ARM Developer Suite Overview

Overview of ADS 1.2

- Integrated set of software development tools for embedded ARM development
  - from evaluating initial prototype software through to producing final optimized ROM code
- Released December 2001
- Supported hosts
  - IBM compatible PCs with Windows 95, 98, 2000, ME or NT4
  - Sun workstations with Solaris 2.6, 2.7 or 2.8
  - HP workstations with HPUX 10.20, 11
  - Red Hat Linux 6.2 & 7.1
- Full version License managed using FlexLM
- Evaluation version available
- Third-party tool-chains are also available:
  http://www.arm.com/DevSupp/

New in ADS 1.2

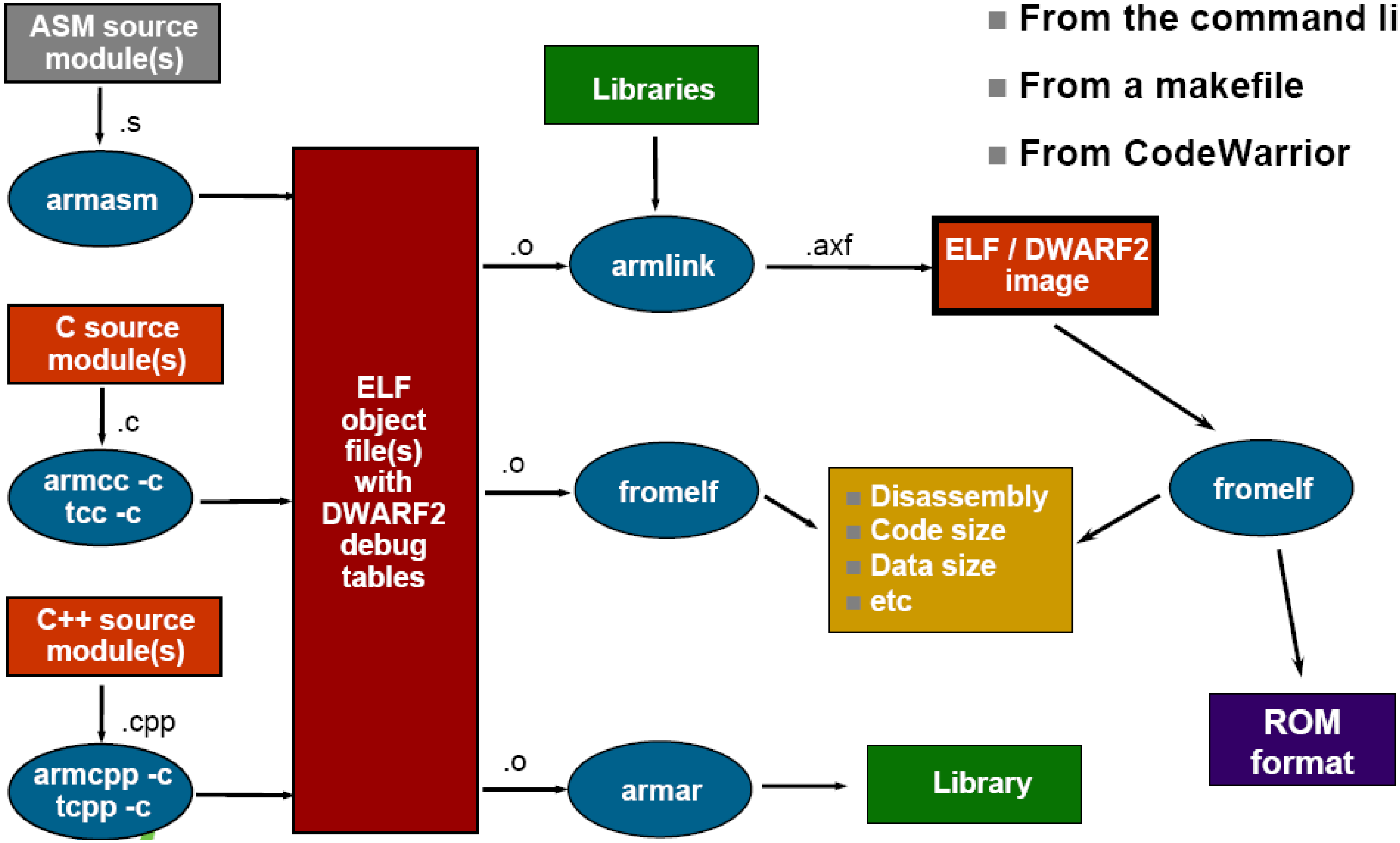
- Support for latest ARM cores
  ARM926EJ-S, ARM9EJ-S, VFPv2
- Support for Architecture V5TEJ
- ARMulator simulates execution of Java bytecode
- Bytecode display in Jazelle state
- Hosted on Red Hat Linux (6.2 and 7.1)
- Librarian can merge libraries
- Finer grained placement of code and data using pragmas
- New linker option for relocatable code
- Parallel assembler and object code output from compilers

Main components

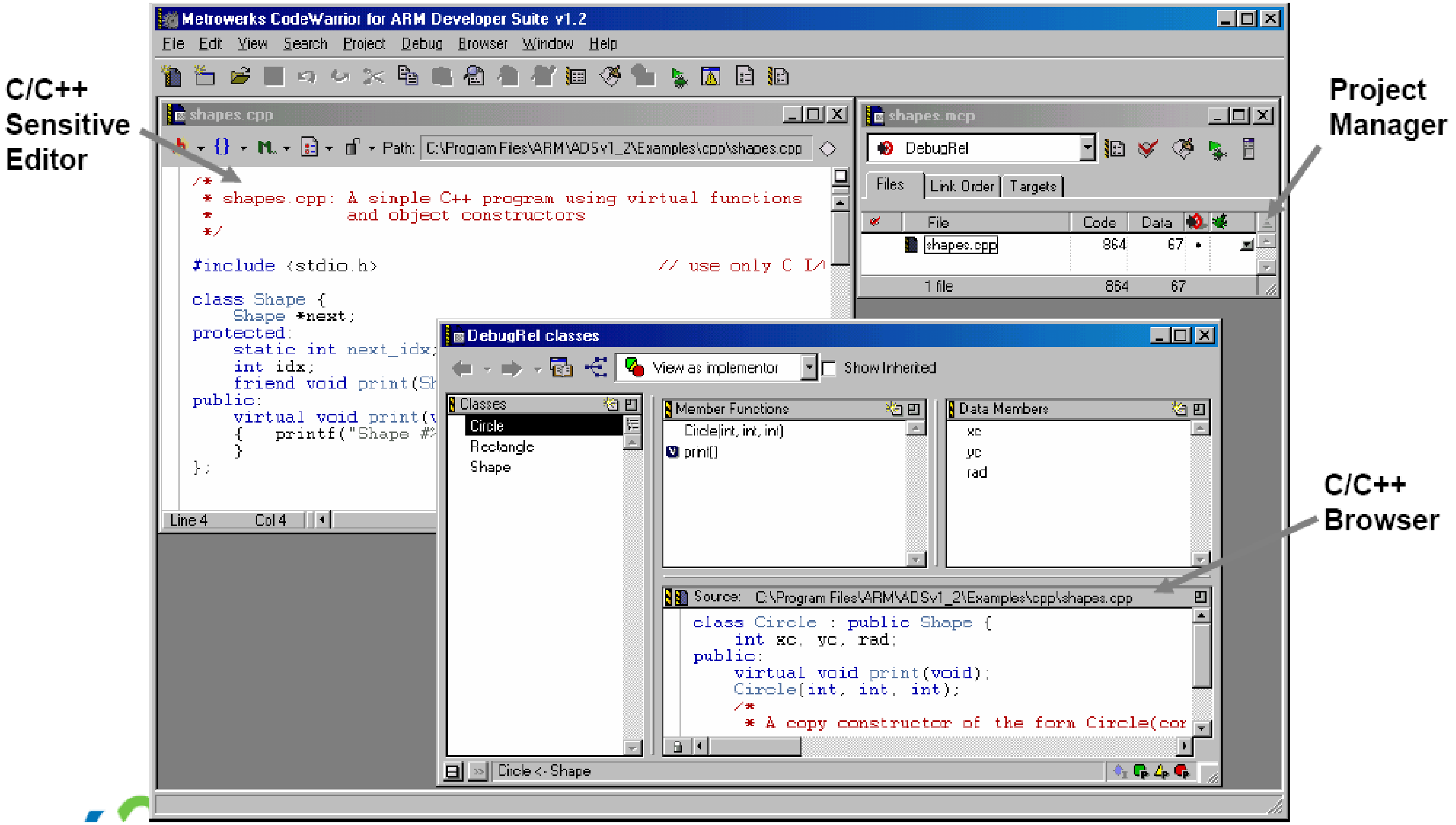
- ANSI C compilers – armcc and tcc
- ISO / Embedded C++ compilers – armcpp and tcpp
- ARM / Thumb assembler – armasm
- Linker – armlink
- Windows IDE – CodeWarrior
- Debugger – AXD
  - armsd also supplied for backwards compatibility
- Format convertor – fromelf
- Librarian – armar
- C and C++ libraries
- Instruction set simulator – ARMulator
- Also ships with
  - ARM Firmware Suite
  - ARM Application Library
  - RealMonitor

Using the Core Tools

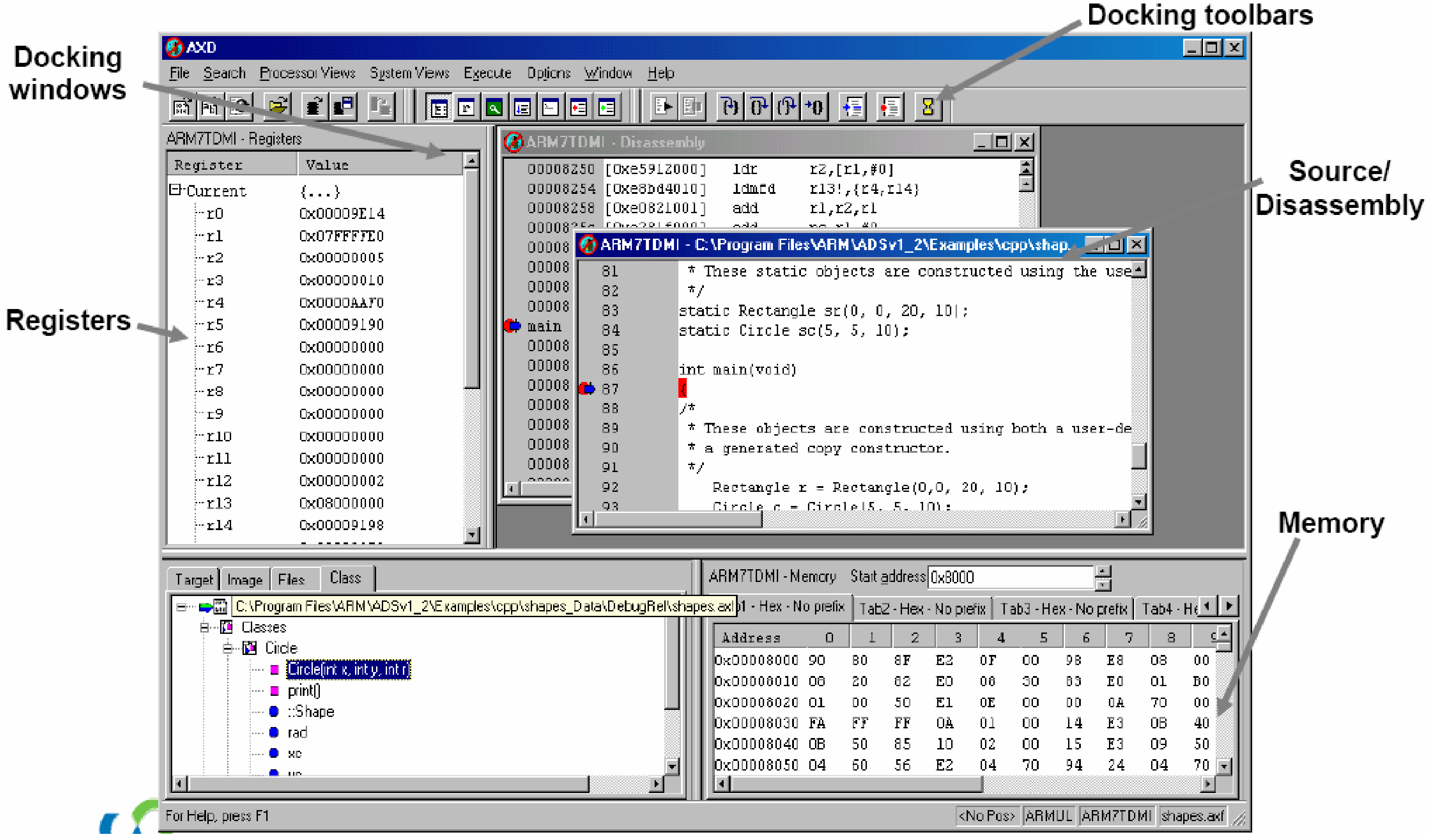
- From the command line
- From a makefile
- From CodeWarrior



CodeWarrior



AXD



Docking toolbars
Docking windows
Registers
Source/ Disassembly
Memory
AXD
File Search Processor Views System Views Execute Options Window Help
ARM7TDMI - Registers
ARM7TDMI - Disassembly
ARM7TDMI - Memory
For Help, press F1

C / C++ Compilers – Key Features

- Fully ANSI compliant C compilers
- ISO / Embedded C++ compilers
- Support for all ARM processors (with '-cpu')
  e.g. ARM7TDMI, StrongARM, ARM9TDMI, ARM9E, ARM10, Xscale
- Allows source-level debugging of optimized code:
  -O0 : best debug view, no optimization (default with -g) "Debug"
  -O1 : most optimizations, good debug view with -g "DebugRel"
  -O2 : full optimization (the default), limited debug view "Release"
- Support for ROPI / RWPI
- Inline assembler
- Interleaved C and Assembler listing (with '-S -fs')

C / C++ Compiler – Data Types

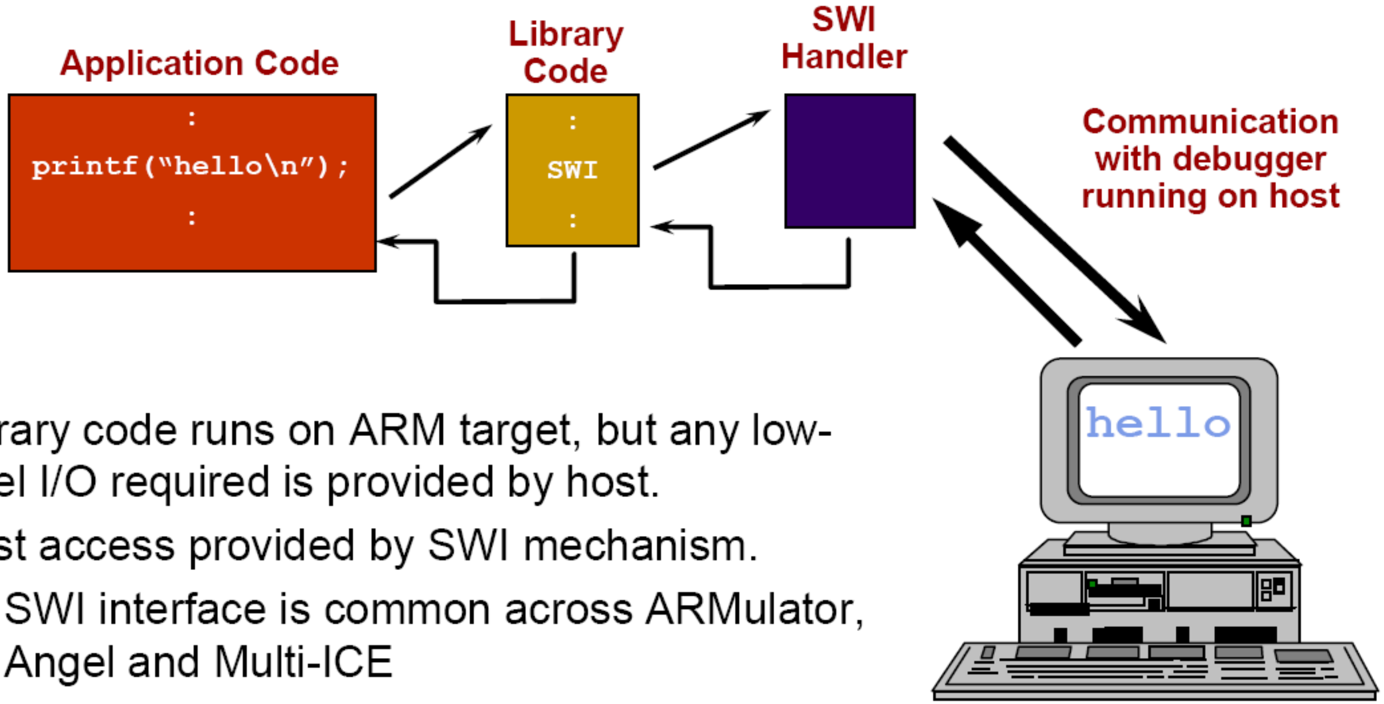
- These data types are supported:
  - char 8 bit byte
  - short 16 bit half-word
  - int 32 bit word
  - long 32 bit integer
  - float 32 bit IEEE single-precision
  - double 64 bit IEEE double-precision
  - pointers 32 bits
  - long long 64 bit integer

All above are signed by default, except pointer, and char (which is unsigned by default and can be made signed with -zc)

Supplied Libraries

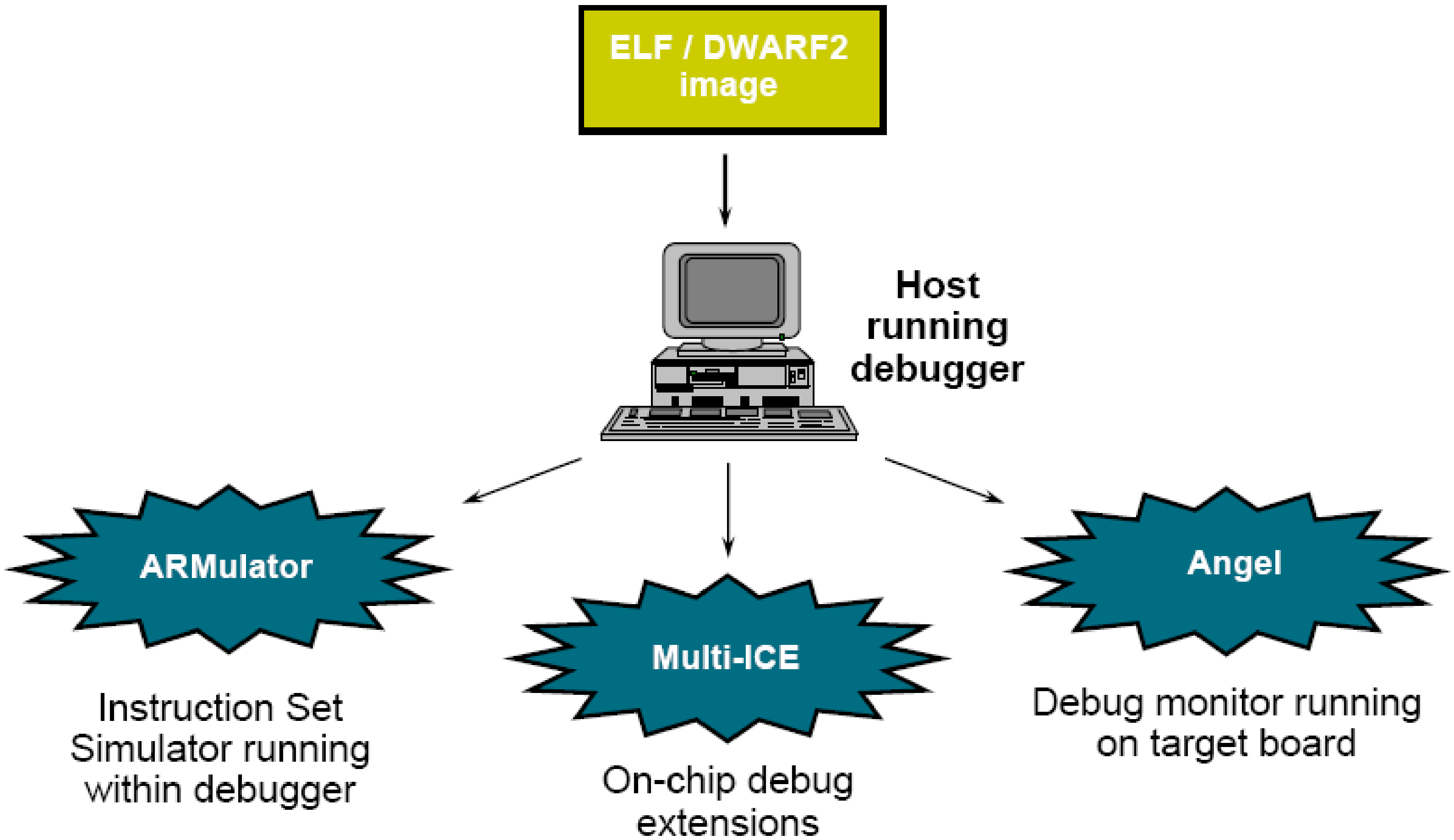
- ANSI C library
  - Full file handling, string, maths, etc., support on the target
  - By default uses semihosted SWI's to access host debugger facilities
    - e.g. file I/O
  - Retargetable without need for rebuild kit
    - Suitable for embedded use – no separate embedded variant
  - Automatic selection of the correct library variant
    - depending on byte order, position independence, stack checking, etc
  - Also contains run time support functions and floating point support
- C++ library includes:
  - Rogue Wave Standard C++ library version 2.01
  - Run time support functions for the C++ compiler.

Semihosting



- Library code runs on ARM target, but any low-level I/O required is provided by host.
- Host access provided by SWI mechanism.
  - SWI interface is common across ARMulator, Angel and Multi-ICE
  - Semihosted programs will run on all ARM targets without need for porting
- Debug tools must be connected to provide this functionality

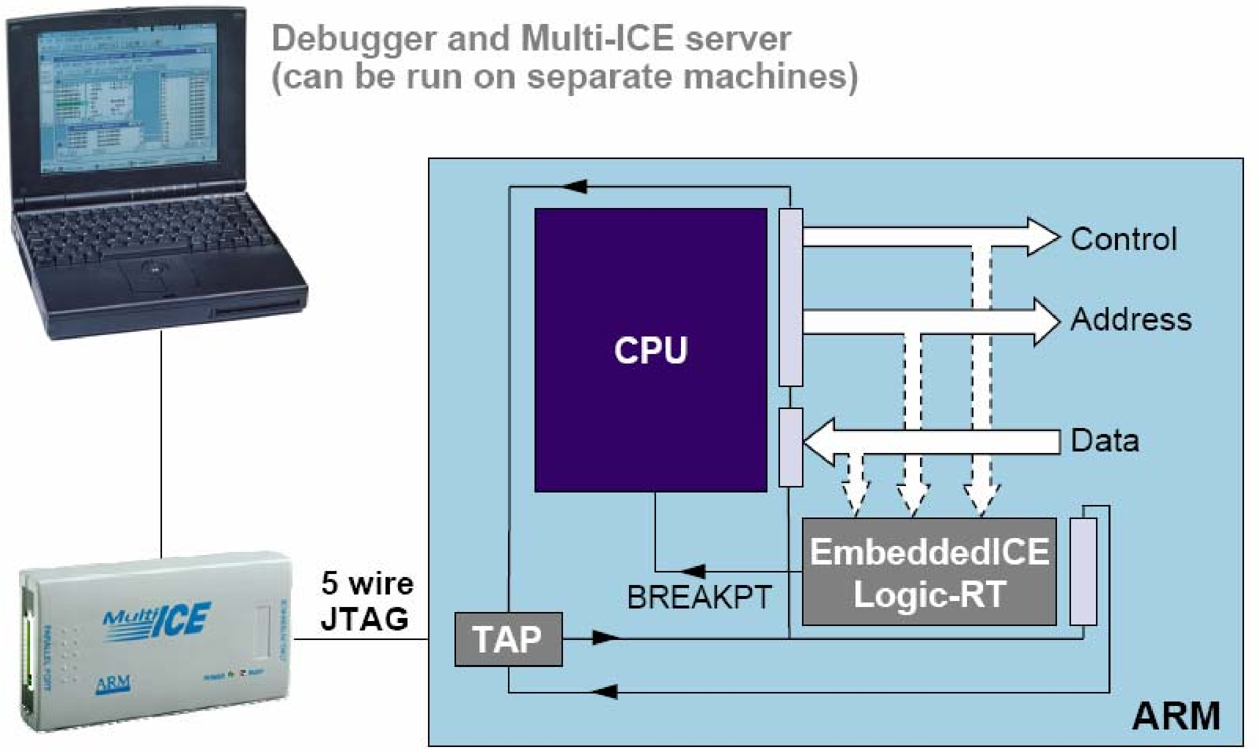
Supported Targets



ARMulator

- Software simulation of an ARM core
- Instruction accurate
  - Allows program execution to be verified
  - Counts elapsed memory cycles
  - Allows benchmarking of systems when given memory map and clock speed
- Configurable for all ARM cores
  - latest cached cores, e.g ARM946E, ARM966E, ARM10 (incl VFP), XScale
- Tracing support
- Customizable – Extension kit supplied
  - Models are written in C, so easy to add new peripheral models with Visual C++ (Interrupt controller and Timers already supplied)
  - Can also simulate IRQ or FIQ interrupts ($irq, $fiq)
  - ADS 1.2 Debug Target Guide has details on writing new models

Multi-ICE



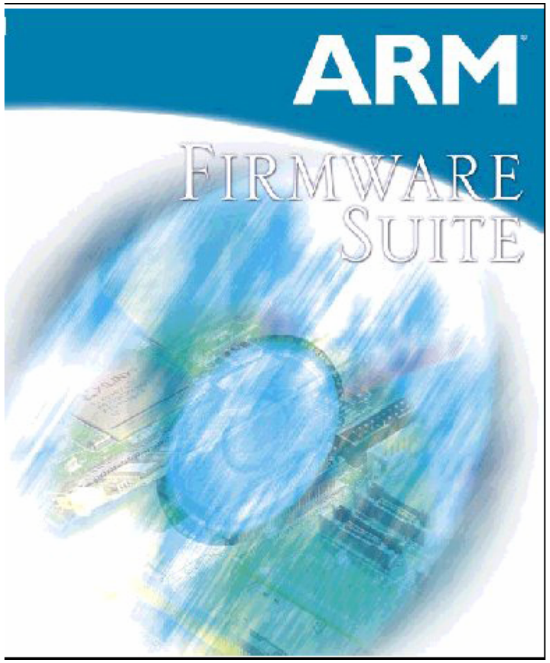
- The system being debugged may be the final system!
- Third party protocol converters are also available

Angel Debug Monitor

- Debug Monitor which runs on target hardware
  - Does not require ARM core to have an EmbeddedICE logic
  - Processor never physically halted – useful for real time applications
  - Application code must run in RAM (to set breakpoints, single-step, etc.)
- Communicates with host debugger via ADP (Angel Debug Protocol)
  - normally over serial link
- Sources supplied as part of the ARM Firmware Suite (AFS)
- Written mainly in C, to ease porting to new hardware
- Ports to third party boards available from other vendors

ARM Firmware Suite

- Library of low-level board software and utilities
  - uHAL (Hardware Abstraction Layer)
    - →Masks hardware differences between platforms from other firmware components and applications
    - →Designed to speed the development cycle by providing code for System Initialization, Memory Management and Interrupt Handling/Installation (including timer support).
  - Boot Monitor
  - Angel Debug Monitor
  - Flash Management Library
  - PCI Management (for Integrator /AP)
- Shipped with latest ARM boards and with ADS

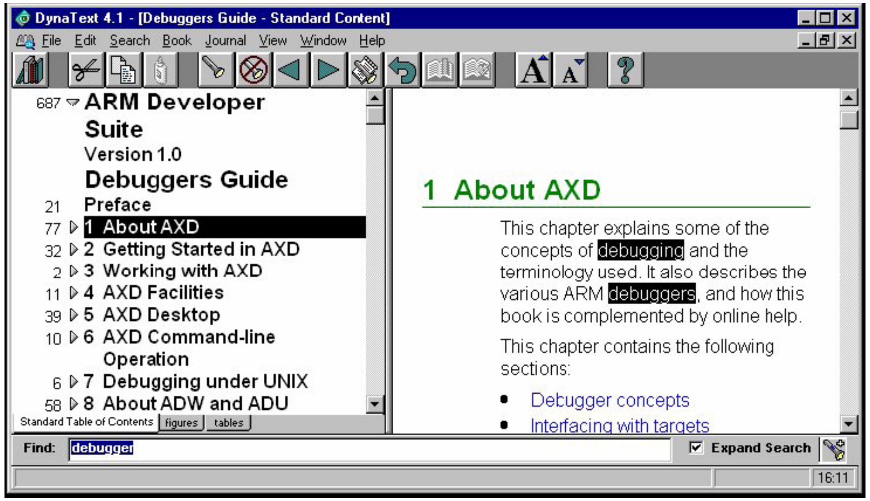


ARM Application Library

- A highly optimized software suite of useful routines, algorithms and applications
- Handcrafted by ARM architecture experts
  - excellent examples of optimized ARM code
- Full source code, build utilities and documentation included
- Royalty free inclusion in application software
- Routines include
  - DSP Transformations – DCT, FFT with Hamming & Hanning windowing
  - DSP Filtering – FIR, IIR & LMS
  - Mathematics –
    - Fast fixed point integer multiplication & divides
    - Square Root, Cube Root, Trigonometric functions
    - Signed saturated addition
  - Plus a large number of other useful functions

Documentation

- Installation and License Management
- Getting Started
- Assembler Guide
- Compiler, Linker and Utilities Guide
- Debug Target Guide
  - Angel, ARMulators, Semihosting
- Debuggers Guide
- CodeWarrior IDE Guide
- Developers Guide
  - Writing code for the ARM
- Available in
  - DynaText online book format
  - Printed Manuals
  - PDF



- On-line help from within Windows tools (just press F1)
- Application Notes downloadable from:
  - http://www.arm.com/Documentation/AppNotes
- Technical Support FAQ at:
  - http://www.arm.com/DevSupp/Sales+Support/faq.html

AXD Training

Command Line Tools

- armcc : ARM C Compiler
- tcc : Thumb C Compiler
- armlink : Object code linker
- armasm : Assembler for ARM/Thumb source code
- armsd : ARM command line debugger
- fromelf : File format conversion tool

Compiling and running the example

- Edit following a source code.

```
/* hello.c Example code */
#include <stdio.h>
#include <stdlib.h> /*for size_t*/
void subroutine(const char *message)
{
        printf(message);
}
        int main(void)
{
        const char *greeting = "Hello from subroutine₩n";
        printf("Hello World from main₩n");
        subroutine(greeting);
        printf("And Goodbye from main₩n₩n");
        return 0;
}
```

- CMD> armcc –g hello.c
- CMD> armsd –exec __image.axf

Compilation Options

- Options
  - -c : Generate object code only. Doesn't invoke the linker.
  - -o <filename> : Name the generated output file as 'filename'
  - -s : Generate an assembly language listing.
  - -s -fs : Generate assembly interleaved with source code
- Use the compiler options with armcc or tcc to generate following output file from hello.c
  - image.axf : An ARM executable image
  - source.s : An assembly source.
  - inter.s : A listing of assembly interleaved with source code
  - thumb.axf : A Thumb executable image.
  - thumb.s : A thumb assembly source.

armlink

- Edit following files, main.c sub.c :

```
// main.c Example code to be used with 'sub.c'
#include <stdio.h>
extern void subroutine(void); //found in 'sub.c'
int main(void)
{
printf("Hello World from main₩n");
subroutine();
printf("And Goodbye from main₩n");
return 0;
}
// sub.c Example code to be used with 'main.c'
#include <stdio.h>
void subroutine(void) // called by 'main.c'
{
printf("Hello from subroutine₩n");
}
```

- CMD> armlink main.o sub.o –o link.axf /* if not used ' –o ', then generated __image.axf */
- CMD> armsd –exec link.axf

fromelf

- CMD> fromelf –text/c hello.o
- CMD> fromelf –text/c hello.o > hello.txt /* save interleaved file into hello.txt */
- Open the ‘hello.txt’ with a editor

Codewarrior and AXD

- Creating header file
  - Click codewarrior icon in the Windows start menu.
  - Select *file->new* from menu.
  - Ensure the *file* tab is selected in the *New* dialog-box.
    - Select *Text File*
    - Click *OK*
  - Enter the following C structure definition

```
/* Struct definition */
struct DateType
{
        int day;
        int month;
        int year;
};
```

  - Select File Save As… from the menu.
  - Save as a 'datetype.h' in "c:₩work₩demo"

Creating new project

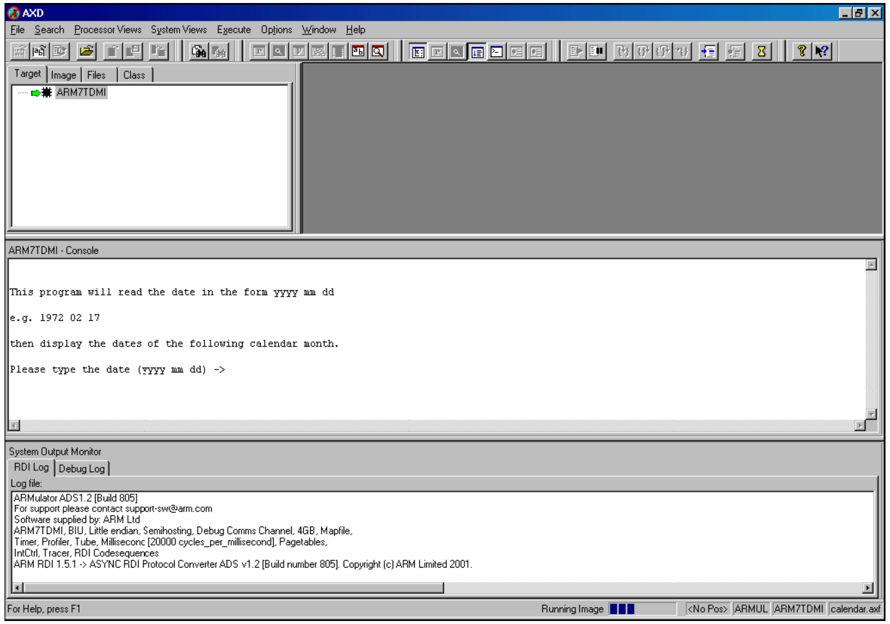
- Select *File->New* from menu.
- At *project* tab, select *ARM Executable Image* and enter the Project name : demo
- Enter the project directory : “c:₩work₩demo”
- Click *OK* to create the project
- At the project window, *demo.mcp*
  - *Debug* : Contain full debug information table and very limited optimization
  - *Release* : No source level debug information, but full optimization
  - *DebugRel* : A trade off between the two.
- From the menu, *select Project->Add Files…* to locate “month.c”, “datetype.h”

Building the project (DebugRel target)

- From the menu select *Project->Make* (or press *F7*)
- Double click on the first error message
  - The editor window is opened.
  - At the top of the file, preprocessor directives contain a reference to the macro, which has not been defined in any of the source file.
- From the menu of the project window, demo.mcp, select *Edit->DebugRel Settings*
- In the *Target Settings Panels* box, click *ARM Compiler*
- Select *Preprocessor* tab, In the field below *List of #DEFINEs* enter "DATETYPE"
- Click *Add,* then click *OK*
- Rebuild the project by pressing *F7*
- To show disassembled code, right-click on month.c in the project window and select *Disassemble*.

Executing the example

- From the menu, select *Project->Run*
- AXD window will be opened. In the console window, enter today's date, e.g. 2006 06 08
- Quit AXD by selecting *File->Exit*

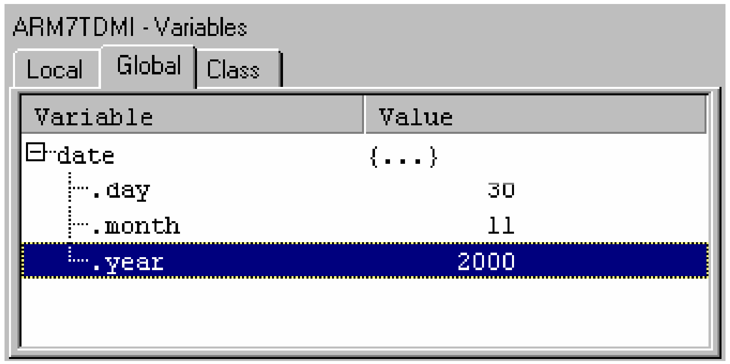


Debugging the example

- Select *Project->Debug* from the IDE menu.
- Select Execute->Go from the menu.
- Select Execute->Go from the menu again.
- This time enter 2005 11 30
  - The program will terminate after it has output.
  - If you use scroll bar of the console window, you will find there is a extra day!
- From the menu reload the image into the debugger.
- Restart the program, and find the function body of nextday()
  - Select *Low-Level Symbols* from the *Processor Views* menu and double clicking the nextday entry.
- Set a breakpoint on the switch statement on line 40 by double clicking in the gray region to the left of the statement.
- Resume execution and enter the date 2005 11 30 again.
- The program will stop at the second breakpoint.

Debugging the example

- Display the local variables.
  - Select *Processor Views→ Variables*, or press *Ctrl+F*.
- Click on the *Global* tab to display the global variables.
  - Right click on the day, month and year fields in turn and select *Format → Decimal* to change the display format of the variables:
- Select *Execute→ Step* (*F10*) to perform the next step in the program.
  - The default path assumes the month has 31 days, and it not correct.

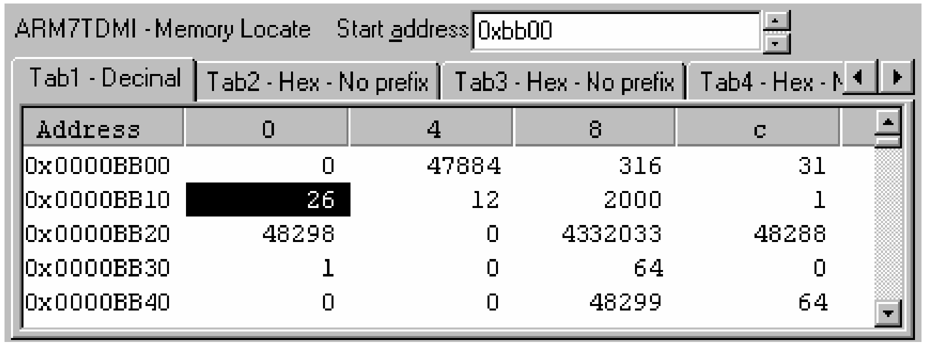


Debugging the example

- Double click on the breakpoint set on line 40 to remove it.
- Set a new breakpoint on line 58 after switch statement.
- Resume program execution,
  - Click on the *Local* tab in the *Variables* window again.
  - You will see that the value of daysInMonth is 31, but we require it to be 30.
  - Double click on the value to edit it and change the value to 30.
- Remove the breakpoint on line 58.
- Restart the program and finish executing the example.
- Check that the output generated by the program is correct.

Viewing registers and memory

- From the menu reload the image into the debugger.
- Select *Execute→ Go* from the menu.
- Set a breakpoint on the **printf** statement on line **29** by double.
- Select *Execute→ Go* from the menu, and this time enter 2005 12 25.
- Open the *Low-Level Symbols* window from the *Processor Views* menu
  - Locate the **date** entry in the *Symbol* column.
  - Right click on the **date** entry and select *Locate using Address*.
  - Right click on the highlighted values in the *Memory* window and select *Format → Other → Size 32 → Decimal*



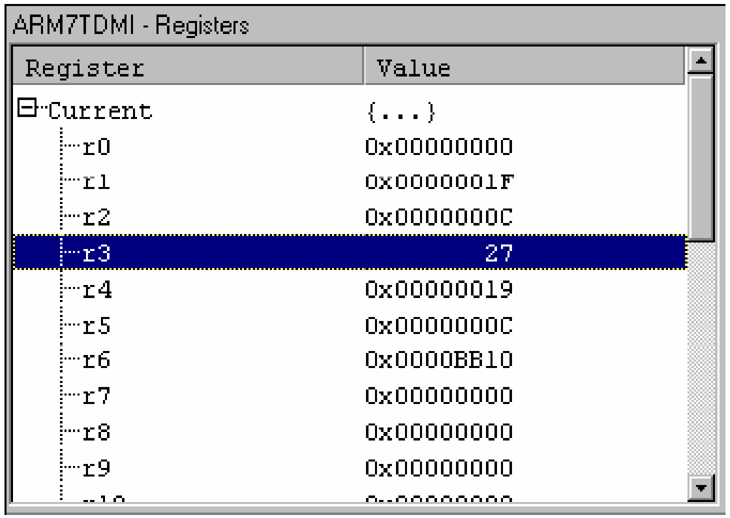
ARM7TDMI - Memory Locate Start address 0xbb00

Tab1 - Decimal | Tab2 - Hex - No prefix | Tab3 - Hex - No prefix | Tab4 - Hex - N

Address	0	4	8	c
0x0000BB00	0	47884	316	31
0x0000BB10	26	12	2000	1
0x0000BB20	48298	0	4332033	48288
0x0000BB30	1	0	64	0
0x0000BB40	0	0	48299	64

Viewing registers and memory

- Open the *Registers* window from the *Processor Views* menu.
- Restart the program, execution will stop at the breakpoint again.
- Right click on the r3 register in the *Register* window, and select
  Format → Decimal from the context menu.
- Use the *Go* button to execute the while loop until r3 has the value 2.
- Double click on the highlighted value 2 in the *Memory* window.
  Change it to 22 and press Enter.
- Use the *Go* button to pass through the while loop until the program ends
- Quit AXD by selecting *File→ Exit*.



Interleaving source code

- Select *Project*→ *Debug* from the IDE menu.
- Start executing the image by selecting *Execute*→ *Go* from the AXD menu (*F5*).
- Right click on the source code window and select *Interleave disassembly*
- Step through the code until you have passed the date entry point and the next two days have been output. (*F10*)
- Quit AXD by selecting *File*→*Exit*.

Using the command line

- Select *Project→ Debug* from the IDE menu to launch AXD.
- Select *System Views → Command Line Interface* from the menu.
- Start program execution by using the go command at the Debug > prompt
  - Debug > go
  - Debug > break month.c|40
  - Debug > go
- Enter the date 2000 11 30 in the *Console* window when prompted.
  - Debug > print daysInMonth dec
  - Debug > format dec
  - Debug > print date.day
  - Debug > print date.month
  - Debug > print date.year
  - Debug > unbreak #2
  - Debug > break month.c|58
  - Debug > go
  - Debug > print daysInMonth
  - Debug > let daysInMonth 30
  - Debug > go
  - Debug > memory @date +0xc 32
  - Debug > step
  - Debug > registers current
  - Debug > unbreak #2
  - Debug > go

Using script file in AXD

- Select *Project*→ *Debug* from the IDE menu to launch AXD
- Ensure the debugger *Command Window* is currently in focus.
  - Debug: obey c:₩work₩demo₩month.txt
- Enter the date 2005 11 30 in the *Console* window when prompted.
- Check the output is correct in the *Console* window then quit the debugger to finish the exercise.

JTAG 소개

Debugger / Emulator

- In-circuit Emulator ?
  - Hardware device used during the development of embedded systems.
  - Have a hardware and a software element, which are separate but tightly interdependent.
  - Allow the software elements to be run and tested on actual hardware.
- Debugger ?
  - used for tracing program execution.
  - used to test and debug software.
  - A tool or program designed to help detect, locate, and correct errors in another program.

Debugger / Emulator

- CPU Emulator
  - Include the same or higher CPU functions than target CPU.
  - Uses emulation memory and provide Trace & Trigger and Profiling & performance monitor functions for powerful debugging.
  - Very high expense, and it is difficult to manufactuere emulator if CPU clock is more than 100Mhz.
  - Since all CPU pins have to be connected with emulator, it is hard to probe.
- JTAG or BDM type debugger
  - Utilize JTAG or BDM port.
  - using a little number of pin (TDI, TDO, TMS, TCK, TRST, VCC, GND...), design and interfaces are simple.
  - As not use target resource (memory, CPU), can debug the target without effect on execution.

Debugger / Emulator

- Embedded trace Macrocell (ETM)
  - Module to provide functions for tracing data and instruction to real time.
  - Include ICE functions. (Trigger & Filter Logic)
  - Capture the core status during operation before and after a specific event occurs.
  - external trace analyzer collects and analyzies the inforamtion.

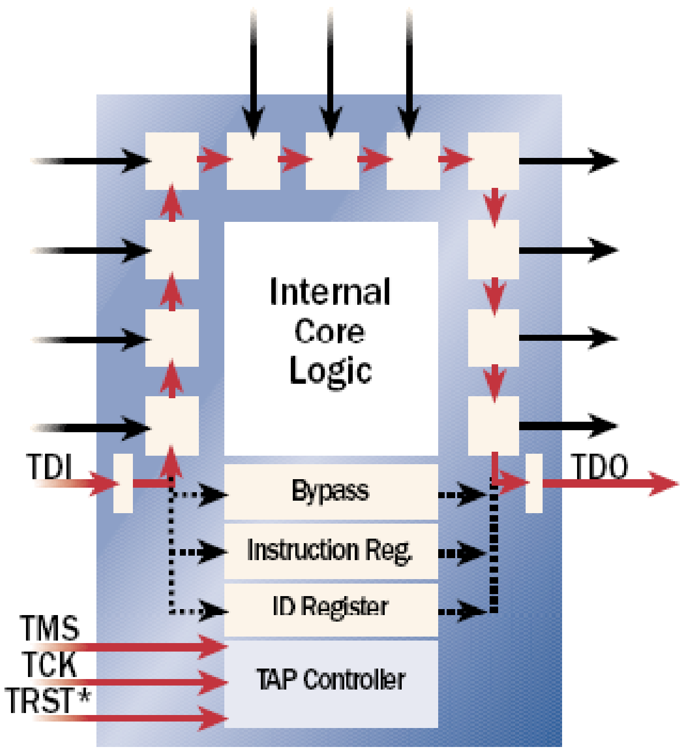
JTAG / IEEE1149.1

- Background
  - Begins the study for testing PCB(Printed Circuit Board)
  - Advences in technology of packaging devices and mounting components.
    - Advent of SMD, High density of pins
    - Difficult to probe pins with previous methods.
- Suggestion
  - Embedded standard logic (TAP) and test pin (Boundary Scan Cell) for testing on a chip.
  - Standardized tests
    - Boundary Scan Architecture
    - Test Access Port(TAP)

Extended JTAG / IEEE1149.1

- Extended JTAG Interface.
  - On-chip Debugging.
  - Memory (DRAM, Flash…) access, logic device control, program fusing…
- JTAG Interface Signal
  - nTRST : test reset
  - TMS : test mode select
  - TDI : test data input
  - TDO : test data output
  - TCK : test clock
- Added additional pins for debugging.
  - nRESET(=nSRST): chip reset
  - VTref (reference voltage)
  - DBGRQ, DBGACK (for external trigger signal)

Device / JTAG Architecture

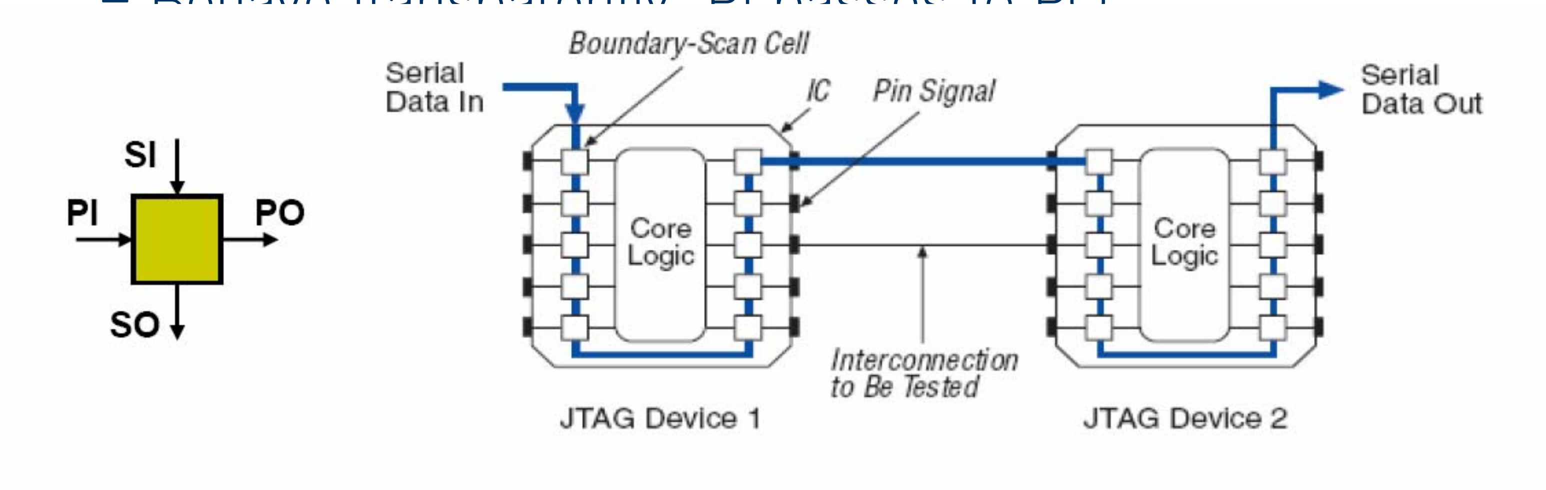


IEEE 1149.1 Device Architecture

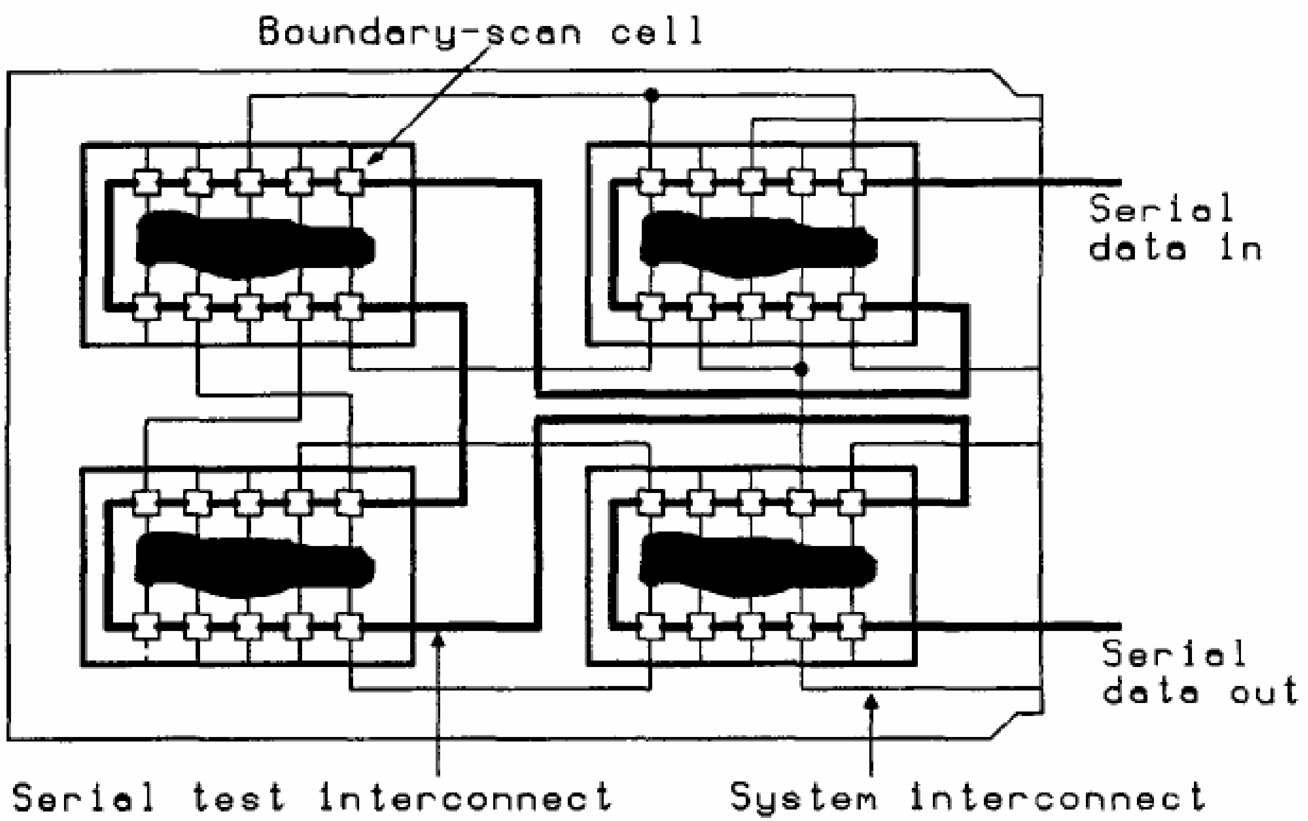
- JTAG block has a CoreLogic for test andothers purpose.
- JTAG block consists of TAP Controller and Registers, and transfers data with TDI, TDO
- JTAG operation is controlled by TAP controller.

Boundary Scan Test

- Boundary-scan cell
  - Capture data on its parallel input PI
  - Update data onto its parallel output PO
  - Serially scan data from SO to its neighbor's SI
  - Behave transparently: PI passes to PO



Boundary Scan Path with Daisy Chain



JTAG Pin Description

- ARM7/ARM9 JTAG Pin Description

Signal	Pin	Pin	Signal
VCCS	1	2	GND
nTRST	3	4	GND
TDI	5	6	GND
TMS	7	8	GND
TCK	9	10	GND
TDO	11	12	nRESET
VCCS	13	14	GND

14 pin connector

Signal	Pin	Pin	Signal
VTref	1	2	NC
nTRST	3	4	GND
TDI	5	6	GND
TMS	7	8	GND
TCK	9	10	GND
RTCK	11	12	GND
TDO	13	14	GND
nSRST	15	16	GND
EDBGRQ	17	18	GND
DBGACK	19	20	GND

20 pin connector

실제 커넥터 모양

JTAG Pin Description

- VTref
  - Input of target reference voltage.
  - Not used for suppling power for JTAG equipment.
- nSRST (=nRESET)
  - Used for detecting reset of target or resetting target CPU.
  - Need pull up resistor to prevent target from resetting unwanted.
- nTRST, TDI, TMS, TCK
  - Need nTRST, TDI, TMS with pull up resistors.
  - Recommend TCK with pull-down resistor.
  - Have to be nTRST with pull up resistor and TCK with pull down resistor for hand-over between tools when using several tools such as multicore debugging environment.

JTAG Pin Description

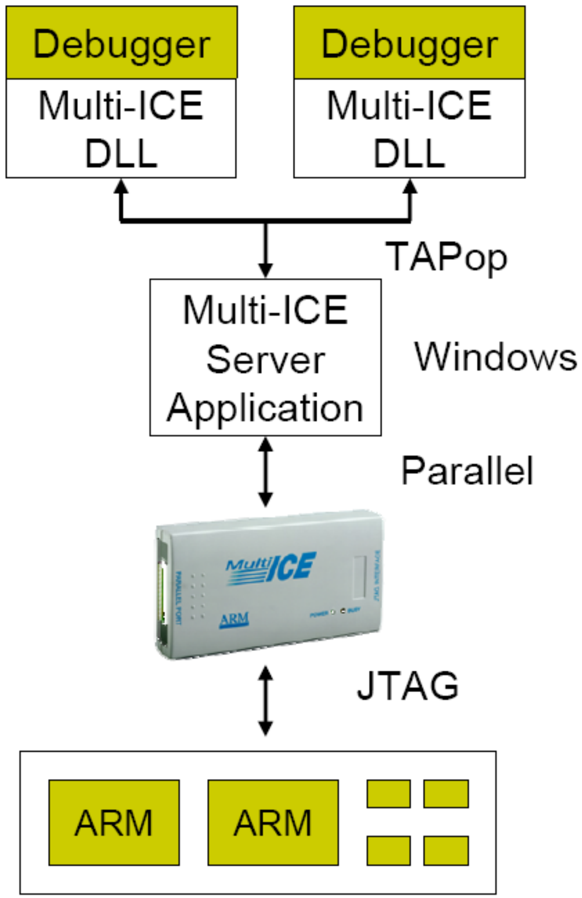
- RTCK
  - Return Test Clock : Input signal from target JTAG port (or processor) to debugger.
  - Clock signal synchronized TCK clock with target processor core clock.
  - have to use this signal to synchronize JTAG port with processor internal clock.
    - for example : specific processor such as ARMXX-S (ARM7TDMI-S, ARM9EJ-S ...)
  - Adaptive Clock Timing
    - Generate next TCK after waiting for returned RTCK as a change of TCK.
    - JTAG clock is variable.
- TDO
  - Tools input, and need not pull-up of pull-down resistor.

Considerations of JTAG Design

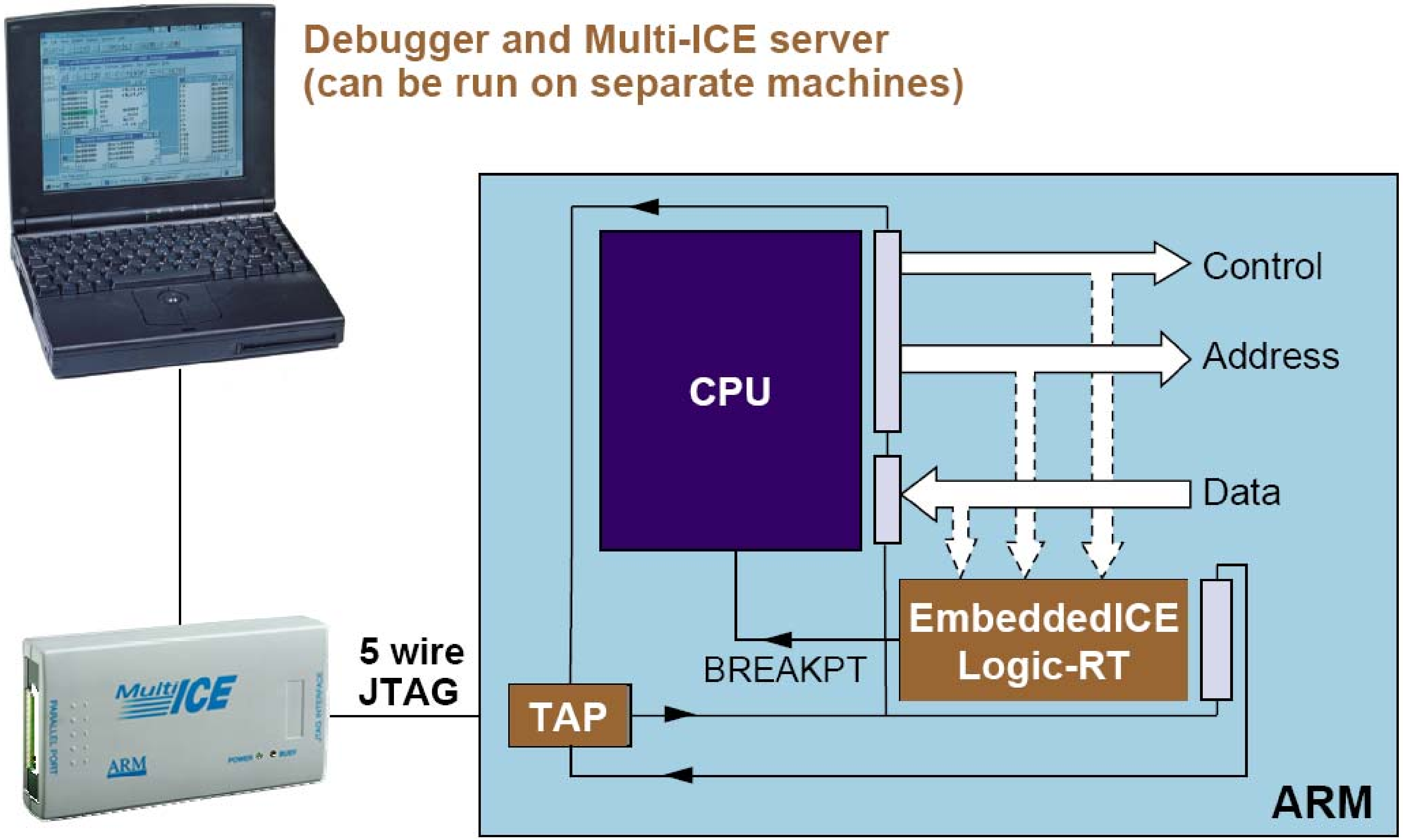
- When design a JTAG circuit, follow the design guide of chip vendor.
- Have to supply VTref with core voltage but not I/O voltage.
- When using ARMxx-S processor core, have to use the RTCK.

Multi-ICE Run Control Unit

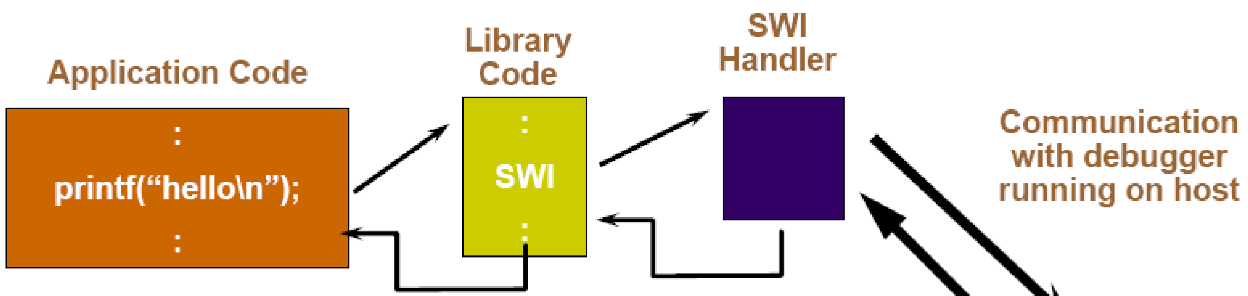
- Change the debugger command to JTAG signal.
- ARM core has to include EmbeddedICE logic.
- Can select JTAG TCK within range form 2.44Khz to 10Mhz.

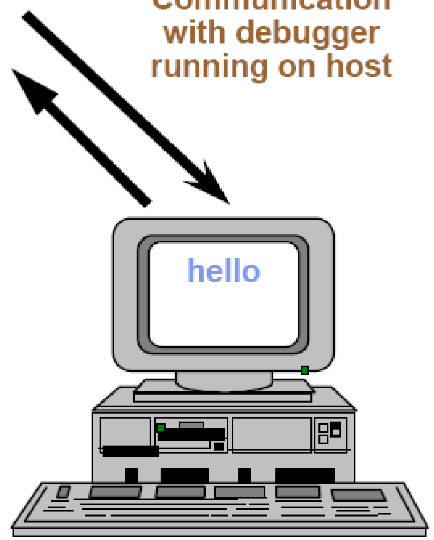


EmbeddedICE Logic



Semihosting





- Library code is executed on ARM target but some low-level I/O is provided by host.
- SWI mechanism serves host access.
  - SWI interface is common in ARMulator, Angel and Multi-ICE.
  - Semihosted programs will run on all of ARM targets without porting.
- have to be connected with Debug tools to provide such a function.