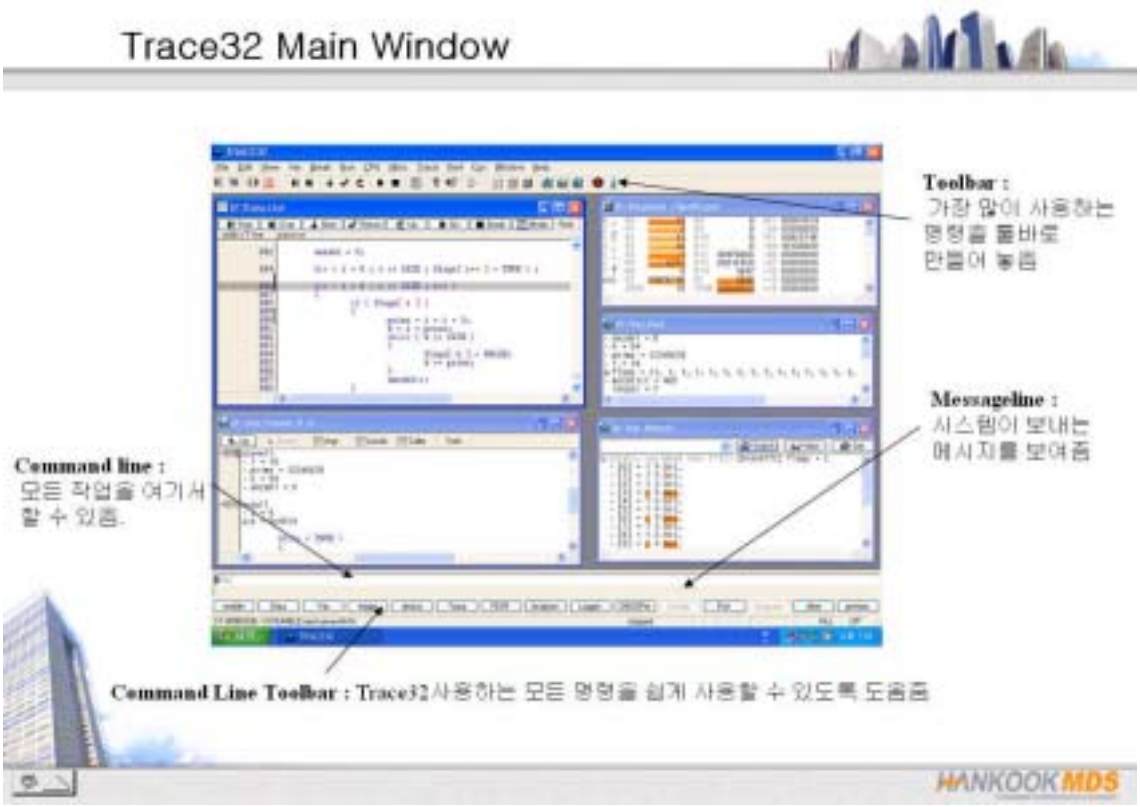
1.11 윈도우 페이지

Trace32의 PowerView에는 유용한 정보를 보여 주는 윈도우가 굉장히 많다. 이러한 윈도우들을 한 화면에서 모두 보기는 불가능하다. 그렇다고 해서 여러 개의 화면을 각각 꾸며 놓고 필요할 때마다 따로 불러내는 일도 불편하다. 그런데, PowerView는 이런 점을 페이지 개념을 도입하여 해결하고 있다. 각 페이지마다 원하는 윈도우들을 꾸며 놓고, 이것을 하나의 매크로 파일로 저장하기만 하면, 원하는 모든 윈도우 환경을 한꺼번에 불러 올 수 있게 된다. 새로운 페이지를 만들고자 할 때는 PowerView 바탕화면에서 pop-up 메뉴를 띄워서 수행할 수 있다.

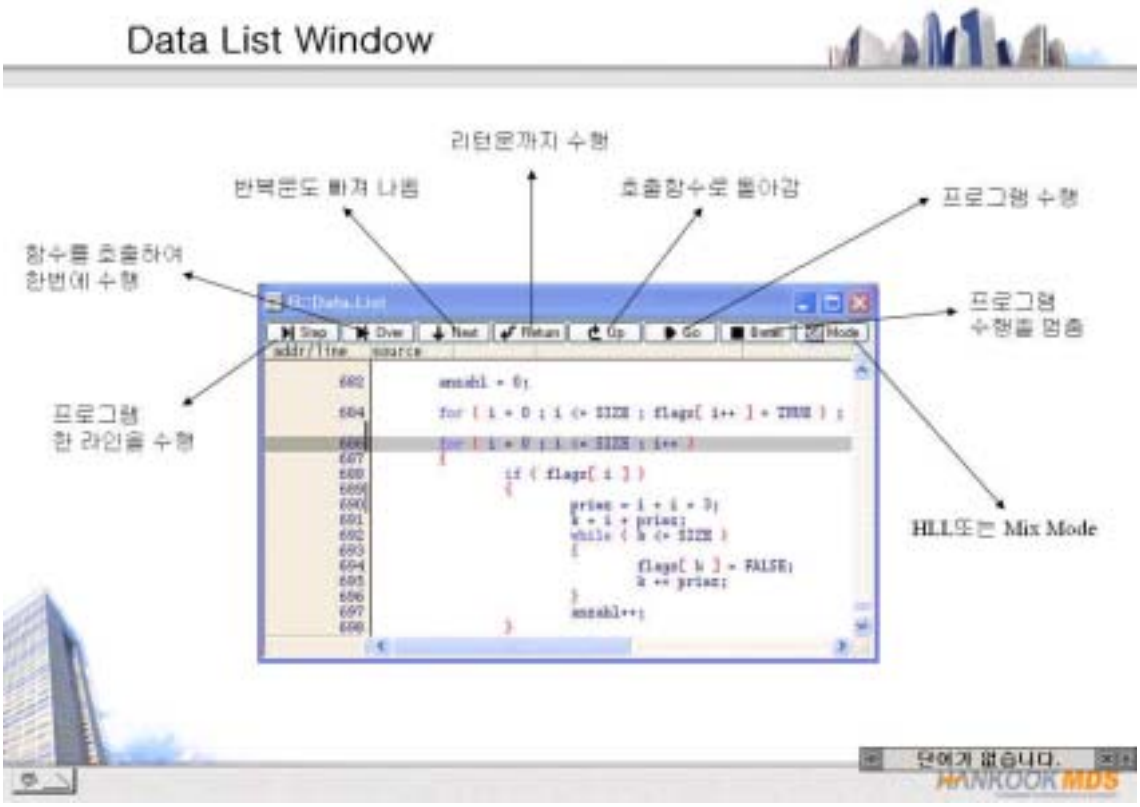
2. HLL(High Level Language) Debugging

2.1 Trace32 PowerView Main Window

PowerView software의 가장 큰 장점은 command line을 쉽게 사용할 수 있다는 것이다. 게다가 command line을 보조하는 툴 바가 item별로 잘 정리되어 있으므로 보다 빠르고 쉽게 command를 통해 타겟을 제어하면서 디버깅할 수 있는 환경을 제공해 준다.



2.2 Source/List 윈도우



Source/List윈도우는 실제로 사용자 작성한 소스코드를 볼 수 있는 윈도우이다.
C/C++, JAVA code등으로 작성된 코드를 직접 보면서 디버깅이 가능하다.
Source/List 윈도우에는 Mixed mode / HLL mode /Disassembled 윈도우가 있다.

2.2.1 Source/List 윈도우 종류

a. Mixed mode 윈도우 : data.list명령을 사용 불러 냄. Mode button을 클릭하여 C/C++ source만을 보거나 C/C++ source/assembled code를 동시에 볼 수 있다.
b. HLL mode (High Level Language) 윈도우: data.listhll명령을 사용 불러냄
c. Disassembled mode 윈도우 : data.listasm명령을 사용 불러냄

2.2.2 유용한 명령어

a. Step : C/C++ 소스라인/disassembled 코드를 한라인씩 동작시키면서 진행. 임의의 함수에 프로그램카운터가 있을 때 step명령을 수행하면 함수 내로 들어 갈 수 있다.(step into function기능); command line에서는 s 또는 step 명령
b. Step Over : step명령과 비슷하나 함수에 프로그램카운터가 있을 때 함수 단위로

step 할 수 있다.(step over function); command line에서 s.o 또는 Step.Over 명령

c. Next : Step Over 명령과 유사하나 mechanism이 step over명령과 전혀 다르다. Next명령의 경우 무조건 현재의 프로그램카운터의 다음번지에 브레이크포인트를 걸고 go command를 수행한 과정과 동일하다.

d. Return : 현재 프로그램 카운터가 있는 함수의 return문까지 수행; command line에서 g.r 또는 Go.return 명령 수행

e. Up : 현재 프로그램 카운터가 있는 함수를 다 수행하고 빠져 나옴; command line에서 g.u 또는 Go.Up 명령 수행.

f. Mode : Mixed mode(C/disassembled) 와 C/C++ mode를 mode button를 클릭함으로서 toggle함

2.2.3 Popup 메뉴

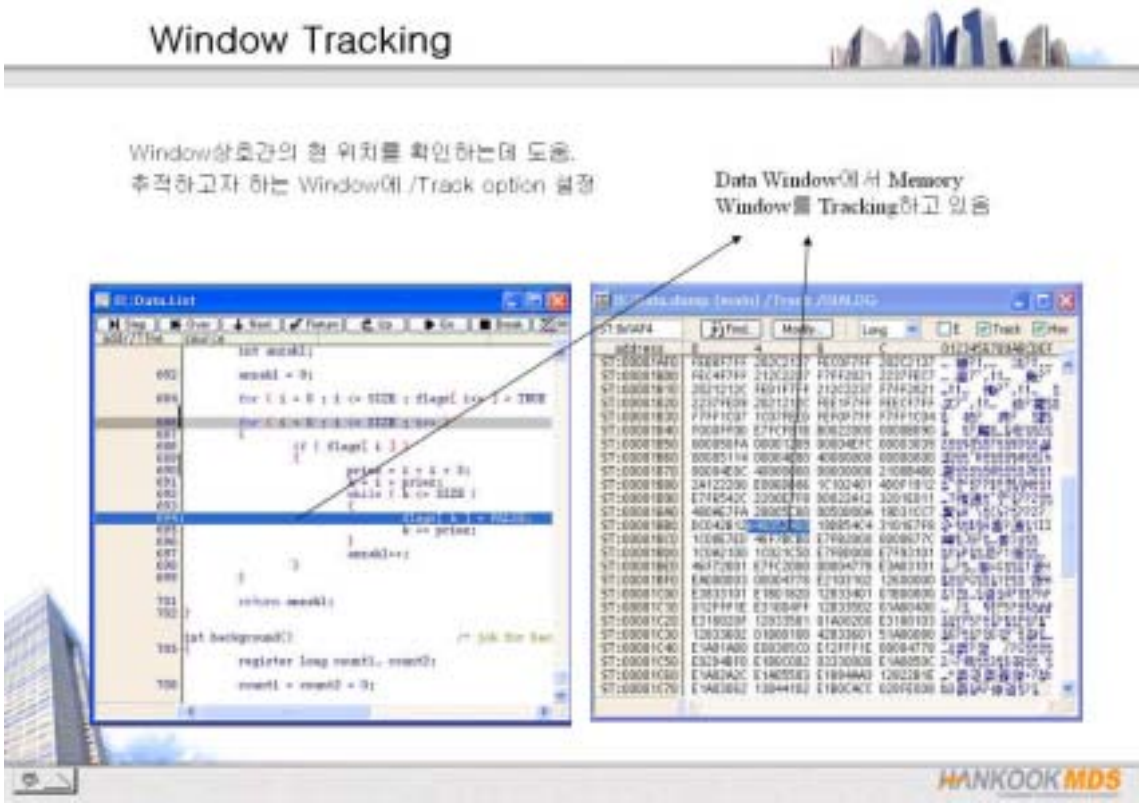
a. Go Till Here : 현재 cursor가 있는 위치까지 수행

b. Go Till Read, Write : 선택한 변수를 Read 또는 Write할 때까지 수행

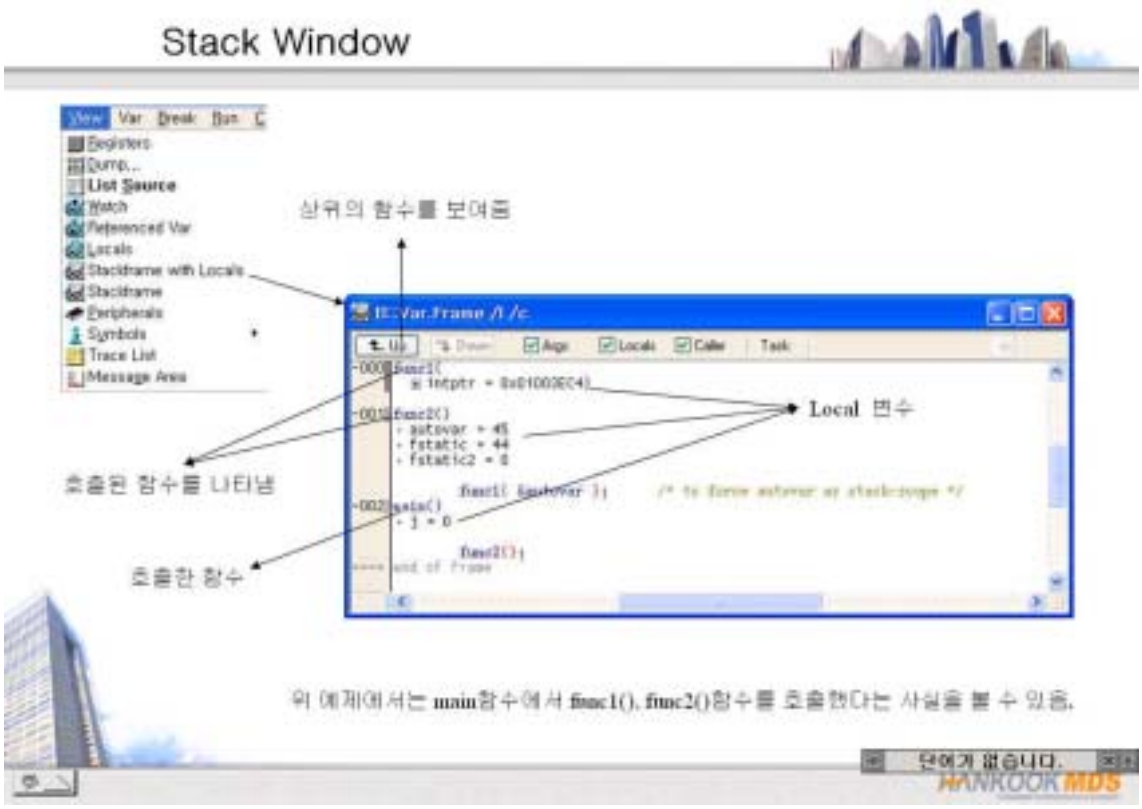
c. Go Till There : 선택한 함수를 처음 call한 시점까지 수행

d. Set PC Here : Program Counter(PC)만 선택한 cursor위치로 옮김.

2.3 윈도우 트랙킹 기능


Window Tracking
Window상호간의 현 위치를 확인하는데 도움.
추적하고자 하는 Window에 /Track option 설정
Data Window에서 Memory
Window를 Tracking하고 있음
HANKOOK MDS

2.4 스택 윈도우: Stackframe menu



관련 Command :

Command : Var.Frame
Format : Var.Frame [%<format>] [/<option>]
<option> : NoVar, Args, Locals, Caller

설명 :

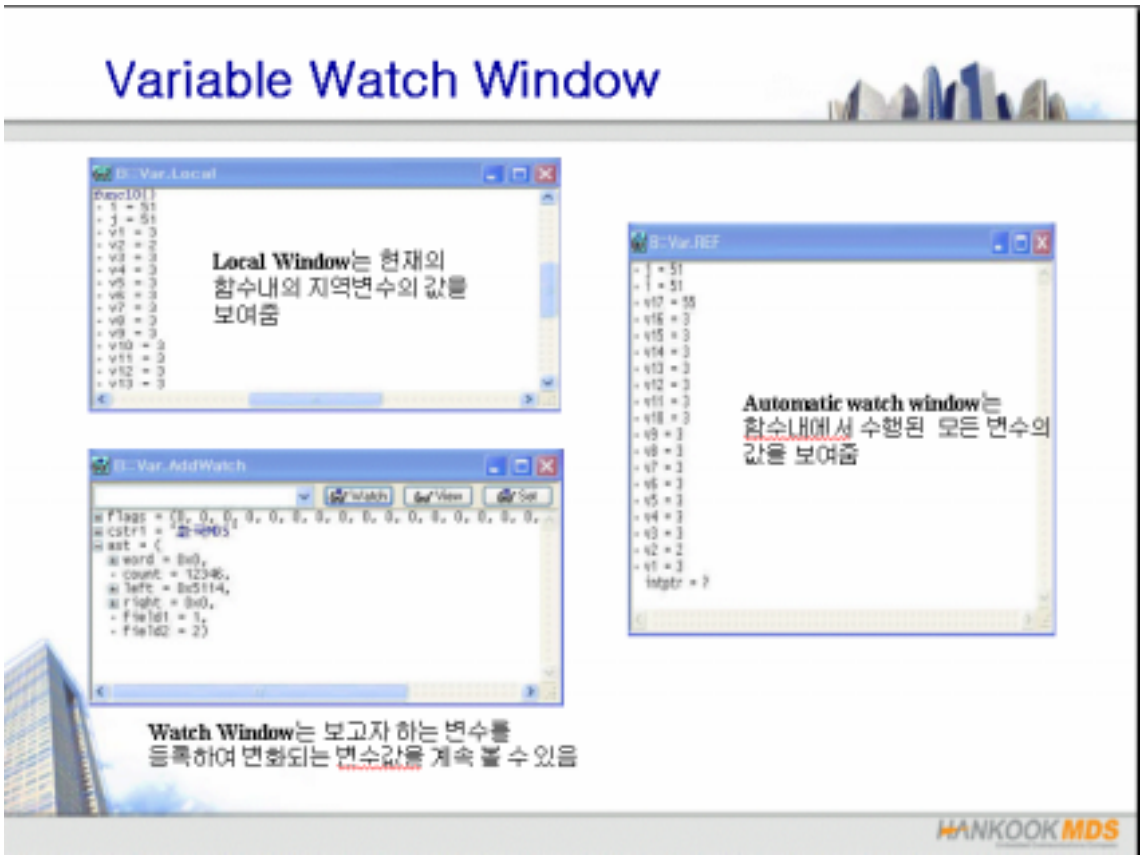
기본적으로 현재의 스택의 내용을 보여 준다. 여기에서 함수들의 nesting구조를 볼 수 있다. “locals” 옵션을 통해서 각각의 함수내의 로컬 변수값을 볼수 있으며, “caller” 옵션을 통해서 각각의 함수들의 상호 호출 관계를 볼 수 있다.

그리고, UP/DOWN button을 통해 레지스터 복사/복원기능을 구현하여 시간절약하는 디버깅 기술을 이용할 수 있다.

(UP) button : 현재의 모든 레지스터값을 저장하고, 현재 함수를 모두 수행하고 빠져 나간다.

(DOWN) button : UP한 횟수만큼 DOWN하여 UP하기 이전의 상태로 돌아간다.

2.5 Variable Watch 윈도우



2.5.1 Var.Watch 윈도우

관련 Command :

Command : Var.Watch

Format : Var.Watch [%<format>] [<variable>] ...

설명 :

모니터링하고자 하는 변수를 등록해 놓고 볼 수 있는 윈도우. 소스윈도우와 심벌 리스트 윈도우에서 선택한 변수를 watch window상에 drag & drop하거나, pop-up 메뉴에서 add to watch window를 실행하면 된다.

이름을 알고 있는 변수라면, watch window상의 텍스트박스에 직접 입력해도 좋다.

Watch window은 디폴트로 일반 변수의 경우는 십진수, 포인트의 경우는 헥사값으로 디스플레이한다. 사용자가 선택한 변수를 다른 포맷으로 보고 싶은 경우에는 변수의 popup 메뉴에서 Format 메뉴를 선택하면 된다.

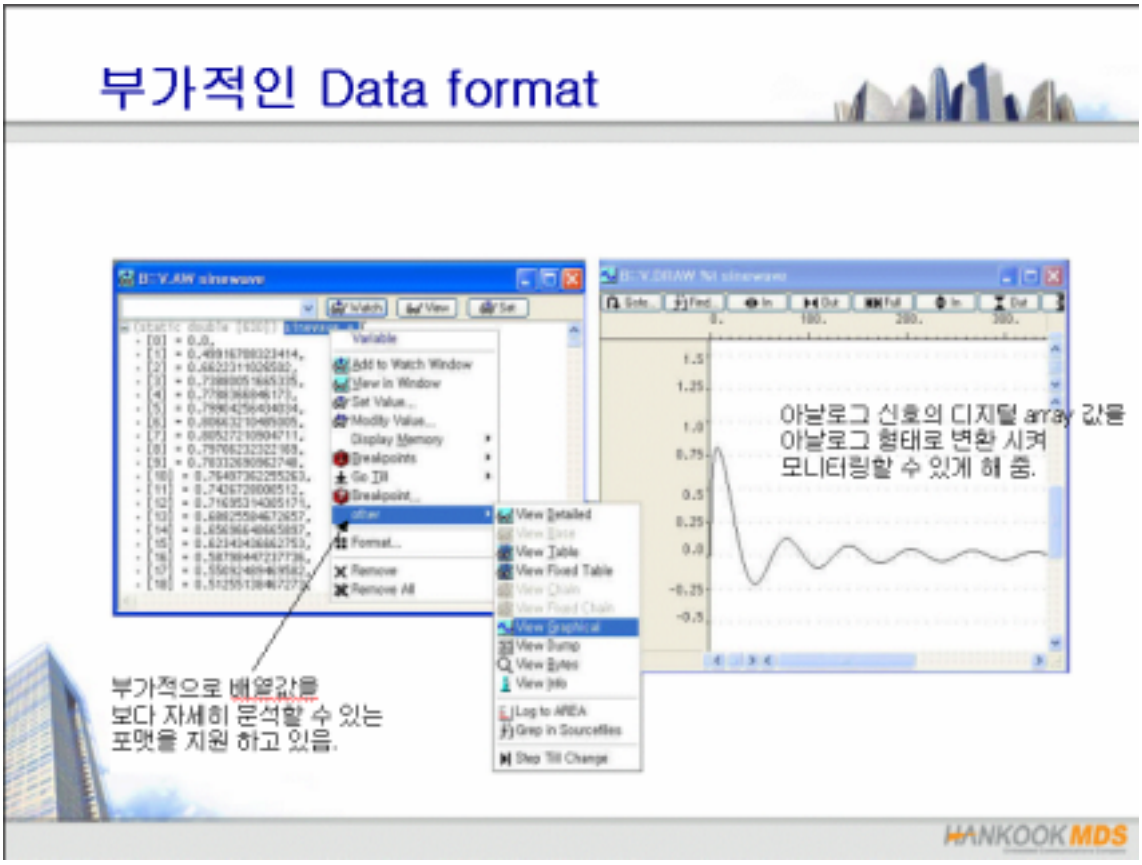
Format 설정 윈도우는 radix(변수 포맷), display, format, pointer, recursive, other options등을 설정하여 사용자 편의에 따라 원하는 형태로 볼 수 있게 도와 준다.

특히 ,recursive기능과 other options중 “E”는 주목할 만하다. Recursive기능은 2중 포인터, 3중 포인터의 경우나 구조체 변수의 element중 구조체 포인터가 있을 경우 linked list

형태로 포인터의 포인트, 포인터의 포인터의 포인터처럼 계속 추적하여 해당하는 값을 한 윈도우상에서 한꺼번에 볼 수 있게 해 준다.
한편 “E”라는 옵션은 타겟이 동작하고 있을 때 변수의 변화값을 볼 수 있게 제공해 주는 기능이다. 보통의 JTAG debugger의 경우는 타겟이 동작하고 있을 때에는 변수의 변화 유무를 볼 수 없다. 타겟 CPU를 jtag 명령을 통해 idle상태로 빠지게 한 연후에 비로서 타겟의 메모리 혹은 변수 값을 읽어 올 수 있다. 그렇기 때문에 타겟 동작중 보고 싶은 메모리 또는 변수 값이 있다면 거의 불가능하였다. 그러나, PowerView의 경우는 System 윈도우에서 cpu access를 enable해 놓고 “E”라는 옵션을 주면, 타겟은 동작하고 있을 때에도 변수/메모리값을 읽어올 수 뛰어 난 기능이 있다.
그리고, popup 메뉴중 log to area을 선택하게 되면 debugger가 멈추는 시점에서 변수값을 자동으로 AREA윈도우에 로깅하여 중요변수의 변화 history도 저장하면서 볼 수 있다. 참고로 AREA윈도우는 command line에서 “area”를 입력하고 엔터키를 치면 볼 수 있다.

어떤 변수의 경우는 보고 싶은 포맷은 정해 져 있는 경우는 ,일반 watch window대신에 특정 포맷 전용 watch window를 만들 수 있다. 예를 들면 처음부터 binary format으로 보여주는 윈도우를 만들기 위해서는 다음과 같이 하면 된다.
Command line에서 var.watch %binary하고 엔터를 치면, 그 윈도우는 default property가 binary인 윈도우가 되는 것이다.
만약, 보고 싶은 변수가 늘 정해 져 있어 아예 그 변수값을 윈도우 화면과 더불어 저장해 놓고 계속해서 불러내고 싶을 경우 다음과 같이 하시면 된다.

Command line에서 var.watch %format 변수name을 입력하게 되면, 그 변수에 해당하는 변수 전용 윈도우 또한 만들 수 있다.



2.5.2 Automatic watch 윈도우

관련 Command :

Command : Var.Ref

Format: Var.Ref [%<format>] [/Track]

설명 :

현재의 수행 소스라인에 의해 참조된 모든 변수 및 변수의 값을 자동적으로 보여주는 창이다. 마찬가지로 변수에서 pop-up메뉴에서 format창을 띄우면 다양한 포맷 및 설정으로 그 변수값을 볼 수 있다.

command line에서 var.ref 라는 명령에 의해서 불러올 수 있음.

2.5.3 Local 윈도우

관련 Command :

Command : Var.Local

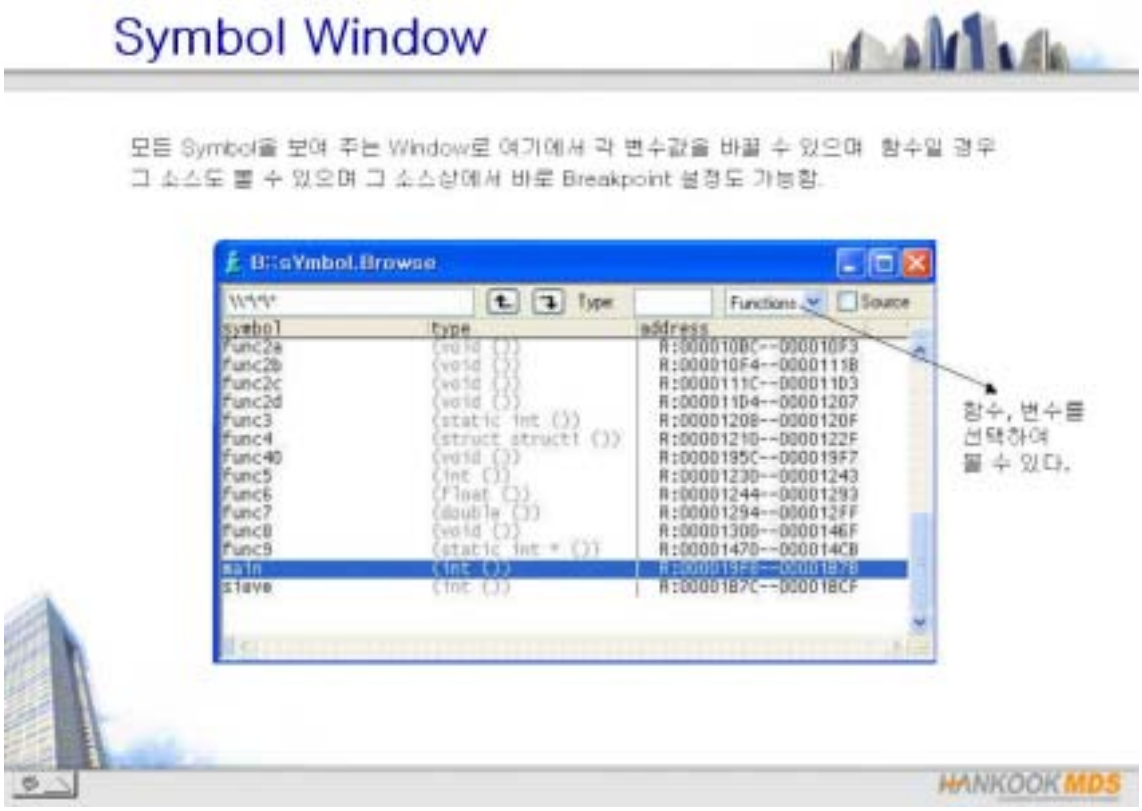
Format: Var.Local [%<format> ...]

설명:

현재 프로그램 카운터가 있는 함수와 그 함수 내에 속해 있는 local 변수값을 자동적으로 보여 주는 윈도우이다.

나머지는 watch윈도우/automatic watch윈도우와 마찬가지이다.

2.6 Symbol 윈도우



관련 Command :

Command : Symbol.Browse

Format : Symbol

Command : sYmbol.SPATH Define source search path

Format : sYmbol.SPATH [+ | - | -- | <directory>] ...

설명 :

elf/aif format 파일과 같은 절대파일(absolute file)은 symbol information을 가지고 있다. 이 파일을 다운로드하면 PowerView는 사용자들이 알기 쉽게 symbol table을 만들어 준다. 이 심벌 테이블에서 모듈별(함수/변수), 함수,변수별로 쉽게 검색이 가능하다.

여러 개의 absolute file들을 multiloadling할 경우 선택한 absolute file별로 검색이 가능하다.

2.7 Register 윈도우

관련 Command :

Command : Register.view

Format : Register /SpotLight

Command : Register.Set

Format: Register.Set <register> [<value>]

<register>: R0 , R1 ,R2 ,.....

Command : Register.Reset

Format : Register.RES

설명 :

Trace32의 PowerView은 register값들을 저장할 수 있는 기능이 있다. Register값을 저장할 수 있는 기능을 대단히 중요하다. 이 기능을 적절히 잘 이용하면, target에 critical bug가 있어 target이 자주 리셋되거나 다운되는 위치를 찾는데 유용하다.

많이 사용되는 기능으로 register copy/swap, undo라는 기능이 있다.

Register copy/swap기능은 다음과 같다.

특정한 어느 소스 라인에서 register copy기능을 수행하고 디버깅을 해 가다가 문제가 생기거나 디버깅 과정중 다시 그 위치로 돌아가서 다른 방법으로 디버깅해 보고 싶은 경우

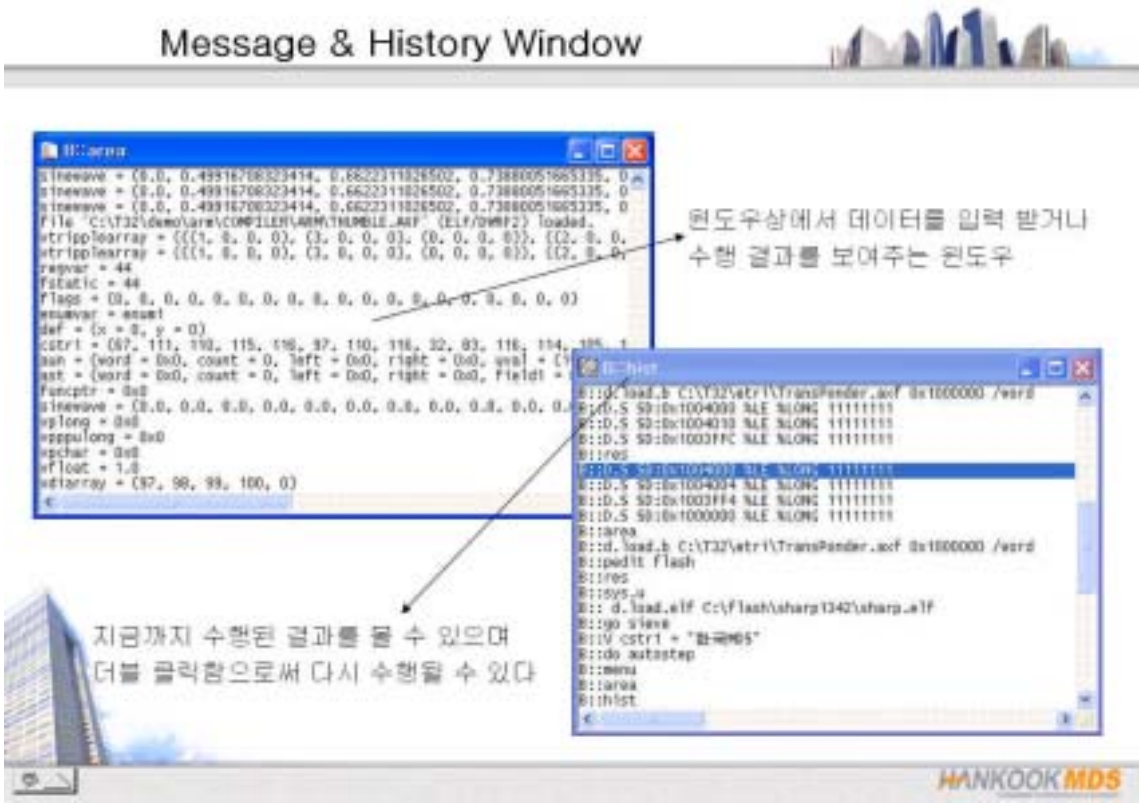
처음부터 시작할 필요없이 register swap기능을 수행하면 register copy했던 그 소스 지점으로 돌아갈 수 있다. 이 경우 단순히 프로그램카운터만을 그쪽으로 옮긴 것이 아니라 모든 core register값을 다 복원시켜 주어 유용하다. 단, 메모리쪽은 복원이 되지 않는 점에 주의 해야 한다.

이에 반해 Register undo기능은 어떤 라인을 수행한 후, 다시 수행되기 이전으로 돌아가고 싶을 때 register.undo기능을 수행하면 된다.

이 경우 바로 앞시점 뿐만 아니라 과거10단계의 디버깅과정의 위치를 기억해 준다.

그러므로, undo를 10번 수행하면, 디버깅과정중 과거로 10번째전에 멈추었던 지점으로 돌아갈 수 있다. 앞에서 잠깐 설명했듯이 macro기능중 auto stepping기능을 함께 사용했을 때 타겟이 리셋되거나 다운되는 시점까지 찾아 낼 수 있다.

2.8 메시지 윈도우



관련 Command :

Command : AREA

Format: AREA.CLEAR [<area>]

Format: AREA.Create [<area> [<columns>] [<lines>]]

Format: AREA.OPEN <area> <filename>

Format: AREA.RESet

Format: AREA.Select [<area>]

설명 :

메시지윈도우는 메시지 라인에 디스플레이되는 정보를 자동으로 로깅해 준다.

로깅되는 정보로는 에레메시지, 변수를 클릭했을 경우는 변수의 값, 다운로드 정보등을 자동으로 저장하여 보여준다.

더하여 print문을 통해 디스플레이되는 정보 또한 저장해 준다.

예를 들어 자동으로 스텝하면서 a라는 변수값을 로깅하고 싶은 경우

EX 2.8.1-1 자동으로 스텝하면서 a라는 변수값을 로깅

```
start:
step
screen
print “a=” a
goto start
```

를 수행하면서 area창을 열어보면, a라는 변수가 어떤 추이로 변해가는 지를 볼 수 있다. 단, area윈도우 속성상 최근 80개의 데이터만을 보여 주기 때문에, 과거의 모든 변화 추이를 저장하고자 할 때는 다른 명령을 수행해야 한다.
즉, area.open이라는 명령을 써서 파일로 저장해야 한다.
여기서 디폴트 메시지윈도우가 아니라, 사용자 정의 메시지 윈도우를 만든 후 위의 변수의 변화 추이를 완전히 파일로 저장하는 방법에 대해서 예를 들 것이다.

```
Area.create log ;log라는 메시지 윈도우를 만든다.
Area.select log ;여러 개의 메시지 윈도우 중에서 print문을 수행했을 때 디스플레이
되는 메시지 윈도우를 선택한다.
Area.open log log.txt ; 모든 변화 히스토리를 log.txt파일로 저장
Area log ; log메시지윈도우를 화면에 띄운다.
start:
step
screen
print “a=” a
goto start
그리고 마지막으로 저장을 끝마치고 싶은 지점에서
area.close log ; open된 area윈도우를 closing한다.
```

그 후, log.txt파일을 열어보면, 해당되는 변수의 과거의 모든 히스토리를 볼 수 있다.

2.9 히스토리 윈도우

관련 Command :

Command : HISTory

Format: HISTory.SAVE [<filename>]

HISTory.SIZE [<size>]

설명 :

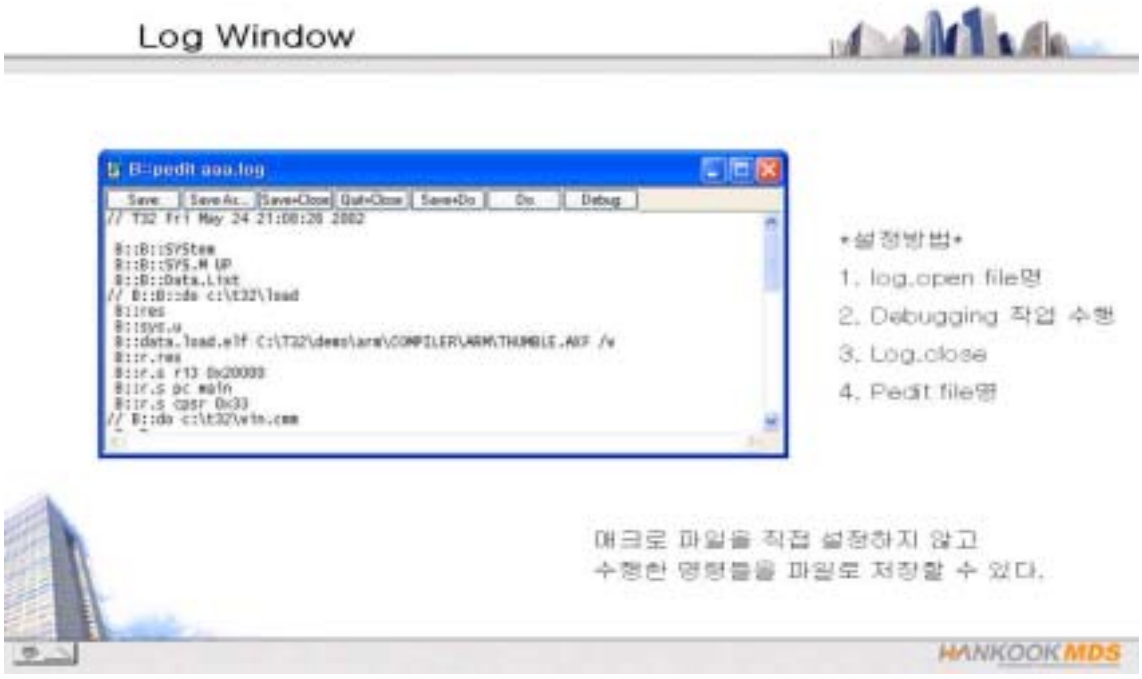
command라인에서 수행한 모든 명령어들은 자동적으로 history 버퍼에 저장되게 되는데, history윈도우를 띄운후 기존에 수행한 명령어를 클릭하면 자동적으로 똑 같은 동작을 수행할 수 있다. 즉, Unix의 history기능과 비슷하다.

참고로, 기존의 명령어 수행과정을 매크로 파일로 저장하여 사용할 수도 있다. 단, 직접적인 command나 매크로 파일에서 수행한 명령어는 저장되지 않는다.

그 내용은 다음과 같다.

History.save history.cmm

2.10 로그 파일 작성하기



관련 Command :

Command : LOG

Format: LOG.CLOSE

Format: LOG.OPEN [<filename>]

설명 :

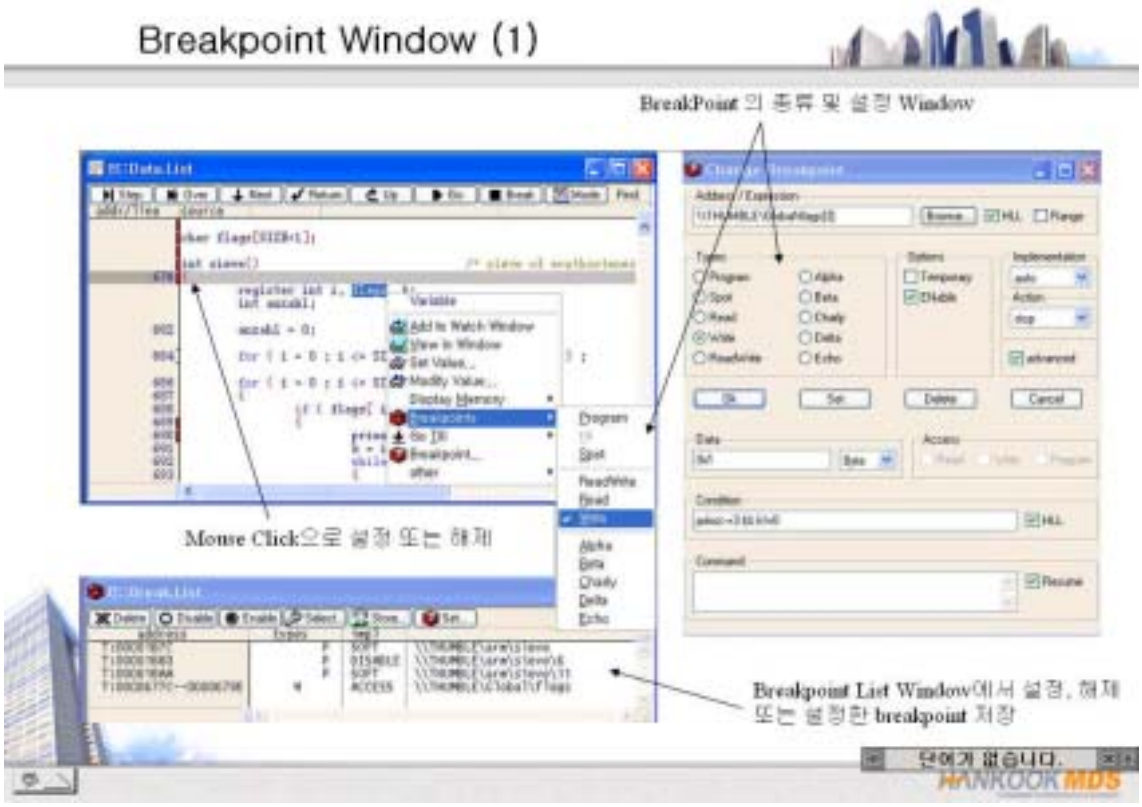
디버깅을 하다보면 매번 반복적으로 수행하는 과정이 있기 마련이다.
이 경우 PowerView는 실행과정을 logging할 수 있는 기능을 제공하고 있다.
즉, routine한 과정을 log.open한 후 수행하게 되면, 그 모든 과정이 매크로 파일(.cmm)로 저장되게 된다.

Log 사용법

```
사용방법은 다음과 같다.
Log.open karma.cmm
  ....
저장하고 싶은 과정을 수행한다.
  .....
log.close
```

이후에 do karma라고 수행하면 과거의 수행했던 것과 똑 같은 과정이 일괄 처리된다.

2.11 브레이크 포인트



관련 Command :

Command : Break.List

Format: Break.Set [<address>|<range>|<Funcname>]

Format: Break.RESet

Format : Break.List Break Browser

설명 :

브레이크 포인트는 크게 3가지로 나눌 수 있다.

장비가 지원하는 breakpoint로 hardware와 software 브레이크 포인트가 있고 칩이 지원하는 breakpoint로 onchip breakpoint가 있다. 그러나, 불행히도 hardware 브레이크포인트는 icd(jtag debugger은 모두 똑 같음)에서는 걸 수 없다.

Software 브레이크는 브레이크를 걸려는 line에 trap command(ARM의 경우는 EEDEEEDE)을 삽입하여 걸리게 된다.

CPU는 그러한 trap command을 만나게 되면 멈추면서 idle상태로 들어가게 되는데 이것이 바로 software적인 브레이크포인터로 전통적으로 많이 사용해 오던 방식이다. 브레이크포인트을 숫자 제한 없이 쓸 수 있다는 장점이 있다. 불행히도 flash/rom영역에는 이러한 소프트웨어적인 방법을 사용할 수 없다.

이에 대한 대안으로 Onchip 브레이크포인트가 있는데, 이는 칩이 지원하는 브레이크포인트 기능을 이용한 것이다. 요즈음 cpu는 칩 자체에 브레이크포인트를 걸 수 있는 로직이 들어 있다. 보통 명칭은 다르겠으나 ICE breaker register와 같은 브레이크포인트와 관련된 레지스터를 이용하여 onchip 브레이크포인트를 수행하게 된다.
이 경우 breakpoint은 breaker register수만큼만 걸 수 있다.
그 내용을 간략하게 요약하면 다음과 같다.

(ICD의 breakpoint 종류)

종류	Breakable AREA	Properties	Number of Break
Software	RAM	Address	Unlimited
Hardware	FLASH/ROM	Address/Data	Unlimited
Onchip	FLASH/ROM	Address/Data	Breaker register 수

2.11.1 ICD BreakPoint 종류

2.11.1.1 program 브레이크포인트

address line에 직접 거는 브레이크포인트로서 C source line 또는 assembly line에 모두에 걸 수 있다. 이 경우 프로그램 수행중 브레이크포인트를 건 라인을 수행하기 바로 전에 브레이크포인트가 걸리기 때문에 synchronous 브레이크포인트라고 부른다.

2.11.1.2 Variable/memory의 read/write 브레이크포인트

지정한 변수 혹은 메모리에 특정한 값을 read/write 과정을 수행한 후에 멈추게 된다. 이 경우는 read/write한후에 브레이크포인트가 걸리므로 asynchronous 브레이크포인트라고 부른다.

한편, condition을 설정할 수 있어 다양한 조건에서 브레이크포인트를 걸 수 있다.

예를 들면, a에 3을 write하는 순간에 b라는 변수 값이 3보다 작을 때 브레이크를 걸고 싶다면, break.set윈도우에서 advanced를 클릭하고 data영역에는 3을 값을 condition부분에는 b<3이라고 입력하면 된다.

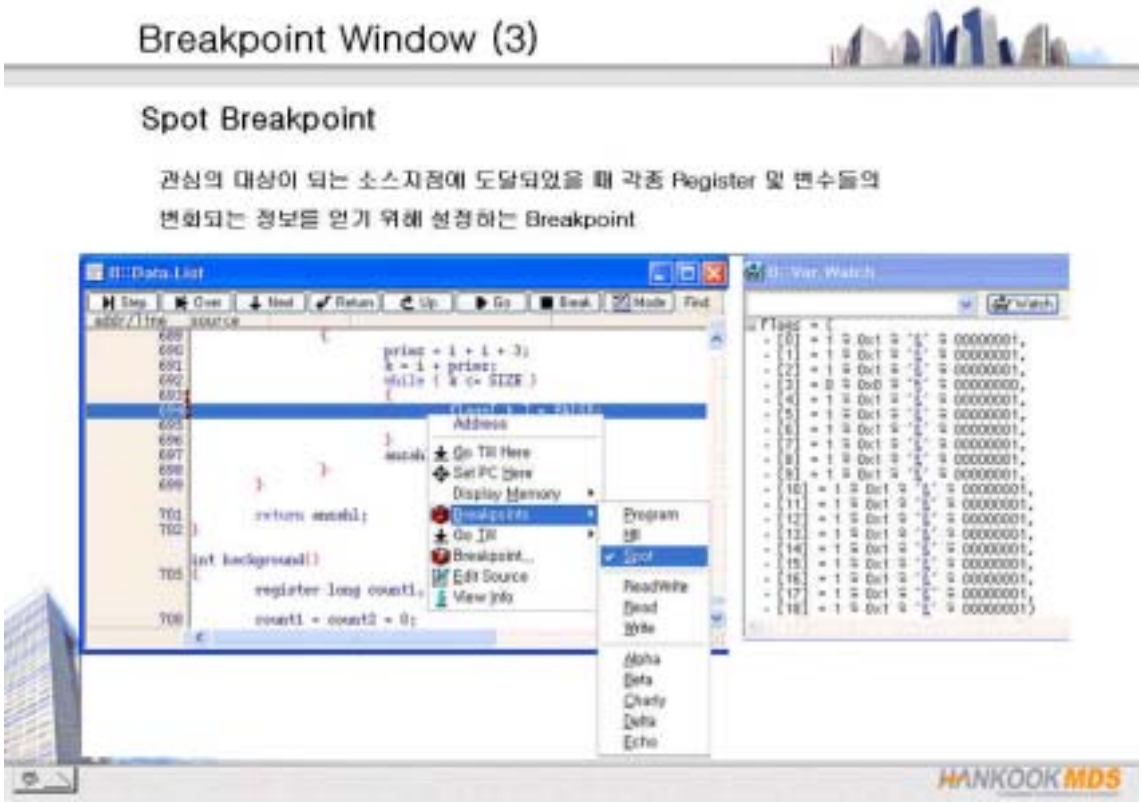
만약 command line에서 수행하고자 한다면 다음과 같이 하면 된다.

EX 2.11.1.2-1 Command line에서 변수에 Break point 셋팅

```
v.b.s a / WRITE /data.byte 3   /VARCONDITION b < 3
```

셋팅후에 Break.List 창에서 Break가 등록되어 있으면 정상적으로 셋팅이 된것임.

2.11.1.3 Spot 브레이크포인트



스크린상에 디스플레이된 정보는 CPU가 프로그램 수행을 멈추고 debug mode로 바뀔 때에만 디폴트로 업데이트 된다. 만약 어느 특정한 시점에서 중요한 변수/메모리/레지스터값이 어떻게 변하고 있는 직접 확인하고 싶은 경우 spot 브레이크를 걸 수가 있다. 보통 디버그 인터페이스의 속도와 보고자하는 정보의 양에 따라 달라지겠지만 50 milisecond내지 100 milisecond가 걸린다.

Spot 브레이크포인트가 걸리는 시점에서 중요한 변수의 히스토리를 로깅할 수도 있다.

예를 들면 다음과 같다.

만약 특정한 라인에 spot 브레이크포인트를 건 후 다음의 macro file을 수행하면 sunny라는 변수의 변화 히스토리를 파일로 저장할 수 있다. 물론 이 경우 spot 브레이크포인트가 걸려 있는 시점에서 변수의 히스토리라는 점에 주목한다.

EX 2.11.1.3-1 Spot Break이용해서 변수 파일로 로깅하기

```
Area.create log
Area.select log'
Area log
Area.open log log.txt

Var.log sunny /area log /ontime 1.s /onspot /timestamp
---------
일정한 시간이 경과한 후 다음의 명령어들을 수행하여 log을 close한다.

Var.log
Area.close log
```

마지막으로 log.txt파일을 열면 sunny라는 변수의 1초마다의 상태값을 볼 수 있다.
만약 /onchange라는 옵션을 주면, spot breakpoint가 걸려있는 시점에서 값이 변하는 경우에만 저장할 수도 있다. /timestamp라는 옵션은 로깅할 때마다 걸리는 시간을 측정하여 보여준다.

2.11.1.4 Alpha/Beta 브레이크 포인트

Bit단위로 브레이크포인트를 걸거나 두개의 address selector를 가지고 조건 브레이크(A or B; A and B; B after A)를 걸고자 할 때 onchip 브레이크포인트를 걸고자 할 때 이것들을 사용할 수 있다. Alpha와 Beta 브레이크포인트상의 차이점은 없으며, 명칭상 구분하기 위해서 나누어 진 것이다.
Breaker register가 2개 이상인 경우는 Charley, Delta, Echo 브레이크포인트까지 사용할 수 있다.
Alpha/beta 브레이크포인트는 엄격히 말하면 브레이크포인트라기 보다는 address selector라고 부르는 편이 타당할 것이다.
지금부터는 2개의 breaker register를 가지고 있는 processor(예;ARM)를 예를 들어 설명하겠다.

EX 2.11.1.4-1 Command Line 에서 Alpha, Beta Break 설정

```
v.b.s a /alpha
v.b.s b /alpha
```

만약 a,b 라는 두개의 변수중의 하나라도 임의의 값을 write한 후에는 멈추고 싶다면, 다음과 같이 두개의 변수 각각에 alpha라는 address selector를 설정해 준다.

그 다음에 TrOnchip윈도우를 띄워서 원하는 이벤트에 적절하도록 A.size와 A.cycle를 선택하고 A.A 부분에서 해당하는 address selctor를 지정하고 모드(AORB;AANDB;BAFTERA)를 “AORB”로 선택하면 모든 설정은 끝나게 된다. 물론, A.Data텍스트 박스에 원하는 데이터값을 입력하게 되면, 단순하게 임의 값을 write할 때 잡는 것이 아니라, 특정한 값을 write/read할 때 브레이크를 거는 것이 가능하다.
물론 모드를 AANDB로 설정하면 두조건이 모두 만족했을 때 브레이크를 걸며, BAFTERA의 경우는 A조건이 만족된 후에 B조건이 만족될 때 브레이크를 걸게 할 수 있다.
한편 비트단위로도 브레이크포인트를 걸 수 있는데, 이 방법도 위와 비슷한 방법으로 수행할 수 있다. 단, A.Data부분을 다음과 같이 이용하면 된다.
예를 들어 alpha 브레이크포인트를 걸어 놓은 곳의 마지막 비트값을 1로 write할 때만 브레이크를 걸고 자 할 때, 다음과 같이 나머지 부분을 masking 시킬 수 있다.

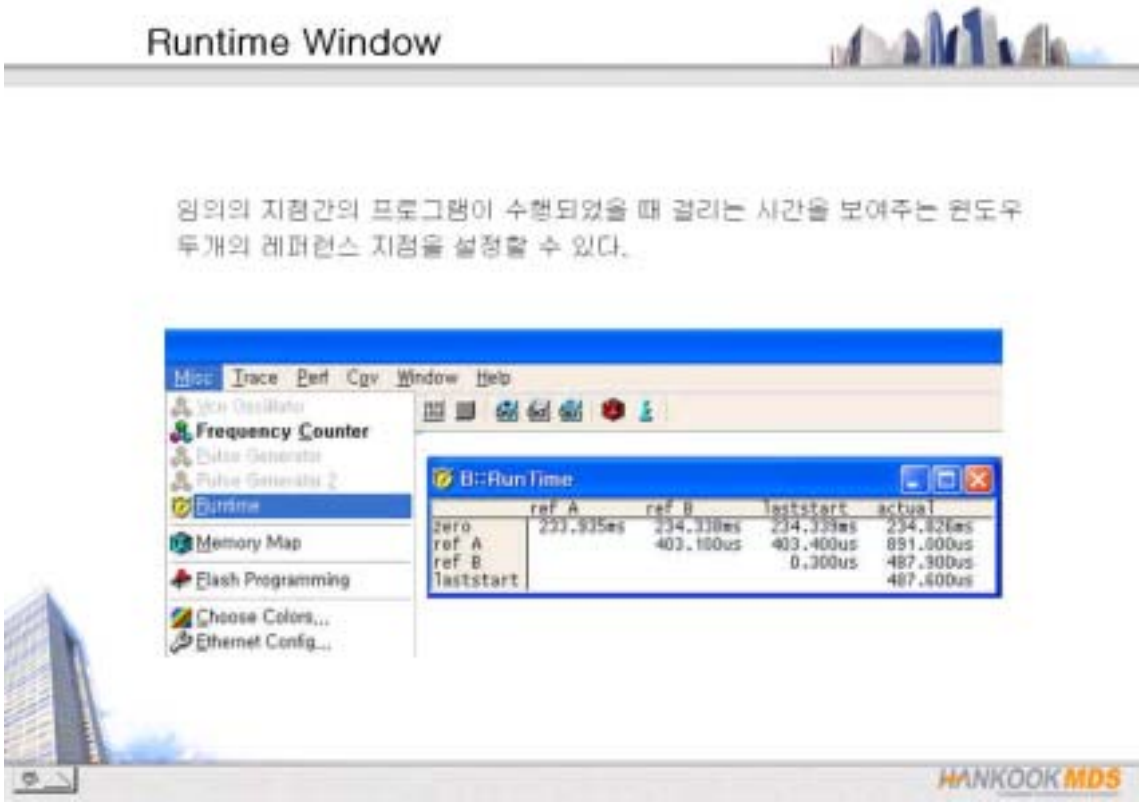
EX 2.11.1.4-2 한Bit가 Write될 때 Alpha Breakpoint 설정

```
A.data 0yxxxxxxx1 ; 여기서 x는 don’t care condition
A.size byte
A.cycle write
A.address alpha
```

다시 말해 위의 스크립파일은 alpha 브레이크포인트가 지정된 위치에 byte단위로 마직막 비트값에 high값을 writing하면 멈추게 하도록 할 수 있다. 이 경우 다른 비트값은 어떤 값은 don’t care condition이다.

2.12 Runtime 윈도우

관련 Command :



Format: RunTime.Init

Format: RunTime.refA | RunTime.refB

Format: RunTime.RESet

Format: RunTime.state

설명:

두 시점간의 프로그램 수행시간을 측정해 주는 기능이다. 타임 분해능은 100us이다. Reference A/B를 설정할 수 있어서 여러 시점사이의 시간을 서로 비교하는데 용이하다. 초기에는 zero/reference A/reference B의 actual time값이 모두 동일하다. 사용자는 이 actual time만 읽으면 된다. 현재의 위치를 reference로 두고 싶을 때 runtime.refa 또는 runtime.refb라는 명령을 주면 되며, 이 경우 현재의 위치가 reference로 잡혀서 그 시점에서 actual time이 0으로 초기화 된다. 다음 브레이크가 걸리는 시점까지의 시간은 마찬가지고 맨 좌측 reference에 해당하는 actual time을 읽으면 된다.

2.13 Performance 기능

관련 Command :

Format: PERF.Address <addressrange>

Format: PERF.Mode <mode>

<mode>: OFF, Program, TREE, LINE, Function, Module, Func, Mod, LABEL, S10, S100, S1000, DistriBution,VarState, TASK

Format: PERF.Program [<file>]

Format: PERF.Sort <mode>

<mode>: OFF, Address, symbol, Ratio

Format: PERF.state

Format: PERF.View

설명 :

관심의 대상이 되는 modules/functions/address ranges가 CPU시간을 얼마나 많이 점유하고 있는지 그 정보를 보여 주는 기능이다.

Performance기능을 수행하기 위해서는 몇 가지 사항을 설정하여야 한다.

(1) 먼저 분석하고자 하는 어드레스 영역을 "Address"란에 지정한다.
디폴트로 현재 사용중이 프로세서의 모든 메모리 영역이 자동을 설정되어 있다.

(2) 분석 모드부분을 설정한다.

- Program : 사용자가 직접 perf.p창을 열어 performance 측정을 원하는 모듈,함수, 어드레스를 직접 입력한다. 이때 commands group의 Program 버튼을 클릭하면 된다.
- TREE : Modules/Functions/Lines의 Hierarchical display
- LINE : HLL line table을 사용하여 ranges을 정의
- Function : HLL function의 분석
- Module : modules의 분석
- FuncMod : 지정된 영역에서의 함수와 그 영역 외부에서 모듈들의 분석
- S10,S100,S1000 : 10,100,1000 hex byte 단위로 screening mode

(3) sorting된 정보를 보기 위해서 sort tab에서 원하는 타입을 결정한다. 특히 이 탭에서 Ratio를 선택하고 options에서 SCAN을 선택하면, PowerView 소프트웨어가 자동적으로 가장 CPU시간을 많이 점유하고 있는 32의 item을 browsing하여 보여준다.

(4) List button을 클릭하여 Performance정보를 본다.
이 정보중 watchtime의 전체 프로그램이 수행하는데 걸린 시간이고 time은 각각의 item이 수행할 때 걸린 시간이다.

(참고) Gate란에 설정한 시간은 측정 interval을 나타내며 디폴트 값은 1초이다.

Chapter 3. PowerView의 고급 기능

3.1 문서화하는 방법

디버깅을 하다보면, 저장하고 나중에 분석하기를 원하는 정보가 있다. 예를 들면, go,break과정을 반복하면서 중요한 스택이나 레지스터 윈도우를 저장하거나 특정한 메모리 영역을 인쇄하여 보고 싶은 경우에 다음의 documentation하는 방법을 알면 도움이 될 것이다.
문서화하는 방법은 다음과 같다.

지정하는 방법은 File메뉴의 Print..서브 메뉴의 Printer FileType이라는 아이템을 선택하면 저장할 파일의 포맷을 지정하는 윈도우 창이 디스플레이되는데, 이때 원하는 포맷을 클릭하여 지정하면 된다. 아니면 다음과 같이 커맨드라인에서 명령을 직접 입력해 도 좋다.

```
Printe.filetype ascii      ; 만약 ascii포맷으로 저장하고 싶을 경우
```

이제 저장될 파일 포맷이 정해졌으므로 어떤 파일 이름으로 저장할 것인가를 지정하면 된다.

```
Printer.open source.txt
```

반드시 저장을 끝마치고자 하는 시점에 printer.close라는 명령을 주어야 한다. 마지막으로 저장하고자 하는 윈도우를 지정해야 한다.
참고로, 소스윈도우를 화면을 띄우고자 할 경우는 “data.list”라는 커맨드를 주면 된다.
만약 이 소스윈도우를 파일로 저장하고 싶다면, WinPrint(약어로 wp)라는 명령을 앞에 추가하면 된다. 즉, “wp.data.list”라고 명령을 주면 이 내용은 화면에 디스플레이되는 것이 아니라 앞에서 지정한 파일로 자동적으로 저장된다.

3.2 문서화 예제

EX 3.2.1. 0번지부터 0xfffff번지까지의 C source와 assembly source를 ascii 포맷으로 source.txt라는 파일로 저장하고 싶은 경우

```
printer.filetype ascii
printer.file source.txt
wp.d.l 0x0—0xfffff
```

EX 3.2.2. 100 miliseconds간격으로 자동적으로 go/break하면서 stack window내용과 Register 윈도우를 저장하는 방법

```
printer.filetype ascii
printer.open stack.txt
wp.var.frame /l /c
wp.r /spotlight
```

EX 3.2.3. memory 내용을 text file로 저장하는 방법과 binary file로 저장하는 방법

```
* text file로 저장할 때
    printer.filetype ascii
    printer.file memory.txt
    wp.d 0x0—0xfffff

* binary file로 저장할 때
   data.save.b binary.bin 0x0—0xfffff
```

3.3 Complex debug control

디버깅하는 과정에서 자동으로 어떤 특정한 레지스터값이 일정한 값을 가질 때 까지 running시키거나 스텝을 하고 싶을 경우나 루프문에서 몇번째 루프문을 수행했을 때 프로그램을 멈추게 하고 싶은 경우, 아니면, 변수값이 특별히 어떤 값을 갖을 때까지 스텝하고 싶은 경우에 간단한 명령만을 이를 수행할 수 있다면 매우 편리할 것이다. Trace32 PowerView는 바로 이런 기능을 가능하게 해 준다.

크게 step 기능과 go 기능으로 나눌 수 있다. 이러한 기능은 주로 변수,레지스터, 메모리

에 대해서 비교하여 사용할 수 있다. 단, 비교대상이 변수일 경우는 step,go 명령앞에 “Var.” 을 추가한다.

EX 3.3.1. step 기능을 이용한 break

```
Step.Change r(r3)                          ; register r3값이 변할 때까지 스텝
Step.Till data.byte(d:0x10000) > 4         ;메모리 0x10000값이 4보다 클 때까지 스텝
Var.step.till sunny[3] >1 && sunny[4]==3 ; sunny[3]이 1보다 크고 sunny[4]가 3과 같을 때까지 수행
(참고) && : AND조건; || : OR 조건
```

go 기능

JTAG debugger은 target이 running중에는 정보를 읽어오거나 비교을 해 볼 수 없으므로, 비교하기에 중요한 시점에 반드시 breakpoint를 걸고 수행해야 한다. 그 다음 go명령을 수행하면 조건을 만족할 때까지 프로그램은 running하게 된다.

만약, 어드레스 0x10000의 instruction을 수행한 순간에 특정한 메모리,레지스터,변수값이 변하거나 어떤 값을 가질 때까지 수행하고 싶다면 다음과 같이 설정하면 된다.

```
Break.set 0x10000
```

EX 3.3.2. go 기능을 이용한 break

```
Var.Go.change sunny[3]   ; 0x10000번지의 instruction을 수행하는 시점에서
                           sunny[3]가 변할 때 멈춘다.

 Go.till r(r13)==0x1          ;이 시점에서 r13값이 1이면 멈춘다.
```

4. Multicore System

4.1 Multicore란?

시스템이 복잡하고 소형화됨에 따라서 여러 개의 core를 하나의 ASIC칩으로 집적시켜 만들어가고 있는 추세이다. 게다가 die 사이즈를 작게 하고 비용을 절감하기 위해서 테스트할 수 있는 jtag시그널을 각각의 core별로 만들어 놓지 않고 하나의 daisy chain으로 연결하여 핀수를 줄이는 경향이 있는데 이것이 바로 multicore system이다. Trace32 ICD를 사용하여 mulitcore디버깅을 하기 위해서는 다음과 같은 과정을 거쳐야 한다.

JTAG chain안에 한 개 이상의 core 가 있다면, IRPRE,IRPOST,DRPRE,DRPOST 네개의 파라미터를 이용하여 디버거에서 멀티코어 시스템 정보를 알려 주어야 한다. 이러한 정보는 system.up명령 전에 주어야 한다.

한 개이상의 ICD 디거버(이하 디버거)가 동시에 한 개의 공동 jtag port에 연결되어 있다면 Tristate가 사용되어야 한다. 메뉴중 TAPstate와 TCKLevel은 디버거가 tristate 모드로 스위치할 때 선택되는 TAP와 TCK level을 정의한다.

Notice !! **********
반드시 nTRST은 타겟상에 pullup resistor를 있어야 하며 EDBGRQ은 pulldown resistor가 있어야 한다.
****

4.2. Multicore의 설정

Format : System.MultiCore <location> <number>

<location> IPPRE, IRPOST, DRPRE, DRPOST, TriState, TAPState, TCKLevel, Slave

IPPRE : (default 0) ARM core와 TDO signal사이 JTAG chain안에 있는 모든 코어의 인덱스 레지스터 비트의 수

IRPOST : (default 1) TDI와 ARM core사이 JTAG chain안에 있는 모든 코어의 인덱스 레지스터비트의 수

DRPRE : (default 0) ARM core와 TDO 사이 JTAG chain안에 있는 코어의 수(Bypass mode에 있는 코어당 1개의 한 개의 데이터 레지스터 비트)

DRPOST : (default 0) TDI와 ARM core사이 JTAG chain안에 있는 코어의 수

TriState : (default off) 한 개 이상의 디버거가 똑 같은 JTAG 포트을 공유한다면, 이 옵션이 요구된다. 디버거는 각각 JTAG 액세스이후 tristate모드로 스위치한다. 그때, 다른 디버거들이 그 포트에 접근할 수 있다.

TAPState : (default: 7 = Select-DR-Scan) 디버거가 tristate 모드로 갈 때 TAP controller의 상태. JTAG TAP controller의 모든 상태가 선택될 수 있다.

TCKLevel : (default:0) 모든 디버거가 tristate로 될 때 TCK signal의 레벨

Slave : (default :0) 한 개 이상의 디버거가 똑 같은 JTAG 포트를 공유할 때 한 개를 제외한 모든 것은 이 옵션이 active해야한다. 오직 한 개의 디버거(master)이 nTRST와 nSRST을 제어하도록 되어 있다.

4.3. Multicore 설정의 예제

EX 4.3.1. Multicore setting의 한 예

```
TDI-→ Core A-→ Core B-→ ARM-→ Core C-→ TDO
Instruction register length:
Core A: 3bit
Core B: 5 bit
Core C: 6 bit
위의 경우....
(settings)

SYStem.MultiCore IRPRE 6; IR Core C
SYStem.MultiCore IRPOST 8; IR Core A+B
SYStem.MultiCore DRPRE 1; Core C
SYStem.MultiCore DRPOST 2 ; Core A+B
SYStem.Up
```

EX 4.3.2. TI's ORION

```
EMU0=EMU1=0(ARM7 only) :
System.cpu GEMINI
System.up
EMU0=EMU1=1(all three)
System.cpu GEMINI
System.multicore irpre 16. ;decimal로 표시할 때는 반드시 숫자뒤에 dot표시를 한다
System.multicore drpre 2.
System.multicore irpost 0.
System.multicore drpost 0.
System.up
```

EX 4.3 .3. TI's HERCROH20

```
Sys.cpu Gemini
Sys.multicore IRPRE 8.
Sys.multicore DRPRE 1.
Sys.u
```

EX 4.3.4. TI's Helen_DC_ES1(old version)

```
System.cpu ARM925T
Sys.mc   IRPRE 0.
Sys.mc IRPOST 46.
Sys.mc DRPRE 0.
Sys.u
```

EX 4.3.5. TI's Henlen_DC_ES2

```
Sys.cpu ARM925T
Sys.mc irpre 38.
Sys.mc drpre 1.
Sys.mc irpost 8.
Sys.mc drpost 1.
Sys.u
```

EX 4.3.6. TI's OMAP1510DC

```
sys.mc irpre 0x38
sys.mc drpre 0x01
sys.mc irpost 0x08
sys.mc drpost 0x01
sys.cpu arm925t
sys.up
```

5. Flash programming

5.1 Flash programming의 종류

마이크로컨트롤러의 외부 flash 메모리와 내부 flash메모리은 flash command에 의해서 프로그램되거나 지워질 수 있다.

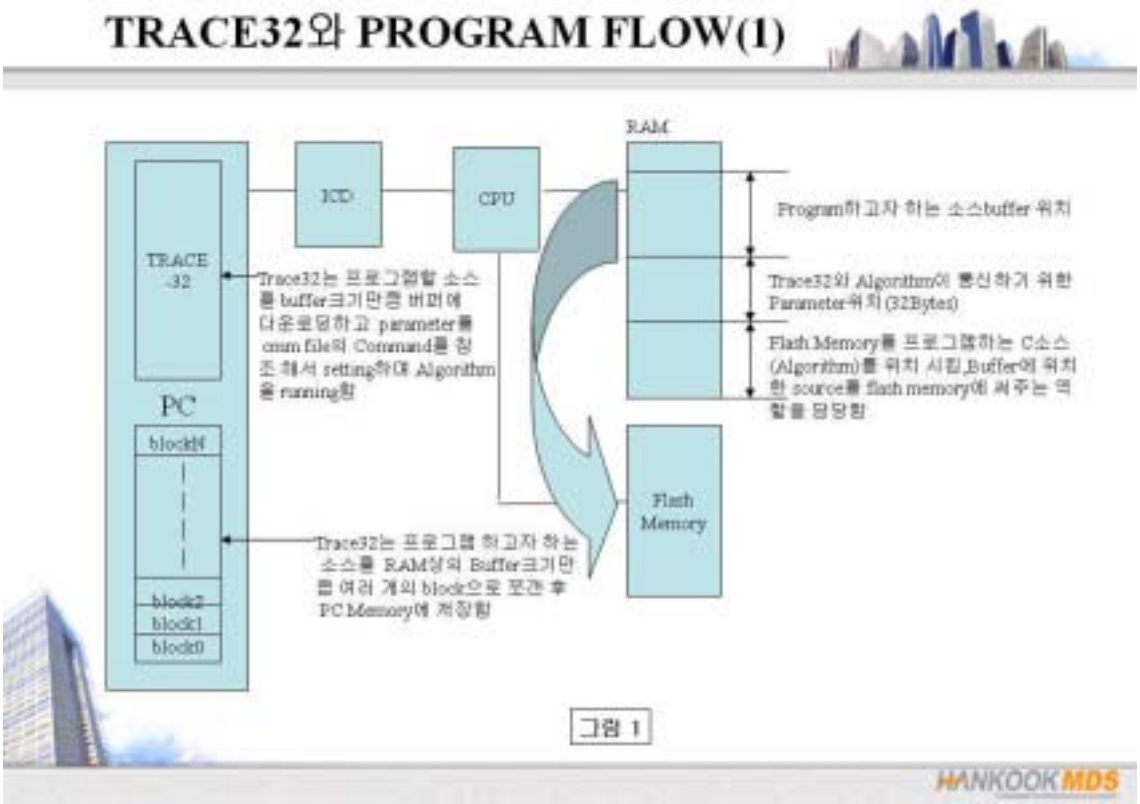
Target이 여러 개인 경우 플래시 프로그래밍 또한 병렬로 지원되며, 4개까지 가능하다.

Flash 프로그래밍은 두 가지 모드로 수행될 수 있다.

즉, ICD를 통해서 직접 프로그래밍하는 JTAG 방식과 flash 알고리즘을 이용하는 target base 방법이 있다.

JTAG방식의 경우는 타겟상에서 running하는 어떠한 프로그램도 필요로 하지 않으며, 보통 전형적인 다운로드 속도는 초당 1내지 5Kbyte 정도로 느리다. 단, 예외로 Intel FlashFile과 같은 buffered flash 메모리의 경우는 초당 100K로 다운로드 할 수 있다.

한편, target base 방법의 경우는 사용자가 작성한 플래시 알고리즘을 이용하여 프로그래밍을 한다. 이 루틴은 메모리를 프로그래밍하기 위해 적절한 parameter를 갖은 Trace32에 의해서 호출된다. 이 방법은 JTAG방식에 비해 속도가 빠르다.



TRACE32와 PROGRAM FLOW(2)

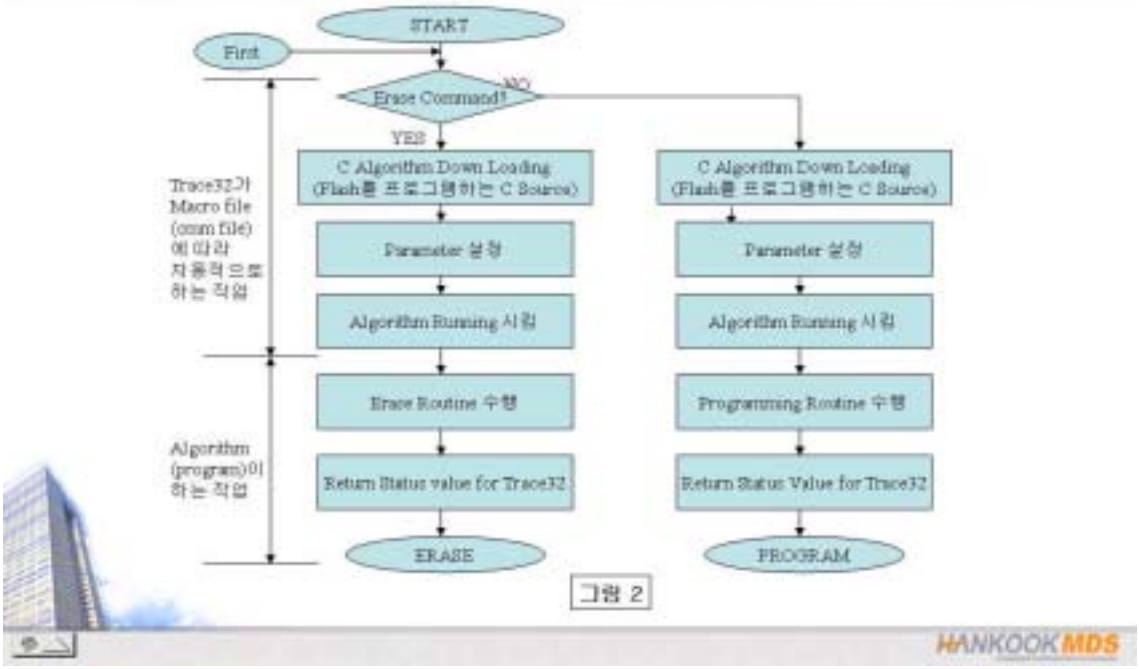


그림 2

HANKOOKMDS

TRACE32와 PROGRAM FLOW(3)

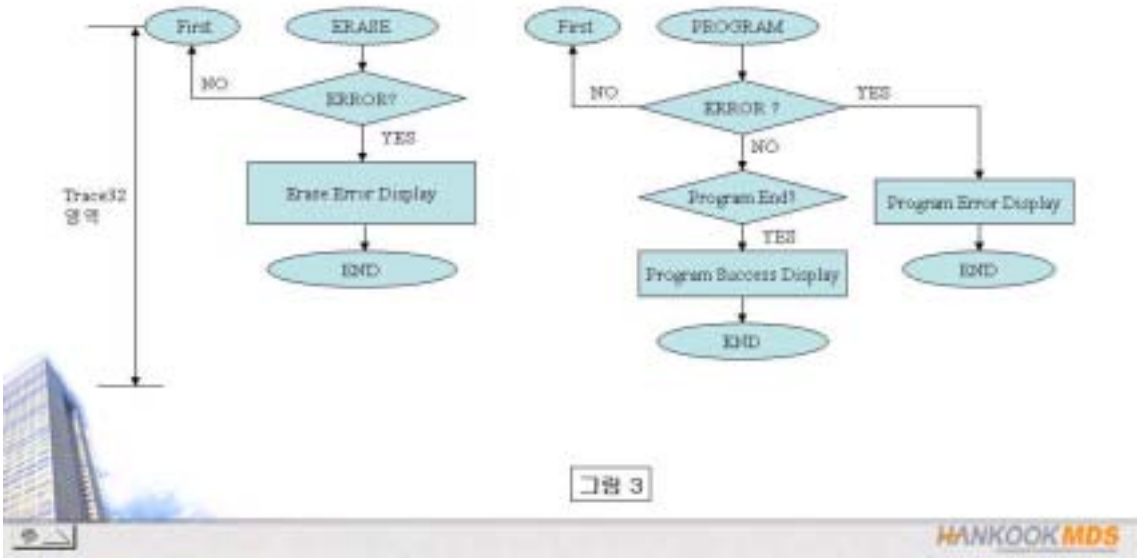


그림 3

HANKOOKMDS

가장 빠르게 프로그래밍하기 위해서는 바이너리파일을 다운로드하는 것이 좋다.

5.2 JTAG 방식의 사용 예

```
Flash.res      ; flash memory 설정을 리셋
Flash.Create [flash physical range] [family code(지원 플래시는 도움말 참조) WORD
Flash.Erase all  ;플래시 메모리 erase
Flash.Program all ; 플래시 write시작 명령
Data.load.b a.bin 0x0 /w ; 플래시에 write할 코드
Flash.Program off ; 플래시 write 종료
```

5.3. Target base 방식 사용 예

```
[RAM;ROM 초기화]
data.load.elf flash_algorithm.elf  ;플래시 알고리즘을 RAM 번지에 다운로드
                                   (예제에서는 0x1000000번지 다운로드)
break.set swbp ; main루틴을 수행후에 멈추는 시점
flash.res  ; 플래시 설정을 리셋한다.
Flash.target 0x1000000 0x1008000 0x1000
             (플래시 알고리즘이 다운로드된 번지;플래시 파라미터 번지; 버퍼사이
                                   즈)
flash.erase all ; 플래시 파라미터의 여덟번째값을 5로 설정,erase 부분 수행
flash.program all ; 플래시 파라미터의 여덟번째값을 1로 설정, write 부분 수행
data.load.elf  user_source.elf /w
flash.program off ; 플래시 프로그램을 종료시킨다.
```

5.4.Target base 방식에 사용되는 <flash.h> <param.c> 파일에 정의되고 선언 된 parameter의 정보

5.4.1. parameter 종류

- parameter0: flash memory physical한 address정보
- parameter1: flash memory operation type (ex: 8bits or 16bits)
- parameter2: cpu bus operation type
- parameter3: reserved
- parameter4: 현재 프로그램 할 block start address
- parameter5: buffer size (프로그램 함 source를 저장할 buffer)
- parameter6: status정보 (Algorithm stop시 여기에 값은 program하다 error난 address가 들어있게 됨. Error addr은 Algorithm 에서 넣어 주어야 함)
- parameter7: status정보
  (Algorithm running시 값이 1이면 program, 값이 2면 erase ; 프로그램 수행 전 command)
  (Algorithm stop시 값이 0x00이면 성공이고, 값이 0x100이면 실패 ; 프로그램 수행 후 status)

5.4.2. Parameter의 역할: flash program을 위한 macro file의 flash program에 관련된 command들을 참조하여 Trace32가 자동으로 설정

5.4.3. Parameter의 내용: Flash Erase/Program 관한 정보, flash memory 의 physical address, 몇 bits bus operation memory 인지에 관한 정보, 현재 프로그램 할 block start address에 관한 정보를 가지게 된다.

5.5. Flash 알고리즘 예제

예제 구성

- 16BITS BUS OPERATION 을 기준
- Compile 해서 같이 link 한 파일은 sharp.c 와 jfstart.s 및 param.c file 인데 flash.h file 을 참조 했다.
- parameter 구조체는 flash.h 에 정의 되어있고 그 변수 선언은 param.c 에서 정의 했다
- 특정한 번지(사용자가 지정한 번지)에 module 을 위치 시키기 위해 moonprj.scl 파일을 참조해서 link
- scl 을 살펴보면 jfstart.s 와 intel.c 는 1000000 번지에 위치 시키도록, param.c 는 1008000 번지에 위치하도록 했는데 앞에서 살펴본 cmm file 에서 command 를 그렇게 사용했으므로 이러한 map 을 잡게 된 것이다.
- 만약 command 를 그렇게 주고 다른 번지로 link 하게 되면 Algorithm 은 trace32 가 준 정보를 정상적으로 가져오지 못하기 때문에 실패하게 된다.

< flash.h >

```
struct type_jtag_flash_param {
unsigned int flashaddr;      //parameter0,flash memory physical start addr
unsigned int flashwidth;   //bus width
unsigned int width;              //cpu bus width
unsigned int offset;              //reserved
unsigned int addr;                // first address of the flash to write/erase
unsigned int size;                //   size of data to write/erase
unsigned int status0;         //program error시  error가 발생된 addr을 return할 인자
unsigned int status;   //running시: 1=program, 2=erase, stop시: 0=success, 0x100=fail
unsigned short   data[0x1000];   // give 128K worth of buffer, 실제로 trace32는 최대
                                  //1000만을 인식해서 사용함
```

< jfstart.s >

```
    msr       CPSR_c, #PSR_Supervisor:OR:PSR_Irq_Mask:OR:PSR_Fiq_Mask
    ldr       r13, =svc_stack+SVC_Stack_Size
    ldr       a1, =main
    blatox    a1      ;main function호출
    swbp      DCD       0xEEDEEEDE
    ;main이 끝나고 결국은 swbp를 수행하게 되는데 이 EEDEEEDE라는
    ;값은 MOTOROLA MPU의 TRAP명령과 흡사한 명령인데,
    ;Algorithm은 stop하게 된다.그러면 실제로 제어권은 trace32에게 넘어 가게 되고
    ;앞에서 설명한 parameter check라던가  buffer에 data down load등을 수행하고 다시
    ;프로그램을 running시킨다.
```

< sharp.c >

```
  #include "flash.h"
  #define   NUM_FLASH_BLOCKS   39       //flash memory block수
  void unlock(void);          //sharp flash memory는 power up 될 때 자동으로 lock되어 있
                                으므로 프로그램을 하기 위해서는 항상 unlock을 해야 한다
  typedef struct
  {
    unsigned int start;       // First address of the block : INTEL은 block별로 erase를
                                해야만 하는데  그것을
    unsigned int len;           // Length of the block in bytes : 위해 구조체선언을 한 것
  } block_descpt;

  unsigned int *parameter = (unsigned int *)0x1008000;
 unsigned short *addr16,*data16;   //프로그램할 flash memory address, data를 가지고
                                    올 buffer address
  unsigned long time_out;                 //write나 erase가 안될 경우 timeout value를 위해
  unsigned short flash_status;              //flash memory status data를 위해
  unsigned int number;
  unsigned char block_num;
```

6. ETM(Embedded Trace Macrocell)

6.1 ETM이란 무엇인가?

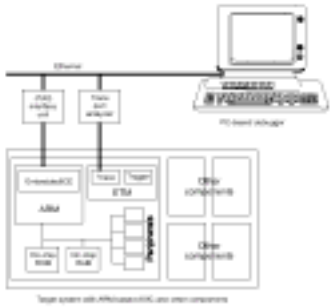
6.1.1 ETM 정의

기존의 CPU에뮬레이터의 경우는 CPU의 모든 핀들을 에뮬레이터와 인터페이스 시켜야 하기 때문에 실제 타겟에서 사용하기는 많은 무리가 따른다. 게다가 비용 또한 고가이다. 이에 대한 대안으로 기존의 JTAG 핀에 몇 개의 Trace관련 핀만을 추가하여 CPU에뮬레이터와 같은 기능을 수행할 수 있도록 디버깅 포트를 구성하게 되었는데 바로 이것이 ETM(embedded trace macrocell)이다. 즉, Instruction과 data를 tracing할 수 있는 realtime trace module이다.

6.1.2 ETM 구조

a. trace port : processor의 operation를 이해하도록 도와 주는 signals을 보내줌
b. triggering and filtering facilities : data address, comparator, counter, sequencer를 이용한 triggering/filtering을 위한 모듈이다.

TPA(Trace Port Analyzer)을 이용하여 Trace information을 압축하여 trace Port를 통해서 export한다. 디버거가 TPA를 통해서 얻어진 압축된 정보를 풀어서 (decompression)disassembled된 코드로 만들어 준다.(심벌 포함)



6.1.3 Operation Mode

ETM control register를 이용하여 PORTMODE, output bus를 제어한다.
그리고, Trace debug tools을 이용하여 ETM핀(PIPEST,TRACEPKT,TRACESYNC)로부터 trace port signals을 ASIC의 trace port pins과 mapping하는 방법을 구성한다.

a. normal trace port signal
   - ETM의 revision 0에 의해서 유일하게 지원되는 모드
   - Normal/half clocking지원

- 100 Mhz이상의 high speed design에서는 clock의 signal integrity를 유지하기 위해 half rate clocking지원

b. multiplexed trace port signals
   - 50Mhz의 low frequency design에 적합하다.
   - TRACECLK의 양 edge에서 TPA가 trace information을 샘플링
   - Half rate clocking은 지원하지 않는다.

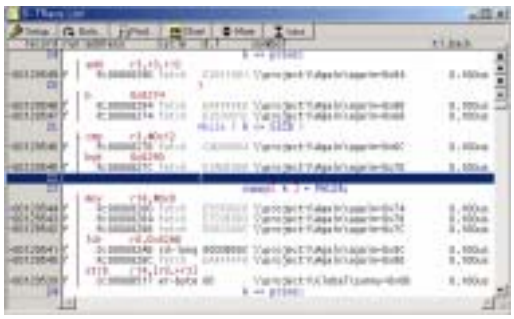
c. Demultiplexed trace port signals
   - high speed system이거나 trace를 위해서 일반적인 ASIC핀들을 재 사용하는 시스템이다.
   - Half rate clocking을 지원한다.
   - 256/512 Kframes trace depth
   - PowerTrace module사용시 64Mframe
   - 200Mhz speed
   -

6.2 Trace의 기능

- Timestamp 기능
- Context Tracking System
- Performance 분석
- Coverage 분석
- 통계 분석 기능
- Graphical trace views
- RTOS statistics
- Full ETM trigger support
- Target voltage 1.8-5v
- ARM7/ARM9를 포함하는 SOC디자인을 분석하기 위해 ETM 매크로 셀이 트레이스 신호를 만들어 낸다.

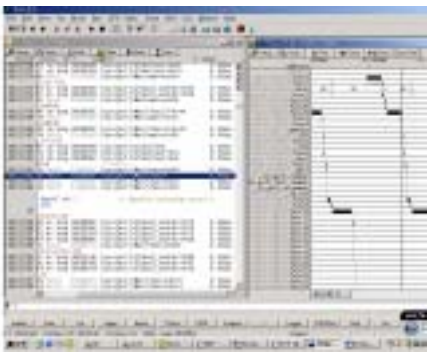
6.2.1 analyzing 기능

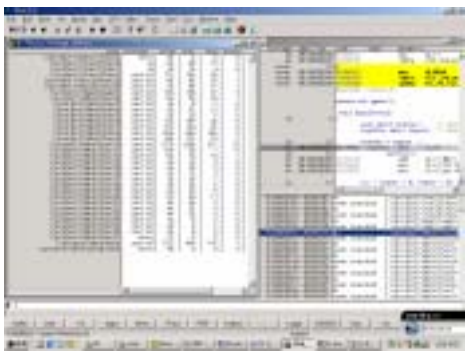
- ETM trace adapter가 최대 200Mhz clock speed로 샘플링가능
- 대상 영역 설정 트레이스
- 다양한 형태로 트레이스 정보를 볼수 있다
- 타겟을 멈추지 않고 트레이스 내용 볼 수 있다
- HLL/Assembler 코드 디스플레이한다
- 지연없이 곧 바로 트레이스 내용을 빠르게 디스플레이한다.



6.2.2 Code 분석 기능/프로그램 흐름 분석 기능

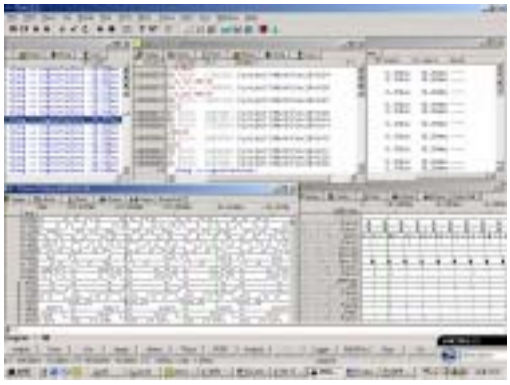
프로그램이 특정한 부분까지 수행했을 경우 어떤 순서에 의해 프로그램이 흘러 갔는지를 자동적으로 보여준다. 프로그램이 특정 부분까지 수행했을 때 어떤 순서로 프로그램이 흘러갔는지를 자동적으로 분석해서 보여준다. 게다가 실제로 수행된 코드와 수행되지 않는 코드가 어느 곳인지 보여줌으로써 code density를 높이는 계기를 마련할 수 있다.





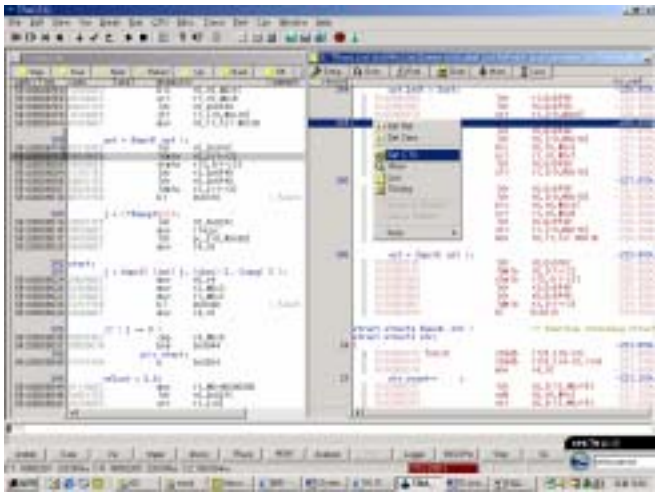
6.2.3 Timestamp

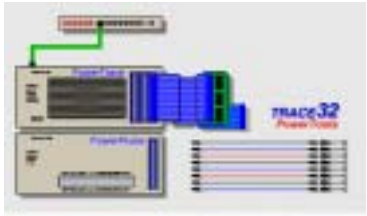
- instruction, address range, function 별로 수행하는데 얼마만큼의 시간이 걸렸는지 실시간으로 측정하여 디스플레이해 준다.
- Timing chart를 통해서 instruction과 tracking하여 보여 준다.



6.2.4 Context Tracking System & Smart Trace

- 리얼타임 프로그램의 싱글 스텝
- 로컬/글로벌 변수 디스플레이
- 스택 프레임 디스플레이
- 레지스터 값 디스플레이
- 함수의 nesting구조 디스플레이
- step,stepover,stepback 가능
- smart trace 기법을 이용하여 missing 될 수 있는 데이터 복원한다.





7. FAQ(Frequently Asked Questions)

7.1 Trace32의 PowerView 수행시 발생한 문제	
Q1	설치 후 ESI::로 보일 경우
A1	실행시 T32MARM.exe가 아닌 T32PBI.exe를 실행한 경우. 가끔씩 T32MARM.exe를 수행했는데도 불구하고 ESI::가 보임. 이 경우 다시 설치 또는 t32.cmm 파일에 B::을 입력해서 사용
Q2	바탕화면을 클릭했을 때 PowerView 부팅이 안되는 경우
A2	config.t32경로 또는 타입설정의 문제 ex) t32marm.exe –c d:₩t32₩config.t32 이런 식으로 등록정보에 입력해준다.
Q3	Parallel사용시 문제(EPP,ECP,Stadard)
A3	Config.t32 file에서 EPP선택하여 사용하도록 설정했다면, 컴퓨터의 BIOS setup도 똑 같은 모드로 설정해야 함. 참고로, Standard mode의 경우는 BIOS setup에서는 bidirectional로 설정해야 함 (예) PBI= LPT1 EPP ;참고로 standard일 경우는 이 라인을 생략한다.
Q4	“ICD PLD access failed, check interface and cables” 메시지
A4	Trace32의 PowerView를 두번 이상 실행했을 경우. 그렇지 않다면, ICD 하드웨어의 EPLD 문제일 수도 있슴
Q5	“PODBUS device scan failed. Check parallel port BIOS settings or try different mode.”라는 에레 메시지
A5	대부분 parallel interface module 내부에 있는 Altera chip이 손상이 된 경우..
7.2. “SYStem.Up”시 발생한 문제	
Q1	SYStem.Up시 debug port failed라는 메시지
A1	JTAG cable이 damage를 입었거나, 타겟쪽의 JTAG port에 문제가 생긴경우 자세한 체크 내용은 본 매뉴얼 뒷편의 “JTAG사용시 주의사항”편 참조
Q2	System.Up 수행시 "target processor in reset" 메세지

A2	System option중에서 EnReset을 disable하거나 target쪽 reset을 끊어준다. 이미 잘 사용하다가 갑자기 이런 문제가 발생한 경우에는 JTAG debugger cable문제인 경우도 있다. 또는 target 전압 level이 낮은 경우, 잘 사용하다가 안 되는 경우, 좀 오래 쉬었다가 해보거나 잘 돌아가는 target의 소스를 저장하여 다시 올려본다.
Q3	**SYStem.Up 수행시 "emulation debug port time-out at C:0x0" 메시지**
A3	Clock check(특히 메모리 동기 clock check(MSM chipset의 경우는 TCXO clock check)). 때대로 TDO,TRST부분에 문제일 경우나 jtag debugger문제일 수도 있슴
Q4	**Sys.u은 되는데 각 윈도우상태는 reset인 상태인 경우.**
A4	Jtag debugger의 문제
Q5	**sys.u은 되지만 target reset이 되어져 있고 화면에는 아무것도 보이지 않고 jtag을 바꾸면 또 잘되고..**
A5	Jtag의 문제인데 정확한 원인규명이 어렵다... 본사로 수리를..-_-;
Q6	**Sys.u 하자마자 target이 running이 되는 경우**
A6	Clock 속도(jtag clock & CPU clock)를 조절해 본다.
Q7	**SYStem.Up은 되지 않고 SYStem.mode Go를 하면 정상 동작하는 경우**
A7	대부분 watchdog 관련 문제일 수 있다. SYStem.Up시 ICD는 타겟을 한번 리셋시키고, PC(program counter)을 reset vector에 두고 CPU를 ilde상태로 들어가게 하는데, 이때 watchdog문제로 debug port failed에러가 발생할 수 있다. 이에 반해 SYStem.mode Go는 jtag port test, 타겟 리셋이후 바로 running하기 때문에 문제가 해결되는 경우가 있다.
Q8	**SYStem.Up은 되나 d.I창을 띄우면 reset이 됨**
A8	d.I창을 띄울 때 현재의 메모리값을 읽어오는데 메모리초기화가 되어 있지 않은 영역을 열거나 메모리 동기화 클록인 TCXO가 불안정할 경우..

7.3. Flash Programming시에 발생하는 문제

Q1	**Flash.erase관련 명령 수행시 flash erase error at 0x0라는 메시지를 얻을 때**

A1	여러가지 원인이 있을 수 있다. 타겟 RAM의 메모리 초기화 되지 않아 전혀 프로그램이 동작하지 않았을 수도 있으며,flash memory의 write protect핀과 Vpp pin이 정상적으로 인가되어 있지 않아서 flash erase가 되지 않을 수 있다. 이 경우, 메모리 초기화 설정이 제대로 되어 있는지 체크해 보아야 하고, write protect나 Vpp는 정상적으로 설정되어 있는지 살펴 보아야 한다. 특히 GPIO를 이용하여 Write protect나 Vpp를 control하도록 되어 있다는 적절한 GPIO값이 설정되어 있는지 체크해 보아야 한다. 때때로 watchdog으로 인해서 flash erase에 문제가 생기는 경우도 있다. 이 경우 watchdog를 주기적으로 flush하는 루틴을 erase부분에 삽입해야 한다. (참고) 오래전에 작성한 flash 알고리즘을 가지고 최근의 Trace32 소프트웨어를 구동시킬 때는 아래의 경우를 고려 해야 한다. Target based flash programming에서 erase routine수행하고자 할 때, Flash.erase [block 지정]을 하게 되면 parameter의 8번째값을 2 또는 5로 설정하게 된다. 예를 들면, flash.erase all하게 되면 *(parameter+7)이 5로 설정되게 되어 있다. 그런데, flash.erase [block 지정]은 그 parameter위치에 2 또는 5값을 설정한다. 여기서 block의 range가 flash.create [block 지정] 명령의 block 범위와 같다면 그 parameter의 값은 flash.erase all과 마찬가지고 5로 설정이 되며, flash.erase [block 지정]의 block범위가 flash.create [block 지정]보다 작을 경우에는 2로 설정되게 되어 있다. 그러나, 99년 이전 소프트웨어에서는 일단 block를 지정하게 되면 무조건 그 parameter값이 2로 설정되게 되어 있다. 그러므로, 예전에 만들어진 소스의 경우 erase routine에서 parameter를 비교할 때 2와 비교하도록 되어 있다. 그러므로, flash.erase 뒤에 full range를 주었을 경우는 이 routine을 수행할 수 없으므로 이 부분을 5와 비교하도록 바꾸던가, 아니면, 임시방편으로 flash.create의 block range을 erase block range보다 크게 설정하면 된다.

Q2	**“Bus error generated by CPU”라는 에러 메시지가 발생할 경우**
A2	소스에서 포인터등을 잘못 선언해서 정의 되어 있지 않은 메모리 영역에 접근하였을 경우 bus error가 발생한다. 때때로 초기화한 메모리 영역이 잘못 설정되어 있는 경우도 있다. 참고로, 오래전에 구입한 usb version일 경우는 앞의 원인과 관계없이 이와 같은 에러가 발생한 경우가 있으므로 이 경우는 usb firmware를 upgrade시켜야한다. USB version이 1.3이상에서는 거의 이런 문제가 일어나지 않는다. 드문경우기는 하지만, MSM chipset의 경우 TMODE가 high로 되어 있을 경우 이런 메시지가 뜨기도 한다.
Q3	**Flash memory에 programming writing할때 step으로 할 때는 잘 되지만, flash algoritm을 running시켰을 경우는 부분적으로 write되지 않았을 경우.** **예를 들면, buffer size만큼 writing할때 첫번째 1word만 writing이 안될 경우.(반복적으로..예를 들어 1000byte 단위로 writing할때 0,1000,2000...번지의 값이 제대로 써지지 않은경우)**
A3	정확한 원인은 알 수 없으나 Flash algorithm에서 programming routine중 첫번째 word쓰는 부분앞에 의미없는 delay routine을 추가하면 된다.
Q4	**Target based flash programming을 할 때, parallel version은 되는데 USB버전은 잘 되지 않을 경우**
A4	Turbo option을 off하고 buffer 사이즈를 0x1000이하로 설정해야 한다.
7.4. Multiprocessor/Multicore 설정 방법	
Q1	**Multi-Processor setting방법**

A1	먼저 config.t32파일을 processor개수만큼 생성해주고, 그것에 따른 바탕화면에 바로가기 아이콘을 설정한다.그 다음 아래와 같이 config.t32파일을 생성해 준다. ********** PBI= USB USE=1100 ;첫번째, 이 경우 usb를 연결한 부분 ;USE=0011 ;두번째 IC=NETASSIST PORT=20000 ;PORT=20001 ;두번째 PACKLEN=1024 ; Printer settings PRINTER=WINDOWS ; Screen fonts SCREEN= HEADER=ARM7 FONT=SMALL ********** 모든 설정이 끝난후 trace32가 실행되면, command line에서 synch라고 명령을 주면 창이 하나 뜨는데 여기서 synch에 on을 해주고 connect부분에 localhost:20001등 설정을 해줘야 한다. 이 설정은 두개의 processor창에 모두 해줘야 하고 localhost의 경우 참조하는 config에 설정된 것을 상대방의 trace32창에 설정해준다. 예) arm7 config파일에 localhost가 20000으로 설정되어 있으면 arm9이 연결된 trace32를 실행한 창의 connect부분에 localhost:20000이라고 셋팅한다.
Q2	**Multicore 설정 방법**
A2	Debugging하는 ARM core 앞 뒤로 어떠한 core들이 daisy chain으로 연결되 있는지를 Trace32에 알려주어야 하는데 이때 Instruction register사이즈와 Data register사이즈가 필요하다. 예를 들면 TDI---A core(IR' size 6 bit)---ARM core---B core(IR's size 7 bit) IRPOST 0x1

	DRPOST 0x6 IRPRE 0x1 DRPRE 0x7 SYStem.up
Q3	**Daisy chain의 구조를 간접적으로 판단할 수 있는 방법**
A3	Trace32 ICD for ARM의 경우는 JTAG port를 테스트할 수 있는 진단 프로그램이 있는데 이 진단 프로그램을 이용하면 multicore system의 daisy chain구조를 알아 내는데 유용한 정보를 얻을 수 있다. AREA ; 이 메시지 윈도우 IR(Instruction Register), DR(Data Register)에 관한 정보가 디스플레이된다. Diag 3400 ; 진단 프로그램 구동 SYStem.Up

7.5. Miscellaneous problems

Q1	**source list, register등 값이 flickering하는 경우**
A1	메모리 초기화가 정확히 설정되어 있다면, jtag clock을 낮춰보고 turbo option을 disable시켜서 동작시켜 본다. Parallel version의 경우는 모드를 standard로 한다. (not ecp, epp)
Q2	**어떤 라인에서 step 또는 go명령을 주어도 더 이상 진행되지 않고 멈춰 있는 것처럼 보이거나 timer에 관련된 인터럽트가 계속 뜰 경우**
A2	Interrupt를 masking한다. Setup.imaskasm on Setup.imaskhll on
Q3	**Ethernet사용시 Symbol 메모리 부족문제**

A3	Ethernet module의 경우는 심벌정보 다운로드시 default로 Ethernet을 사용하도록 되어 있는데 이 경우는 심벌 사이즈가 클 경우는 문제가 될 수 있다. 이 경우 parallel/usb version처럼 Ethernet module대신에 host computer의 메모리를 사용하도록 설정할 수 있다. 이를 위해서는 아래와 같이 config.t32 파일을 수정하여야 한다. PBI= NET NODE=211.41.28.60 LINK=NET NODE=211.41.28.60 PACKLEN=1024 그리고, t32marm.exe파일을 수행시킨다.
Q4	**Elf 파일 다운로드시 "Error: entry near offset 2544998. in file c:data.....(use DUMP)" 에러메시지가 발생하는 경우**
A4	Elf format이 아닌 파일을 data.load.elf명령을 이용해서 다운로드할 경우 위와 같은 문제가 생길 수 있다.
Q5	**"code at software breakpoint has changed at address"라는 메시지를 얻을 경우**
A5	기존에 잡혀있는 software breakpoint를 사용자 프로그램에서 overwrite했을 경우나 mmu initialization에 의해 memory가 remapping되었을 경우. 가끔은 메모리 초기화 문제가 연관되는 경우도 있다.
Q6	**Target의 메모리를 읽고 나서 값이 처음에는 이상이 없다가 몇 번의 동작 후에 값이 흔들리며 run이 되지 않는 경우**
A6	Clock과 reset쪽에 문제가 있을 수 있다. 아니면, stack pointer값이 정의된 메모리영역이외의 곳으로 변했는지 체크해야 함.
Q7	**20pin JTAG connector에서는 정상적으로 동작하고 14pin에서는 동작되지 않는 경우**
A7	20 pin connector와 14 pin connector는 허용 voltage범위가 틀리다. 20pin은 동작 vcc가 1.8~3.3V 이고 14pin은 2.5~5.0V이므로 이 부분을 확인해 보아야 한다.
Q8	**ETHERNET not enabled, demo for 5 min이라는 메시지가 뜰 경우**
A8	Ethernet용 License key를 받아야 한다. Ethernet용 license에는 PC-LAN용과 Workstation용이 있다.

7.6. 기타 유용한 tooltips

Q1	**Memory의 내용을 binary 파일로 저장하고 싶을 경우**
A1	예를 들면 0x0번지부터 1Mbyte size의 code를 a.bin이라는 이름으로 저장하고 싶을 때 다음과 같은 명령을 수행하면 된다. Data.save.binary a.bin 0x0—0xfffff
Q2	**각각의 Window창의 내용을 텍스트 파일 형태로 저장하고 싶을 경우**
A2	예를 들면 0x0번지부터 1Mbyte size의 내용을 소스와 디스어셈블 코드를 포함하는 ascii format의 텍스트 파일로 저장하고자 할 때 다음과 같이 수행된다. Printer.filetype ascii printer.file source.txt wp.data.list 0x0—0xfffff ; 저장하고자 하는 윈도우 이름 앞에 “wp”라는 접두어를 붙힌다.
Q3	**Elf format의 file를 다운로드했는데도 불구하고 C source는 보이지 않고 그 자리에 빗살무늬 표시가 보일 때(symbol path가 달라서 C source를 볼 수 없을 때)**
A3	보통 –g option을 주고 compile하면 symbol information이 들어가게 된다. 이 경우 symbol의 relative path 또한 저장되게 된다. 그런데, 소스 위치가 바뀌었거나 소스가 현재의 컴퓨터상에 없는 경우 위처럼 빗살 무늬 표시만 화면에 보이게 된다. 이 경우 소스를 보기 위해서는 반드시 현재의 컴퓨터상에 elf format file뿐만 아니라 소스파일도 존재해야 한다. 뿐만 아니라 소스 위치도 명확히 지정되어야 한다. 물론, 컴파일할 때의 소스위치가 변경되지 않았을 경우는 상관없지만 만약 소스위치를 임의의 위치로 바꾸었다면 다음과 같이 소스 위치를 지정하는 명령을 수행하여야 한다. y.spath + c:₩the location of the sources
Q4	**다중 elf format file을 다운로드시 주의할 점**
A4	Trace32은 files의 multi-loading이 지원된다. 단 multi-loading시 제일 나중에 다운로드되는 파일의 심벌 정보만이 저장되므로 기존에 먼저 다운로드한 파일에 대해서는 소스정보를 볼 수 없다. 만약 초기에 다운로드된 파일을 소스 레벨로 디버깅하기 위해서는 그 다음 다운로드 명령에서 noclear option을 주면 된다. Data.load.elf a.elf Data.load.elf b.elf /noclear Data.load.elf c.elf /noclear 위처럼 설정하면 a.elf b.elf c.elf 파일 모드를 소스 레벨로 디버깅할 수 있다.

JTAG Cable 사용시 준수 사항

당사의 제품(TRACE32-ICD)을 구입해 주셔서 감사 합니다.
JTAG CABLE을 고장없이 잘 사용하실려면 아래의 사항을 준수 하여 주십시오.

주의 사항

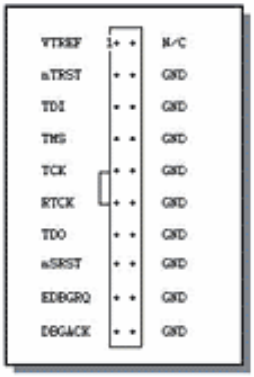
1. Target Board에서 JTAG 핀과 연결할수 있는 전용 소켓을 만들어 주십시오. 그리고 방향을 표시하셔서 절대 다른 방향으로 꽂는 일이 없도록 해주십시오.(JTAG Cable의 Buffer가 손상이 갈수 있는 주된 원인이 됩니다.)

2. 전원의 ON-OFF 순서를 지켜 주십시오. 순간전압(Over Short)에 의해서 JTAG CABLE에 손상이 갈 수 있습니다. Target 만 동작시킬때는 완전히 장비와 분리 시켜 주십시오.
(전원 ON시 : ICD Power ON -> Target Power ON
전원 OFF시 : Target Power OFF -> ICD Power OFF)

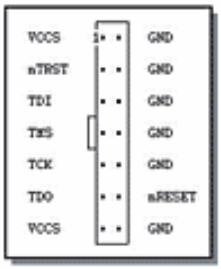
3. JTAG CABLE은 Target의 Vref를 받아서 TDI,TMS,TCK,TRST등의 JTAG 시그널을 Target전압에 맞게 변환해 줍니다. 그렇기 때문에 JTAG CABLE의 VTREF(=VCCS) 핀의 경우는 Target CPU (JTAG 이 들어 있는 Core)의 Vref핀을 사용하셔야 되고 GND핀 역시 CPU의 GND핀(=Digital GND)과 공용해야 합니다.

4. JTAG CABLE의 VTREF(=VCCS) 핀의 전압 레벨은 1.8V-3.3V가 되어야 됩니다. 즉, Target Vref의 1.8V-3.3V를 JTAG CABLE의 VTREF(=VCCS)에 연결하셔야 합니다. (20핀의 경우는 JTAG CABLE의 VTREF(=VCCS)핀에 5V가 공급될경우 Support가 되지 않습니다. 만약에 5V를 사용해야 할 경우는 저희에게 연락 주세요.)

5. nSRST (=nRESET)핀은 target CPU를 디버거가 Reset 할때와 Target CPU 상태가 reset 인지를 확인할때 사용됩니다. Reset을 사용하실려면 CPU의 Reset핀에 연결하시고, 기본적으로 nSRST (=nRESET)핀은 pull up(47k) 되어 있으므로 Open 하셔도 상관없습니다.



20 pin JTAG Cable 핀



14 pin JTAG Cable 핀

< Top View > 소켓의 핀 번호에 유의 해주세요.

HANKOOK MDS
Embedded Communications Company

Self-check list for TRACE32-ICD

ICD를 사용 중에 Host상에서 Error Message가 보인다면 아래의 사항을 확인 바랍니다.

Error Message

- emulation debug port failed
- Target processor in reset
- emulation debug time-out at 0xXXXXX

Debug Module Side Check point

- Pin 사양이 규격에 맞게 연결 되었는가 ?
  (장비의 JTAG Connector <-> Target Connector)
- EnReset, ResBreak, TRST option을 각각 disable 또는 enable한 후에 테스트해도 똑 같은 결과를 얻는가?
- TCK, TDI, TDO, TRST Signals은 Noise가 많이 혼합되어 Generate되고 있는가?
- CPU에 따라 기본 핀(TDI,TDO,TMS,TRST,TCK) 외에 JTAG enable과 관련되는 하드웨어 핀이나 Register setting은 부가적으로 필요하지 않는가?
- 복수개의 Core가 Daisy chain으로 연결된 ASIC chip이라면, Multi-core와 관련된 설정(IRPOST, IRPRE, DRPOST, DRPRE)은 정확히 하였는가?
- 혹시 다른 JTAG Cable을 가지고 있다면, 교체해서 실행해 보았는가?
  (만약 바뀐 JTAG Cable 로 똑 같은 문제가 있다면, 타겟쪽의 JTAG 을 의심해 볼 필요가 있습니다.)

Target Side Check point

- JTAG Connection은 14핀/20핀 규격에 맞게 정확히 연결되어 있는가?
- Target에 Vcc는 정상적으로 인가되는가?
- Ground 핀들은 모두 연결되었는가?
- JTAG enable와 관련된 핀이나 Register setting은 필요한가?
- MSMxxx Chip경우 PS holders는 high로 인가되는가?
- Jumper로 연결한 Jtag line들이 너무 길지는 않은가?
- Target의 Reset 회로쪽은 문제가 없는가?
- CPU의 TCXO signal(메모리 동기와 관련)은 정상적으로 generate되고 있는가?

상기 부분의 Check를 해도 이상이 없으시다면 저희 고객지원 파트로 문의 바랍니다

e-mail : arm@hkmds.com

HANKOOK MDS
Embedded Communications Company