IP Core Lab

ARM/Thumb Interworking

C. W. Jen 任建葳

cwjen@twins.ee.nctu.edu.tw

Outline

- About ARM/Thumb Interworking
- ARM/Thumb Interworking Example
  - ARM/Thumb Interworking in C/C++
  - ARM/Thumb Interworking Example in ASM
  - ARM/Thumb Interworking Example between ARM & C/C++
- LAB Exercise
  - Part A
  - Part B
  - Part C
- Reference Topics & Related Documents

Institute of Electronics, National Chiao Tung University

About ARM/Thumb Interworking

- What is ARM/Thumb Interworking?
  - An application is allowed to be written as a mix of ARM and Thumb instruction sets.
- Why using ARM/Thumb Interworking?
  - Better code density using Thumb.
  - Certain ARM instructions have better performance over Thumb ones.
  - ARM instructions provide some functionality which Thumb does not.
  - Exception handling is required to run under ARM state.
  - Thumb program needs state changes from default ARM state

Institute of Electronics, National Chiao Tung University

About ARM/THUMB Interworking

- System starts in ARM state after reset.
- ARM/Thumb Interworking Veneer
  - A small section of code which performs instructions set change. It's added by linker when a state change is detected.
  - ARM architecture v5T provides method to change processor state without using extra instructions.
- Use ARM compiler *armcc* to compile ARM code. Use Thumb compiler *tcc* to compile Thumb code.

Institute of Electronics, National Chiao Tung University

ARM/Thumb interworking Example

- ARM/Thumb Interworking in C/C++ only.
  - Profiling
- ARM/Thumb Interworking in ASM only.
  - No veneer
  - With veneer
- ARM/Thumb Interworking between C/C++ and ASM.

Institute of Electronics, National Chiao Tung University

ARM/Thumb Interworking in only C/C++

- This program consist of 2 parts
  - *Armmain.c* for main function using ARM instructions set.
    - Print strings
    - Call Thumb function
    - Compiled using ARM C/C++ compiler.
  - *Thumbsub.c* for sub function called by main function using Thumb instructions set.
    - Print strings
    - Return to main function
    - Compiled using Thumb C/C++ compiler.

ARM/Thumb Interworking using C/C++

- Armmain.c

```
#include <stdio.h>

extern void thumb_function(void);

int main(void)
{
    printf("Hello from ARM\n");
    thumb_function();
    printf("And goodbye from ARM\n");
    return (0);
}
```

- Thumbsub.c

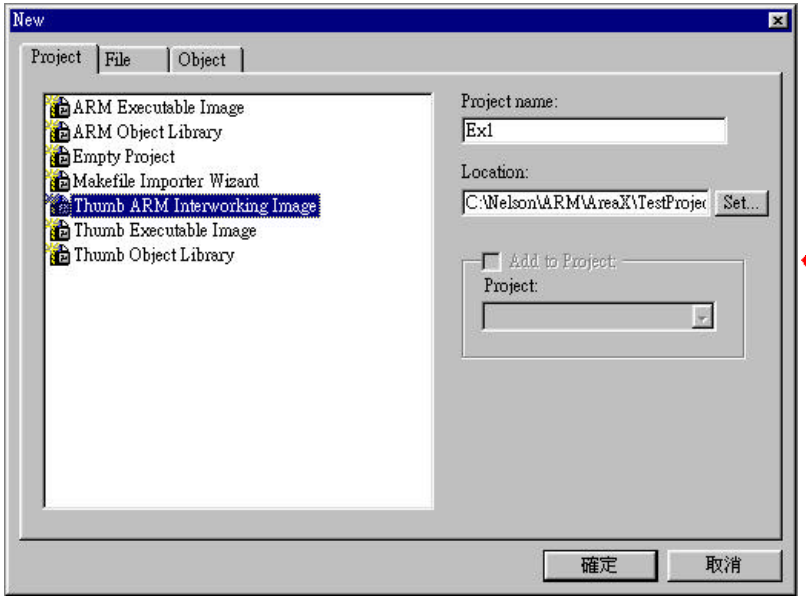
```
#include <stdio.h>

void thumb_function(void)
{
   printf("Hello and goodbye from
          Thumb\n");
}
```

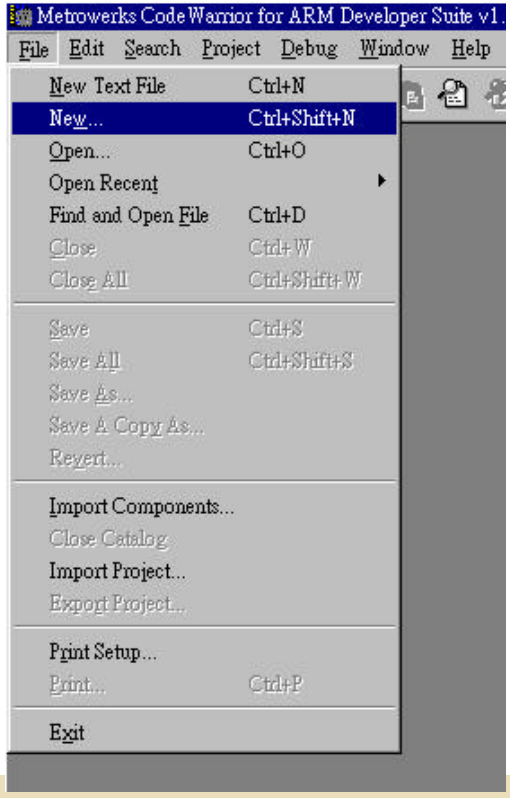
ARM/Thumb Interworking using C/C++

- Building under MS-DOS command line:
  - *armcc -c -g -O1 -apcs /interwork armmain.c*
    - *-c* stands for compile.
    - *-g* generate debug information.
    - *-O1* compile with median optimization.
  - *tcc -c -g -O1 -apcs /interwork thumbsub.c*
  - *armlink armmain.o thumbsub.o -o armtothumb.axf -info veneers -info totals -callgraph -list Ex1.log*
    - *-o* specify output image name
    - *-info veneer* print out veneer information on screen.
    - *-info totals* print out memory size information on screen.
    - *-callgraph* creates static callgraph of functions in an HTML file.
    - *-list XXX.log* redirects information to print in a text file.

ARM/Thumb Interworking using C/C++

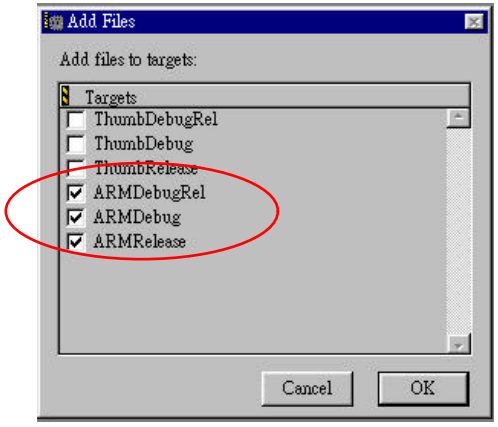
- Building under CodeWarriorIDE:
  - **1.** Start CodeWarriorIDE.
  - **2.** *File>New* to create a new project.
    - **2.1** Select *Thumb ARM Interworking Image* under the *Poject* tab.
    - **2.2** Type the project name, *Ex1* for example.
    - **2.3** Specify the project path.

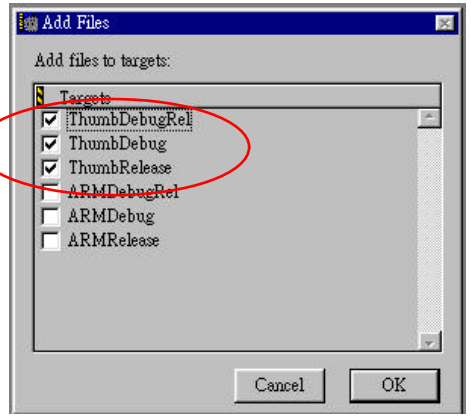




ARM/Thumb Interworking using C/C++

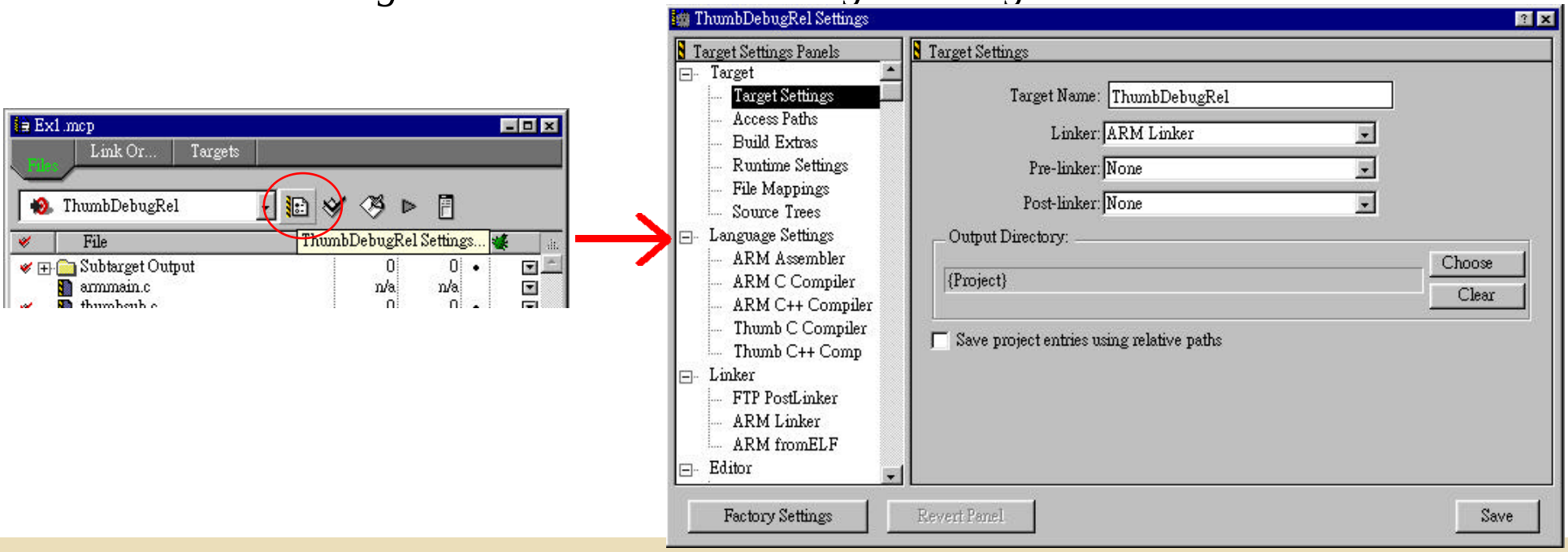
- Building under CodeWarriorIDE (continued):
  - **3.** *Project>Add Files...* to add files to the project.
    - *Please Armmain.c & Thumbsub.c from ARM/ADSv1_1/Examples/Interworking to Ex1 directory first.*
    - **3.1** Add *Armmain.c* for ARM related target.(ARMDebug, ARMRelease, ARMDebRel)
    - **3.2** Add *Thumbsub.c* for Thumb related target.(ThumbDebug, ThumbRelease, ThumbDebRel)





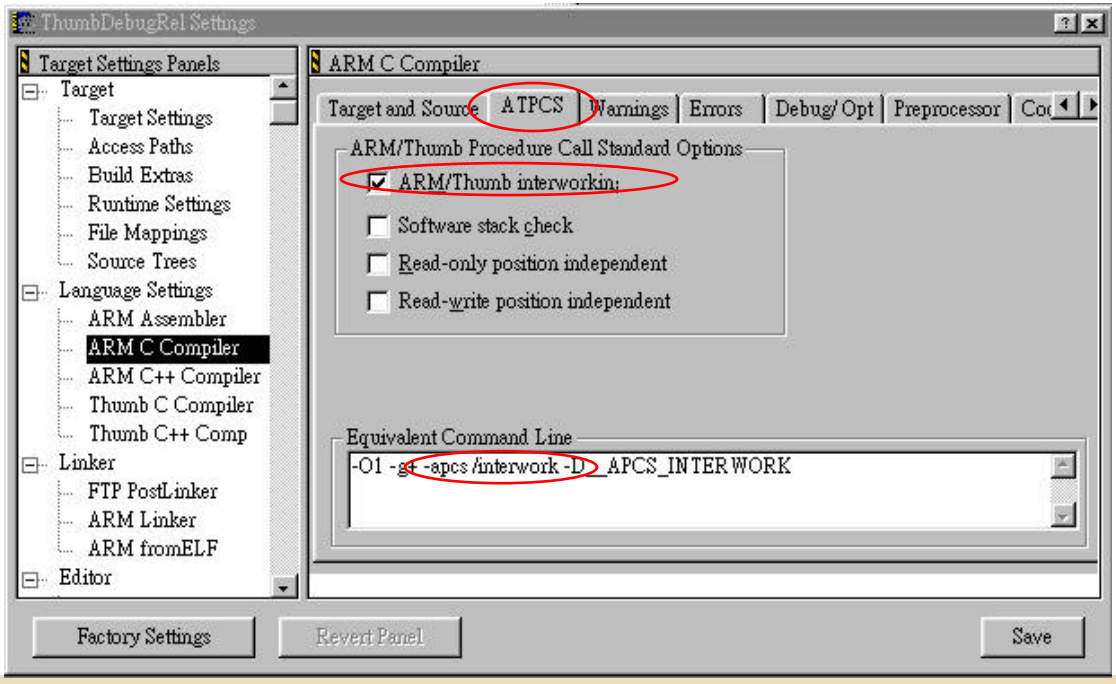
ARM/Thumb Interworking using C/C++

- Building under CodeWarrior (continued):
  - **4.** After adding files to the project, a *Project Management Window* would appear.
    - **4.1** Hit *Build Target Setting* button.
    - **4.2** A *ThumbDebRel Setting* window appears. Click *Language Settings>ARM Assembler* in *Target Setting Panel*.



ARM/Thumb Interworking using C/C++

- Building under CodeWarrior (continued):
  - **4.3** Click *ATPCS* tab. And set as follow:
    - Check *ARM/Thumb Interworking* in *ARM/Thumb Procedure Call Standard Options*.
    - A line " *-apcs /interwork*" would be added to *Equivalent Command line* automatically.

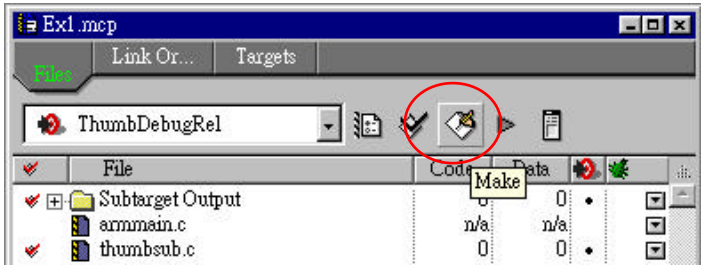


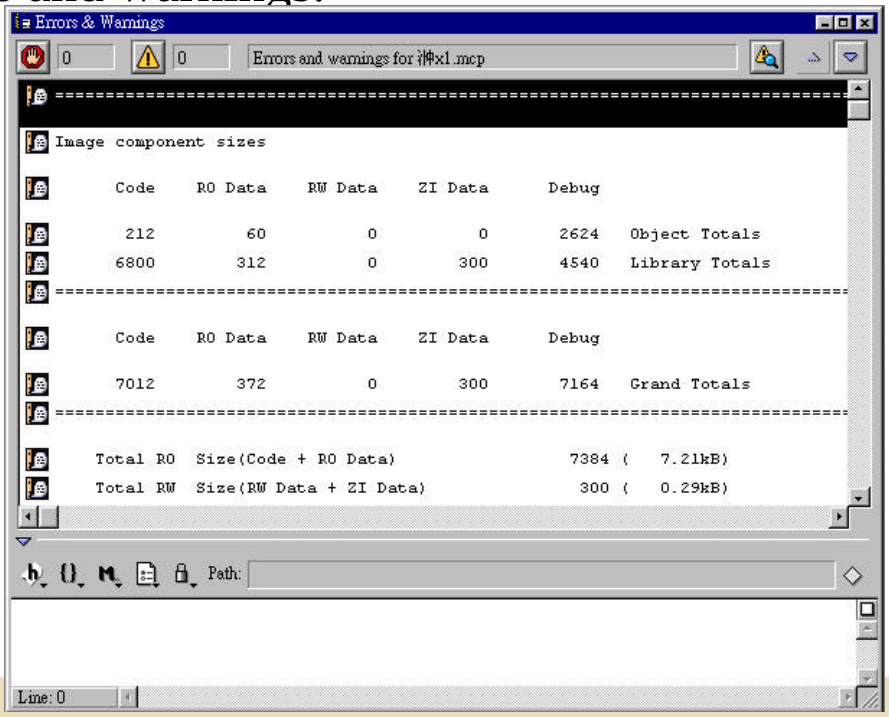
ARM/Thumb Interworking using C/C++

- Building under CodeWarrior (continued):
  - **4.4** Repeat Step **4.**1~**4.2** for the rest of the compilers.
    - *For Thumb ARM Interworking Project, Arm Thumb Interworking check box in ATPCS is automatically checked.*

ARM/Thumb Interworking using C/C++

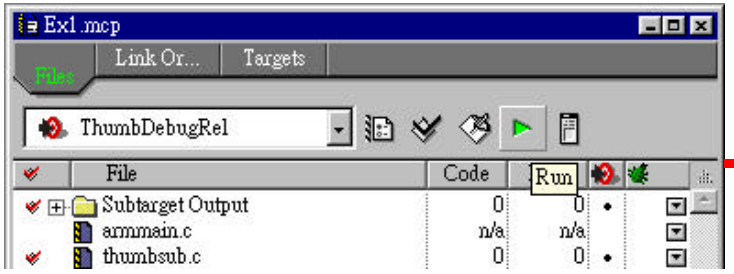
- Building under CodeWarriorIDE (continued):
  - **5.** Hit the *Make* button to compile and link the project.
    - **5.1** A compiling and linking status window would appear to indicate making progress.
    - **5.2** After finishing compiling and linking, a result message windows would appear. Check for errors and warnings.

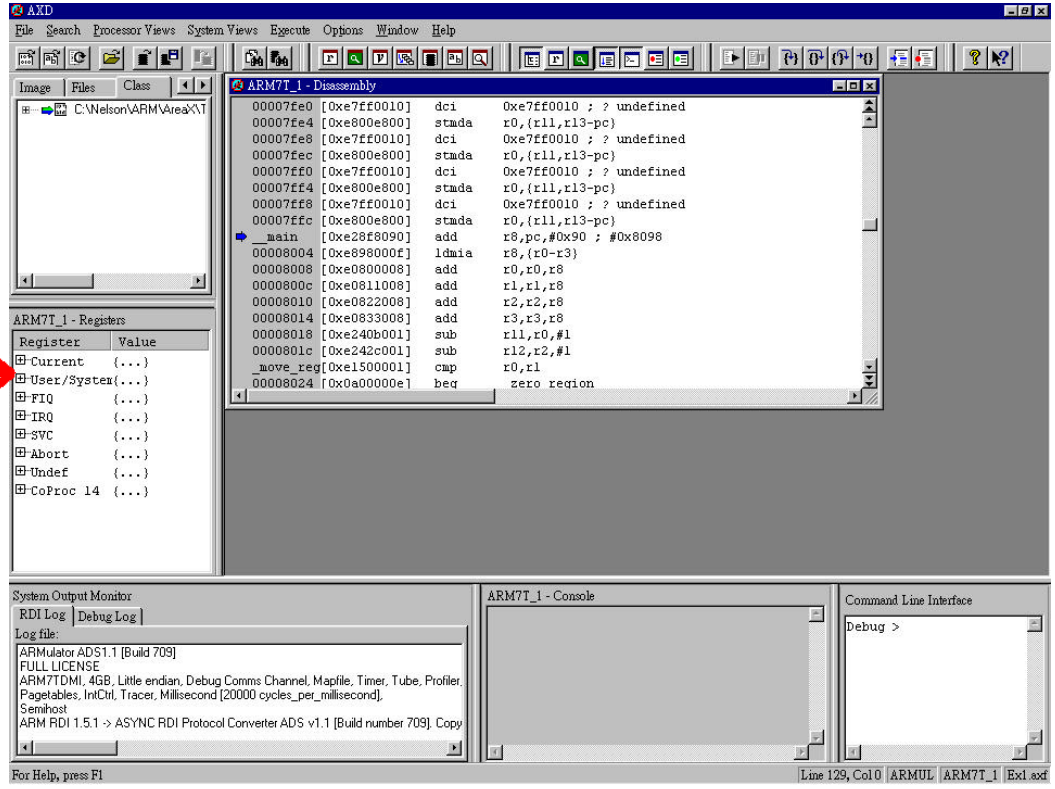




ARM/Thumb Interworking using C/C++

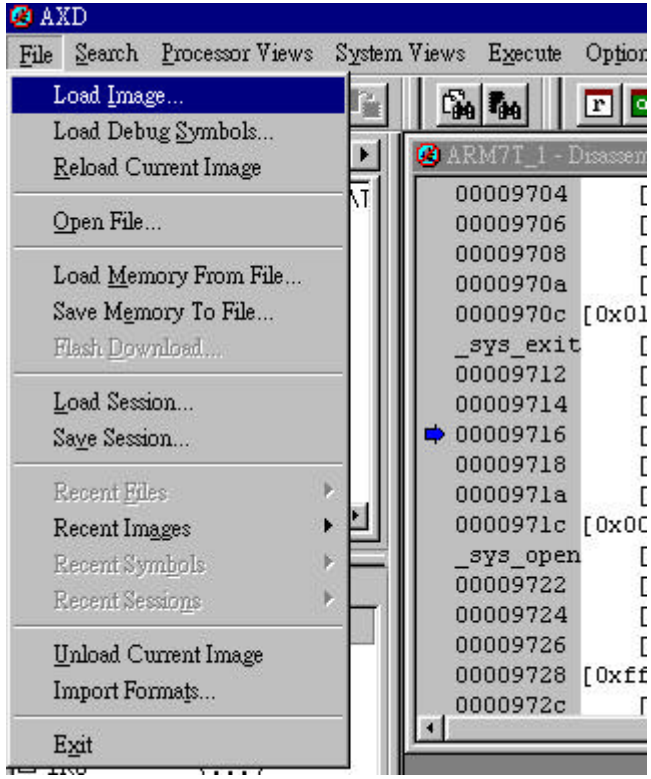
- Building under CodeWarriorIDE (continued):
  - **6.** Hit the *Run* button to run the program.
    - **6.1** This would execute AXD to run the program. The image would be automatic loaded.

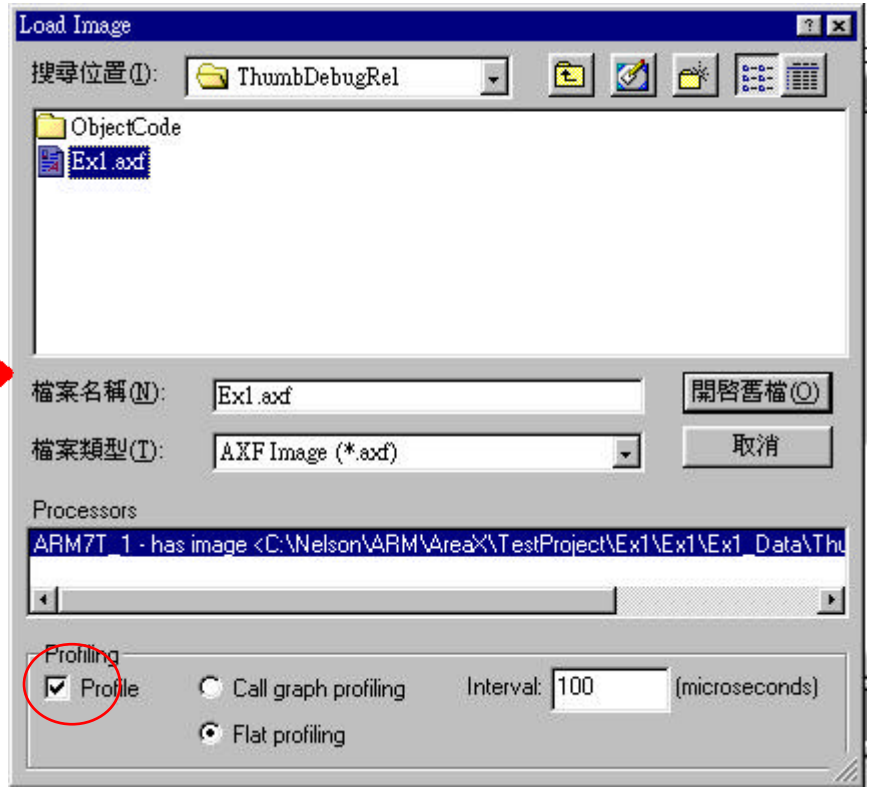




ARM/Thumb Interworking using C/C++

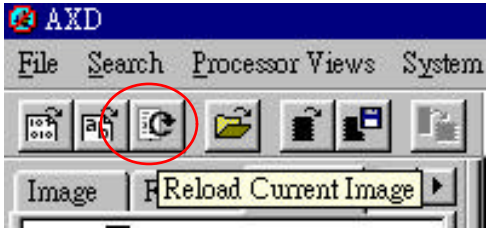
- Running Using AXD
  - **1.** *File>Load Image* to load image file.
    - Load image file *Ex1.axf* in directory *Ex1/Ex1_data/ThumbDebRel*.



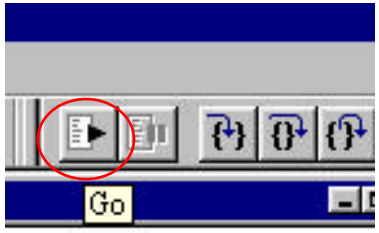


ARM/Thumb Interworking using C/C++

- Running Using AXD (continued):
  - **2.** Hit the *Reload* button in AXD to reload the image.

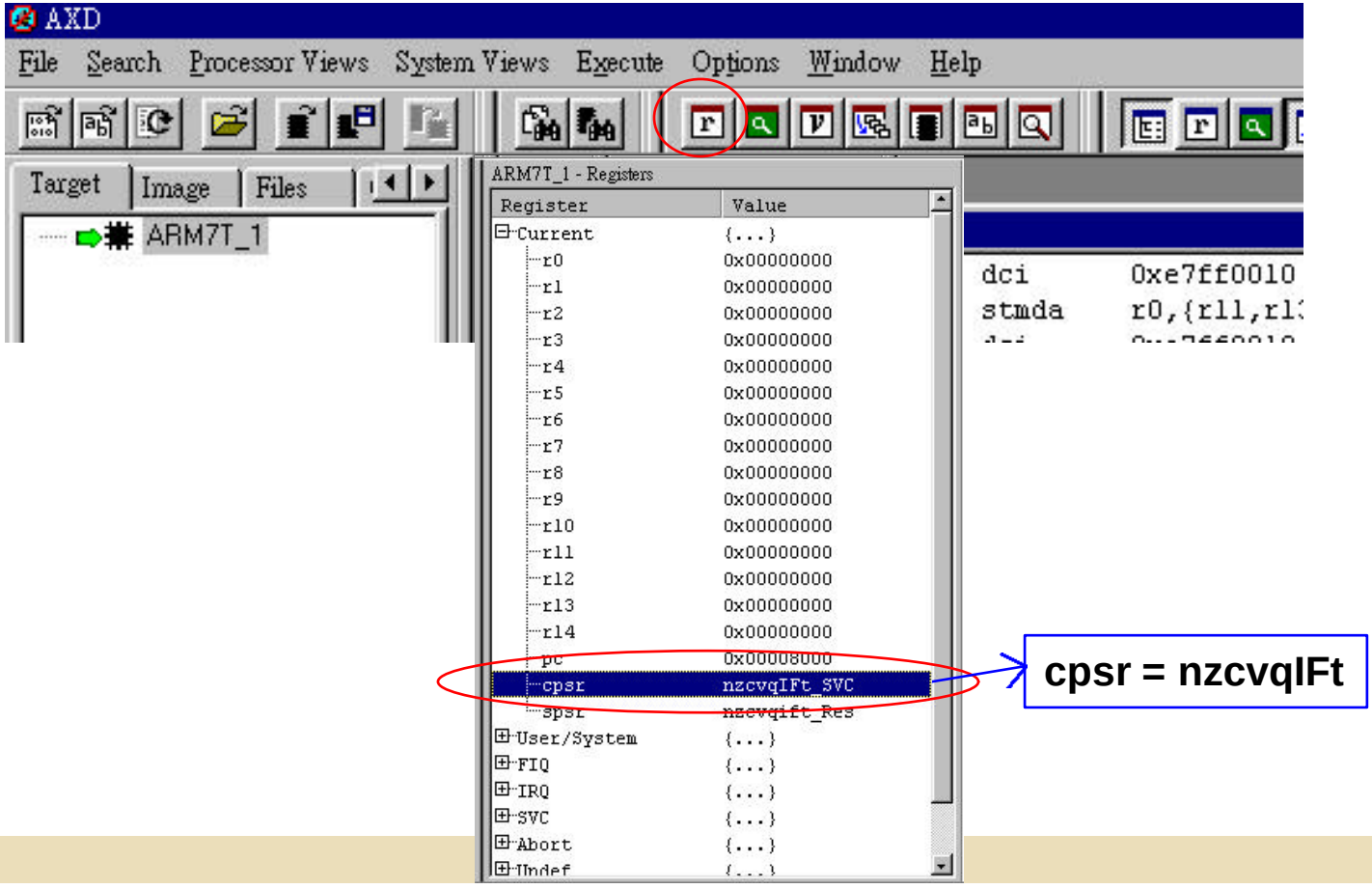


  - **3**. Hit the *Go* button in AXD to run the image.



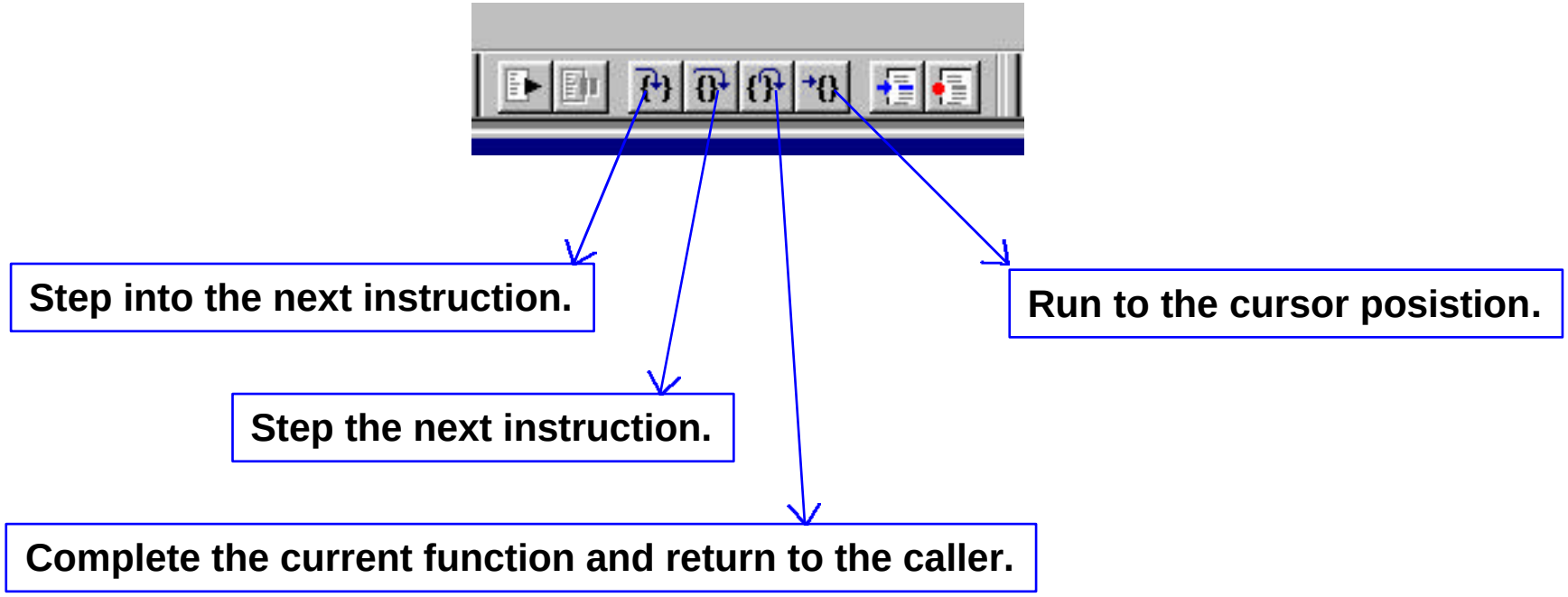
ARM/Thumb Interworking using C/C++

- Running Using AXD (continued):
  - **4.** Hit the *Show Processor Register* button in AXD to show the contents of the processor's register.



ARM/Thumb Interworking using C/C++

- Running Using AXD (continued):
  - **4.** Hit the following buttons to step through the program.



ARM/Thumb Interworking using C/C++

- About Profiling:
  - Profiler samples the program counter and computes the percentage time of each function spent.
  - Flat Profiling: If only pc-sampling info.is present. It can only display the time percentage spent in each function excluding the time in its children.
  - Callgraph Profiling: If function call count info. Is present. It can show the approximations of the time spent in each function including the time in its children.
- Limitations:
  - Profiling is NOT available for code in ROM, or for scatterloaded images.
  - No data is gathered for programs that are too small.

ARM/Thumb Interworking using C/C++

- How to do Profiling:
  - **1.** In AXD, check *Profile* in the *Load Image* window.
  - **2.** Before running, *Options*>*Profile*>*Toggle Profile*.
  - **3.** Run the image.
  - **4.** After running, *Options*>*Profile*>*Write to file*.
  - **5.** After writing profile data file *Ex1.prf* , type *armprof Ex1.prf* under command-line.
  - **6.** The profile information would then be interpreted.

ARM/Thumb Interworking using C/C++

- The t bit in the CPSR (1st bit) would change to indicate which instruction set is being use.
- The linker provided in ADS adds ARM/Thumb Interworking Veneers when it detects ARM/Thumb state change.
  - Veneer: A small section of code which performs instructions set change. It's added by linker when a state change is detected.

ARM/Thumb Interworking in ASM

- This program do computations among registers. No veneer is needed, inteworking instruction change is implemented manually.
- The program consists of 4 parts:
  - main: Generate branch address, and set bit0=1 to arrive at target in Thumb mode. Initial in ARM state.
  - ThumbProg: Set values for r2, r3. Sum r2,r3 to r2. Executed in Thumb state.
  - ArmProg: Set values for r4, r5. Sum r4, r5 to r4. Executed in ARM state.
  - Stop: Terminate the program.

ARM/Thumb Interworking in ASM

- Addreg.s source code:

```
            AREA     AddReg,CODE,READONLY        ;Name this block of code.
            ENTRY                                ;Mark first instruction to call.
main
            ADR r0,ThumbProg +1                          ;Generate branch target address
                                                 ;and set bit 0,hence arrive
                                                 ;at target in Thumb state.
            BX r0                                ;Branch exchange to ThumbProg.
            CODE16                               ;Subsequent instructions are Thumb code.
ThumbProg
            MOV r2,#2                            ;Load r2 with value 2.
            MOV r3,#3                            ;Load r3 with value 3.
            ADD r2,r2,r3                         ;r2 =r2 +r3
            ADR r0,ARMProg
            BX r0
            CODE32                               ;Subsequent instructions are ARM code.
ARMProg
            MOV r4,#4
            MOV r5,#5
            ADD r4,r4,r5
stop        MOV r0,#0x18                         ;angel_SWIreason_ReportException
            LDR r1,=0x20026                      ;ADP_Stopped_ApplicationExit
            SWI 0x123456                         ;ARM semihosting SWI
            END                                  ;Mark end of this file.
```

ARM/Thumb Interworking in ASM

- Building under command line:
  - *armasm -g addreg.s*
  - *armlink addreg.o -o addreg*
- Executing using ARM-synbolic-debugger under command line:
  - *armsd addreg*
  - Type *help* for help info. Type *quit* to quit armsd.
  - Type *step* to step through the program
  - Type *reg* after each instruction execution to display registers.
  - *CPSR* changes from “t” to “T” entering to Thumb state.
    - t: ARM state; T: Thumb state.

ARM/Thumb Interworking in ASM -using veneer

- This program sets the values for r0, r1, r2. Interworking option is added while linking. Veneers are added by linker.
- The program consist of 2 files.
  - Arm.s: Sets the values for r0, r2. Calls for ThumbProg. Executed in ARM state.
  - Thumb.s: Sets the value for r1. Return back to ArmProg. Executed in Thumb state.

ARM/Thumb Interworking in ASM -using Veneer

- Arm.s

```
            AREA      Arm,CODE,READONLY ;Name this block of code.
            IMPORT    ThumbProg
            ENTRY                       ;Mark 1st instruction to call.
ARMProg
            MOV r0,#1             ;Set r0 to show in ARM code.
            BL ThumbProg          ;Call Thumb subroutine.
            MOV r2,#3             ;Set r2 to show returned to ARM.
                                  ;Terminate execution.
            MOV r0,#0x18          ;angel_SWIreason_ReportException
            LDR r1,=0x20026       ;ADP_Stopped_ApplicationExit
            SWI 0x123456          ;ARM semihosting SWI
            END
```

- Thumb.s

```
            AREA Thumb,CODE,READONLY ;Name this block of code.
            CODE16                   ;Subsequent instructions are Thumb.
            EXPORT ThumbProg
ThumbProg
            MOV r1,#2             ;Set r1 to show reached Thumb code.
            BX lr                 ;Return to ARM subroutine.
            END                   ;Mark end of this file.
```

ARM/Thumb Interworking in ASM -using Veneer

- Building under command line:
  - *armasm arm.s*
  - *armasm -16 -apcs /interwork thumb.s*
  - *armlink arm.o thumb.o -o count*
    - *The callee must be compiled with interworking option if it is implemented in a defferent state from the caller.*
- Running under command line:
  - Type *armsd count*.
  - Type *list 0x8000* to list the linked code.
  - Observe that *VenAT$$ThumbProg* is added to the code. This is the veneer added by the linker.

Institute of Electronics, National Chiao Tung University

ARM/Thumb Interworking in ASM -using Veneer

- Linked Code:

```
armsd: list 0x8000
ArmProg
      0x00008000: 0xe3a00001  .... : >  mov      r0,#1
      0x00008004: 0xeb000005  .... :    bl       $Ven$AT$$ThumbProg
      0x00008008: 0xe3a02003  . .. :    mov      r2,#3
      0x0000800c: 0xe3a00018  .... :    mov      r0,#0x18
      0x00008010: 0xe59f1000  .... :    ldr      r1,0x00008018 ; = #0x00020026
      0x00008014: 0xef123456  V4.. :    swi      0x123456
      0x00008018: 0x00020026  &... :    dcd      0x00020026  &...
ThumbProg
+0000 0x0000801c: 0x2102      .!   :    mov      r1,#2
+0002 0x0000801e: 0x4770      pG   :    bx       r14
$Ven$AT$$ThumbProg
+0000 0x00008020: 0xe59fc000  .... :    ldr      r12,0x00008028 ; = #0x0000801d
+0004 0x00008024: 0xe12fff1c  ../. :    bx       r12
+0008 0x00008028: 0x0000801d  .... :    dcd      0x0000801d  ....
+000c 0x0000802c: 0xe800e800  .... :    dcd      0xe800e800  ....
+0010 0x00008030: 0xe7ff0010  .... :    dcd      0xe7ff0010  ....
```

ARM/Thumb Interworking between C/C++ & ASM using veneer

- General rules:
  - If the callee routine is in C, the caller should use the *BL* instruction to make a call , compile it using *-apcs /interwork.*
  - If the callee routine is in assembly language, assemble it with the *-apcs /interwork* option. and return using *BX lr.*

ARM/Thumb Interworking between C/C++ & ASM using veneer

- This program calls the ARM function with a parameter. The ARM function returns that parameter with 4 added.
- The program is consisted of 2 files:
  - thumb.c: The main function. Calls for ARM function with a parameter i. It's implemented in Thumb state using C/C++.
  - Arm.s: Add 4 to the parameter and returns. Called by Thumb main function. Implemented in ARM state using ASM.

ARM/Thumb Interworking between C/C++ & ASM using veneer

- thumb.c

```
#include <stdio.h>
extern int arm_function(int);
int main(void)
{
        int i =1;
        printf("i =%d \n",i);
        printf("And now i =%d \n",arm_function(i));
        return (0);
}
```

- arm.s

```
        AREA Arm,CODE,READONLY          ;Name this block of code.
        EXPORT arm_function
arm_function
        ADD r0,r0,#4        ;Add 4 to first parameter.
        BX LR               ;Return
        END
```

ARM/Thumb Interworking between C/C++ & ASM using veneer

- Building under command line:
  - *tcc -c -apcs /interwork thumb.c*
  - *armasm -apcs /interwork arm.s*
  - *armlink arm.o thumb.o -o add*
- Running under command line:
  - Type *armsd add*.
  - Type *go*.
  - Type *list main* to list the linked code for main function.
  - Type *list arm_function* to list the linked code.
  - Observe that *VenAT$$ThumbProg* is added to the code. This is the veneer added by the linker.

Lab Exercise

- Part A: Interworking in C/C++
  - Thumb Main & ARM Sub
  - Profiling
- Part B: Interworking in ASM
  - No veneer
  - Using veneer
- Part C: Interworking in C/C++ and ASM
  - Modify the last example

Lab Exercise - Part A

- Write a program in C/C++. The main function is implemented in Thumb instructions set. The called function is implemented in ARM state.
- Specifications:
  - Thumbmain: Prints “Hello from thumb main!” & “Goodbye from Thumb main!!”. Calls ARM function. Implemented in Thumb instructions set.
  - Armsub: Prints “Hello from ARM sub.”. Return back to main. Implemented in ARM instruction set.
  - Show the veneers in the linked code and its info.
  - Observe how the t-bit in *CPSR* changes.

Lab Exercise - Part A

- Specification:
  - Load the image with profiling option checked.
    - Use *callgraph* profiling.
  - Toggle profiling and run the program.
  - Save the profiling data to file *a.prf*
  - Execute *armprof a.prf* to see the profiling information.

Lab Exercise - Part B

- Write a program in ASM which swaps the value of [r1,r2], [r3,r4] , no linker added veneers should be added.
- Specifications:
  - Swap function is implemented in ARM instructions.
  - Main Program is implemented in Thumb instructions.
  - Manually change the instruction set using, no linker added veneer.
  - Observe the linked code and the registers.
- Using veneer:
  - Do the above exercise using linker added veneer.
  - Show the veneers added.

Lab Exercise - Part B

- Hints:
  - ARM is in ARM state at the beginning. A change to Thumb state is needed.
  - *ARMASM* doesn't include ARM-to-Thumb header automatically as ARMCC does. You must manually change the state to thumb at initial.
  - Veneers are added when there's a ARM/THUMB or THUMB/ARM procedure call.

Lab Exercise - Part C

- Modify the last example (interworking between C/C++ and ASM using veneer). Such that the main is implemented in ASM, the function is implemented in C.
- Specifications:
  - Main: Implement in ASM using Thumb instructions. Call the subroutine with a parameter.
  - Sub: Implement in C/C++ using ARM instructions. Add 4 to the parameter passed from main and return.
  - Show the linked code.
  - Observe the register.

Lab Exercise - Part C

- Hints:
  - C functions called by ASM code must have a return value.
  - 1st parameter and function return value use *R0 to pass value*.
  - 2nd to 4th parameters use R1 to R3 to pass value.
  - 5th and more parameters should use stack to pass value.
  - Standard I/O in C function does not work (which means you cannot us *printf*() in the C function called by ASM main).

Reference Topics & Related Documents

- Overview of ARM architecture *[ADS_AssemblerGuide 2.2]*
- ARM instruction reference *[ADS_AssemblerGuide 4] [QRC_Armside]*
- Thumb instruction reference *[ADS_AssemblerGuide 5] [QRC_Thumbside]*
- Interworking with ARM & Thumb *[ADS_DeveloperGuide 2.8, 3]*
- About ARM-Thumb Procedure Call Standard (ATPCS) *[ADS_DeveloperGuide 2.1] [ATPCS spec]*
- AXD,armsd *[ADS_DebuggerGuide]*
- Profiling *[ADS_DebuggerGuide 4.7] [ADS_CompilerLinkerUtil 6.4]*
- Mixing C,C++,ASM *[ADS_DeveloperGuide 4]*

Institute of Electronics, National Chiao Tung University