Contents

7. Standard I/O

7.1. 實驗目的

Interface is the basic data transfer method in a system. How to read/write data from interface is very important. In this Lab we introduce students to control IO and learn the principle of polling, interrupt, and semihosting through this Lab.

7.2. 實驗原理

7.2.1. ARM I/O Architecture

All interface are controlled as register based IO, and CPU can control them just like memory control. Output operation is controlled by simple c/c++ function uHALr_WriteLED(..).
But input operation is ho 支援 w interface inform CPU with new data. The details are described bellow.

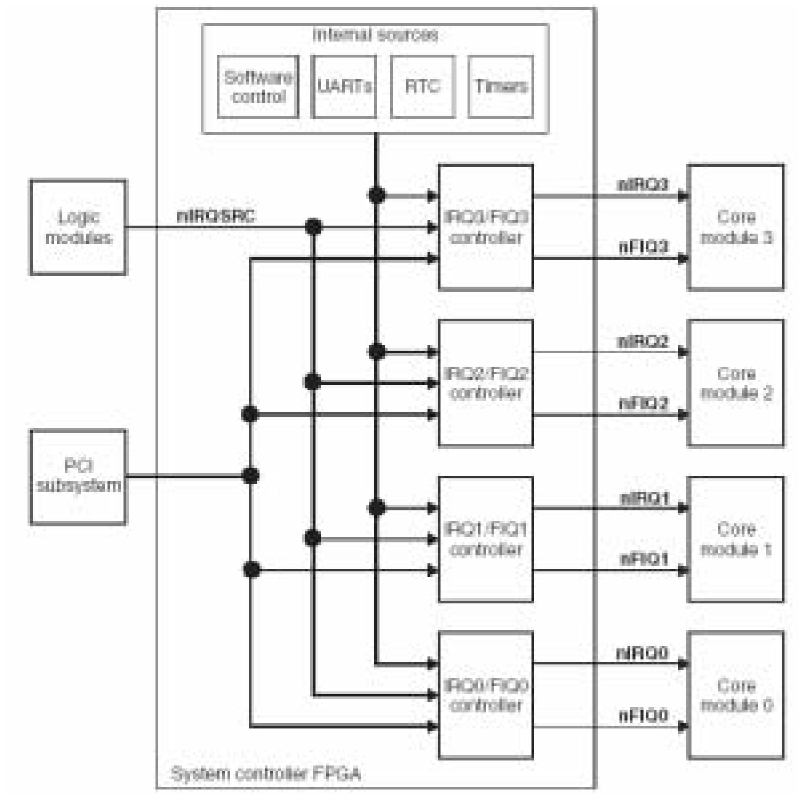


Figure 1 Structure of interrupt

Interface is used to connect CPU and peripherals for specific data transfer. Input components are classified as polling and interrupt.

Polling can be easily implemented by software generation. Polling for an I/O completion can waste a large number of CPU cycles if the processor iterates a busy-waiting loop many times before the I/O completes. But if the I/O device is ready for service, polling can be mush more efficient than is catching and dispatching an interrupt. Describe a device service .For each of these three strategies (pure polling ,pure interrupts ,hybrid) , describe a computing environment in which that strategy is more efficient than is either of the others.

Interrupt is that CPU doesn’t check every peripheral in each cycle. Instead of it, when new data enter peripheral, peripheral send a signal to inform CPU the new data entered.

The interrupt installs the priority of peripheral, when interrupt occurs; CPU use polling to determine which peripheral is interrupt. We always install the first 18 priorities as defaults setting, but it still offer user to install user-defined priority.

7.2.2. Semihosting

ADS prefer to use the higher level control instead of control the detail of IO, so they want to do all the control from host PC. Therefore, they used keyboard and terminal display in host PC as the IO.
The principle of semihosting is that we can read/write all input/output register by checking registers in ARM Integrator through Multi-ICE. Therefore, input from keyboard in the host PC can write data as the input of ARM Integrator. After output from ARM Integrator, all output change will be returned to host PC and show in the terminal of AXD program.
Multi-ICE is following boundary scan standard to read/write the register of desired device. For the reason that all IOs of SOC is register based, to read/write the fitted IO registers is the same with directly input/output the ARM Integrator.
For the semihosting control, users must use this by defining SEMIHOST in their own program. In the ARM design flow, they suppose some function to let this define easier. With these function in uHAL (micro Hardware Abstraction Layer), you can control it without complex settings.

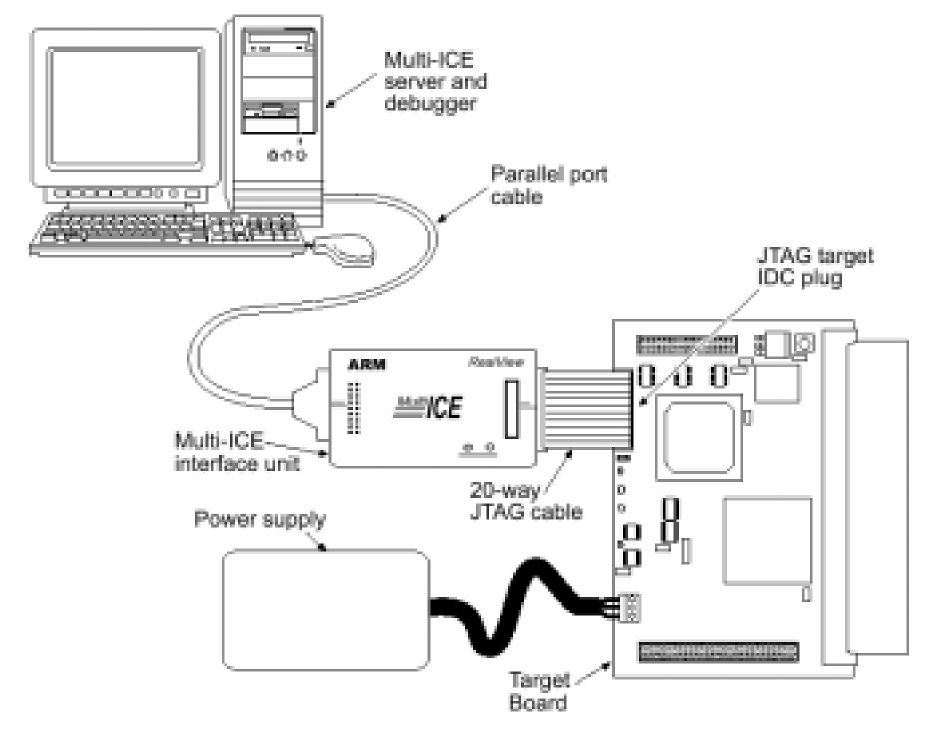


Figure 2 Multi-ICE connector

7.3. 引導實驗

Semihosting

This program controls the Intergator board LED and print strings to the host using *uHal* API. Please trace the linked code and observe how SWI is used.

```
#include "uhal.h"

#ifdef SEMIHOSTED
extern void print_header(void);
extern void print_end(void);
#endif

char *test_name = "LED Flash Tests\n";
char *test_ver = "Program Version 1.1\n";

int main(int argc, int *argv[])
{
    unsigned int count, max, on;
    unsigned int wait, i, j;
    unsigned int ncount;

    count = uHALr_InitLEDs();
    max = (1 << count);

#ifdef SEMIHOSTED
    // init the library
    uHALr_LibraryInit();
    print_header();
    uHALr_printf("\nCheck target for %d flashing LEDs\n", count);
#endif

    while (1)
    {
      // Repeat several times to allow user to move their head
      for (ncount = 0; ncount < 64; ncount++)
      {
          // Do a binary count on the LEDs
          for (i = 0; i < max; i++)
          {
            // which LEDs are on?
            on = (max - 1) & i;

            for (j = 0; j < count; j++)
                uHALr_WriteLED(j + 1, (on & (1 << j) ? 1 : 0));

            // wait a while
            for (wait = 0; wait < 1000000; wait++)
                ;
          }
      }
#ifdef SEMIHOSTED
      // All done, give semihosted a chance to break in..
      uHALr_printf("Press a key to repeat the test.\n");
      uHALr_getchar();
#endif

    }
    return (OK);
}

// End of file - led.c
```

7.3.1. 實驗步驟

1. Start CodeWarrior IDE.
2. Select **File** → **New** to create a new project (Figure 3).
   (1) Select **ARM Executable Image** under the Project tab.
   (2) Type the project name, Semihosting for example.
   (3) Specify the project path. You can see the result in Figure 3.

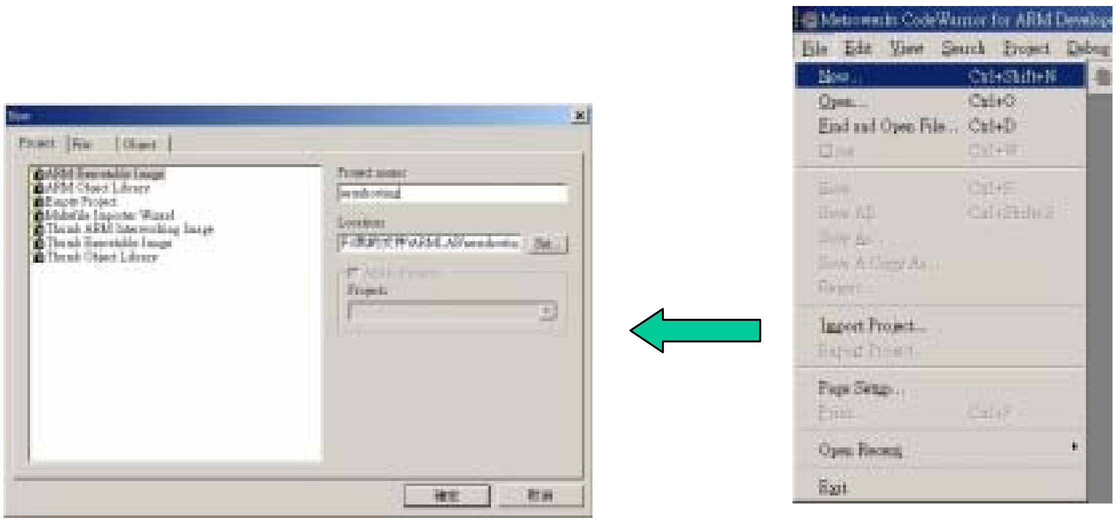
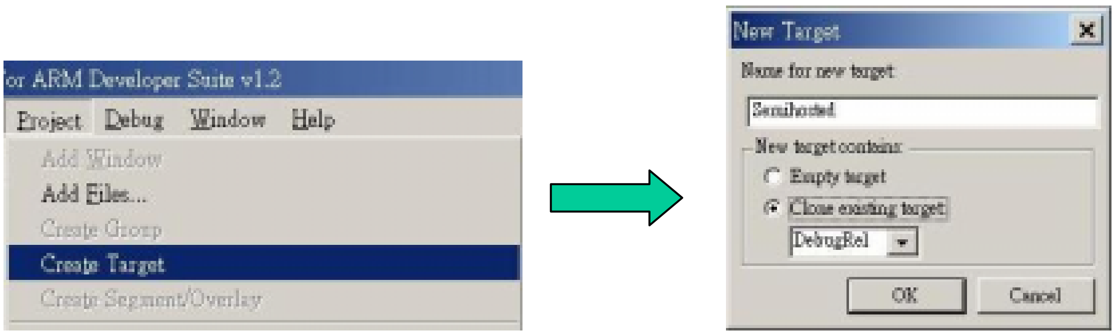


Figure 3 New dialog box

3. Building under CodeWarrior IDE.
   (1) A Project Management Window appears. Click on the Targets tab.
   (2) Select **Project** → **Create Target** (Figure 4).
   (3) A **New Target** window appears.
       (a) Type Semihosted in **Name for new target** tab.
       (b) Click **Clone existing target** in the **New target contains**. We show this in Figure 4.
   (4) Click **OK**.



Name for new target: Semihosted
New target contains: ( ) Empty target (●) Clone existing target: DebugRel

Figure 4 Add New Target to the files

4. Target Semihosted Settings.
   (1) Click the build target drop-down list to select the **Semihosted** target (Figure 5).

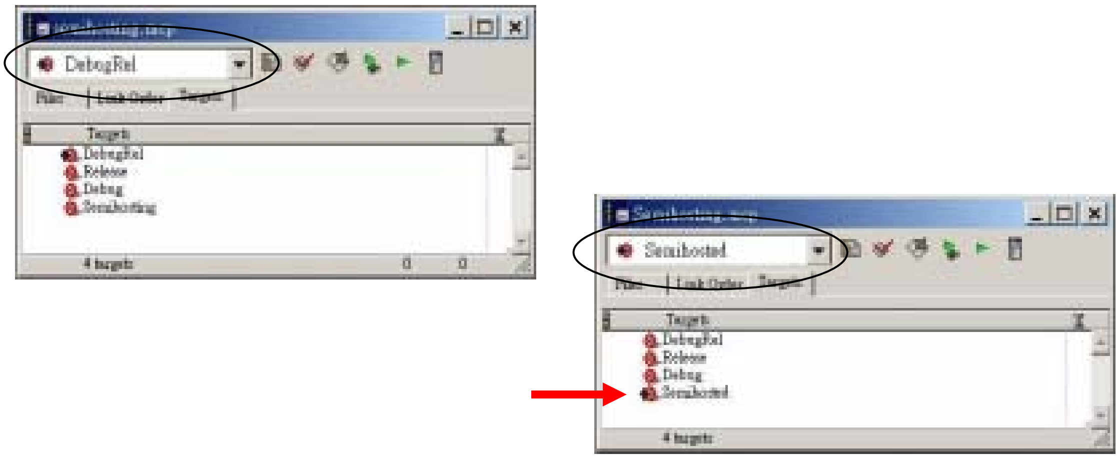
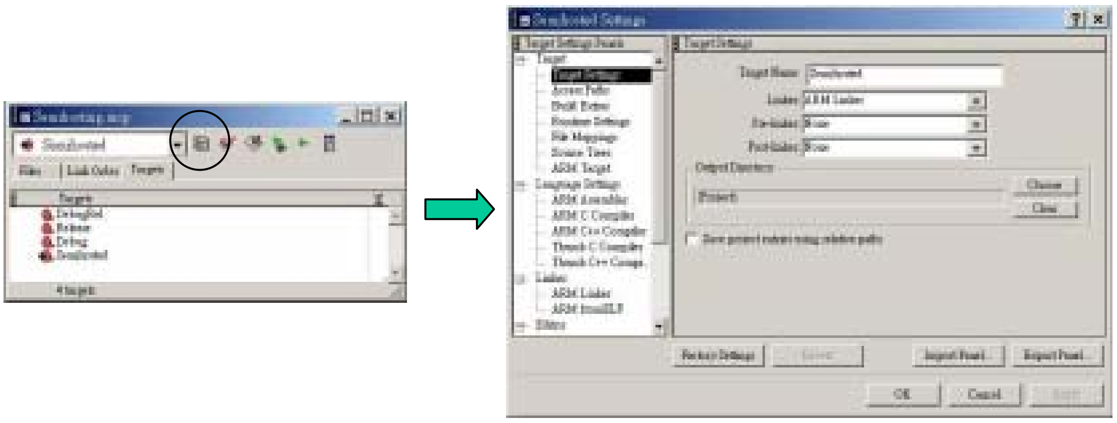


Figure 5 Semihosted Target

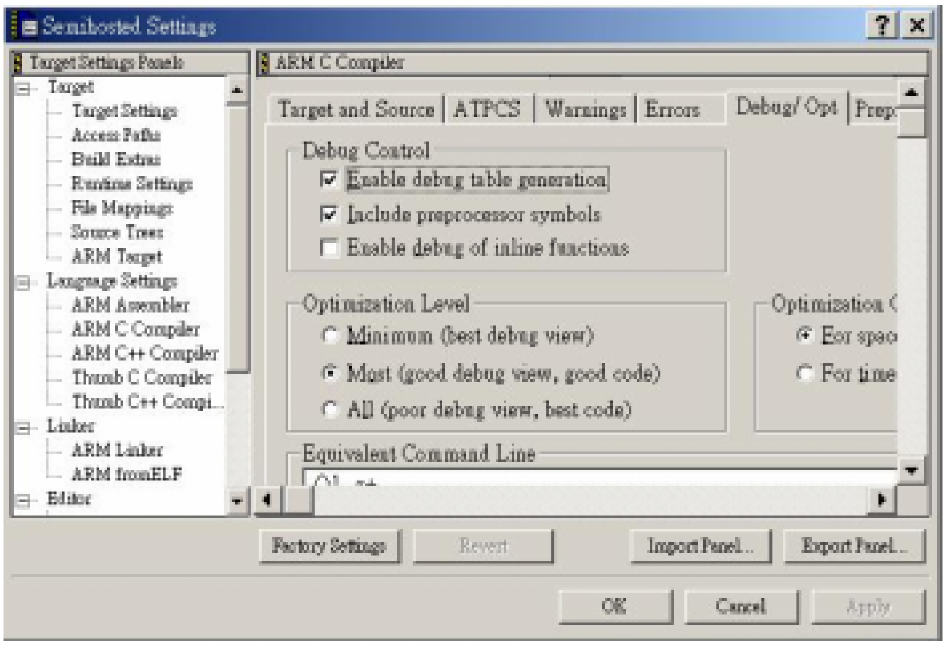
(2) Hit the Build Target Setting button.
(3) A **Semihosted Settings** windows appears (Figure 6). Click **Target Settings** in the **Target**. Make sure Linker is **ARM Linker**.



Target Name: Semihosted
Linker: ARM Linker
Pre-linker: None
Post-linker: None
Output Directory: {Project}

Figure 6 DebugRel Settings

(4) Click **ARM C compiler** in the **Language Settings** (Figure 7).
(5) Make sure **Enable debug table generation** and **Include preprocessor symbols** in **Debug Control** are chosen. Click **Most (good debug view, good code).**
(6) Click **Apply**.


Semihosted Settings

Target Settings Panels: Target (Target Settings, Access Paths, Build Extras, Runtime Settings, File Mappings, Source Trees, ARM Target), Language Settings (ARM Assembler, ARM C Compiler, ARM C++ Compiler, Thumb C Compiler, Thumb C++ Compi...), Linker (ARM Linker, ARM fromELF), Editor

ARM C Compiler — Target and Source | ATPCS | Warnings | Errors | Debug/ Opt | Prep

Debug Control
- [✓] Enable debug table generation
- [✓] Include preprocessor symbols
- [ ] Enable debug of inline functions

Optimization Level
- ( ) Minimum (best debug view)
- (●) Most (good debug view, good code)
- ( ) All (poor debug view, best code)

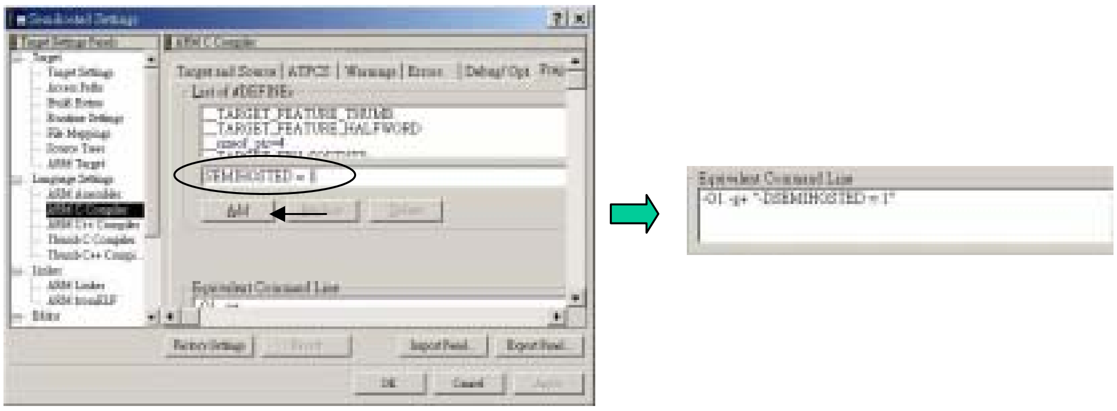
Optimization
- (●) For spac
- ( ) For time

Equivalent Command Line

Figure 7 ARM C Compiler Panel

(7) Click **Preprocessor** tab in the **ARM C Compiler** (Figure 8). Type SEMIHOSTED = 1 into the text field beneath the List of #DEFINEs and click **Add** to define the SEMIHOSTED. Figure 8 is showed the result. The Equivalent Command Line text box displays the result.

(8) Click **Apply** to apply your changes.


List of #DEFINEs

```
__TARGET_FEATURE_THUMB
__TARGET_FEATURE_HALFWORD
__sizeof_ptr=4
```

SEMIHOSTED = 1

Equivalent Command Line

```
-O1 -g+ "-DSEMIHOSTED = 1"
```

Figure 8 Semihosting defined

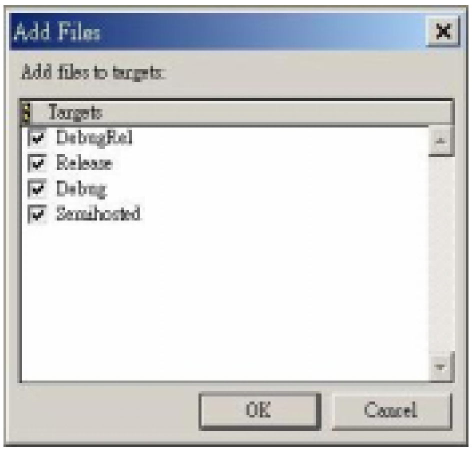
5. Adding source files to the project.

(1) Copy file led.c, pr_header.c, and uHAL_u_.a to your semihosting directory.

(2) Select **Project → Add Files.** Add Files dialog is shown in Figure 9.

(3) Navigate to the semihosting directory and choose above three files.

(4) Click **Open**. Then Add all files to targets.

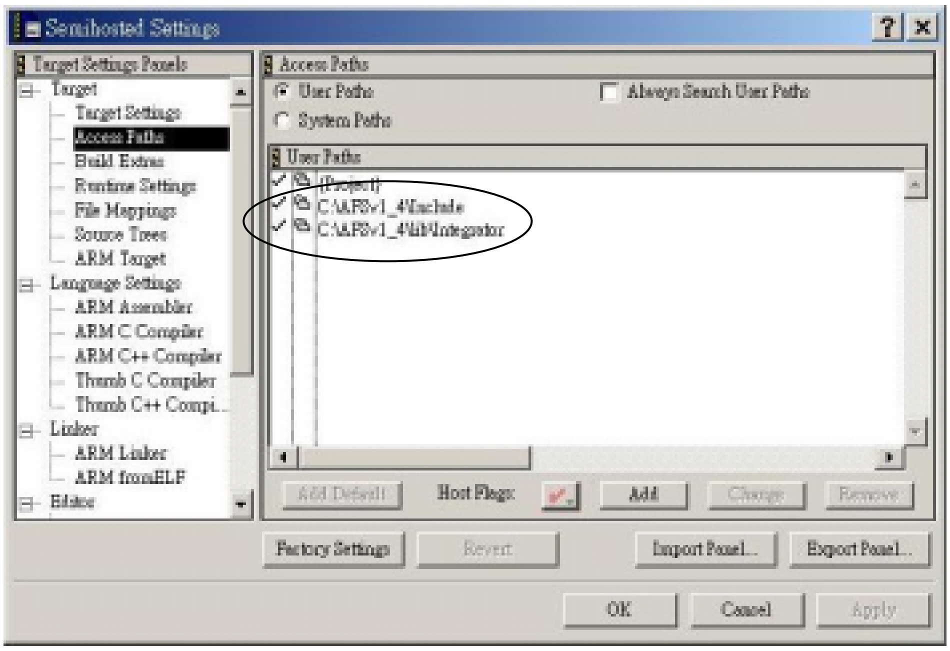


Add files to targets:

Targets
✓ DebugRel
✓ Release
✓ Debug
✓ Semihosted

Figure 9 Add Files

6. Set Access Path.
   (1) Double click Semihosting on the targets and Semihosted Settings appears (Figure 10).
   (2) Click **Access Paths** in the Target. Hit Add button to add following path:
   (a) C:\AFSv1_4\Include.
   (b) C:\AFSv1_4\Lib\Integrator.



Access Paths

(● User Paths) ○ System Paths ☐ Always Search User Paths

User Paths
✓ {Project}
✓ C:\AFSv1_4\Include
✓ C:\AFSv1_4\Lib\Integrator

Figure 10 Access Path configuration

7. Delete Debug, DegRel and Release targets.
   (1) Click Target tab in Project Managing Window.
   (2) Click on targets you want to delete.
   (3) Press Del key on the keyboard.
   (4) Leave only target Semishoted not delete.

8. Hit the **Make** button to compile and link the project.
   (1) A compiling and linking status windows would appear to indicate making progress.
   (2) After finishing compiling and linking, a result message windows would appear (Figure 11). Check for errors and warnings.

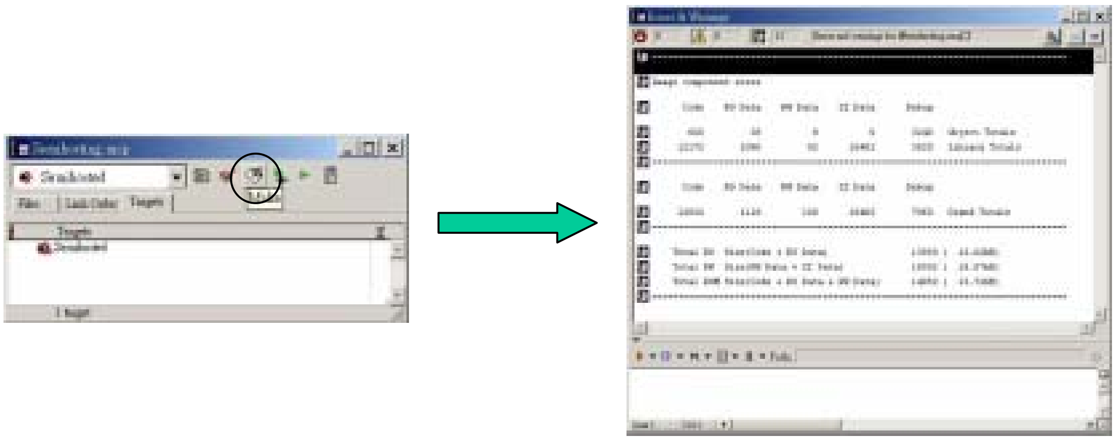


Figure 11 Make the project from the Project Window

9. Hit the **Run** button to run the program (Figure 12).
   (1) The CodeWarrior IDE calls AXD debugger to load and execute the image.

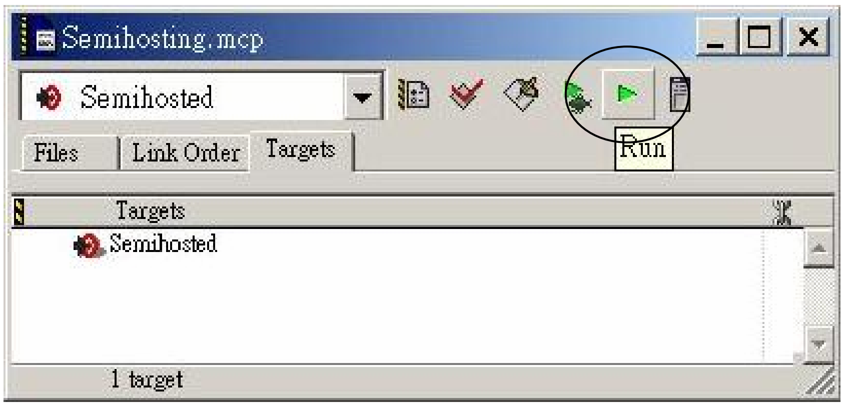
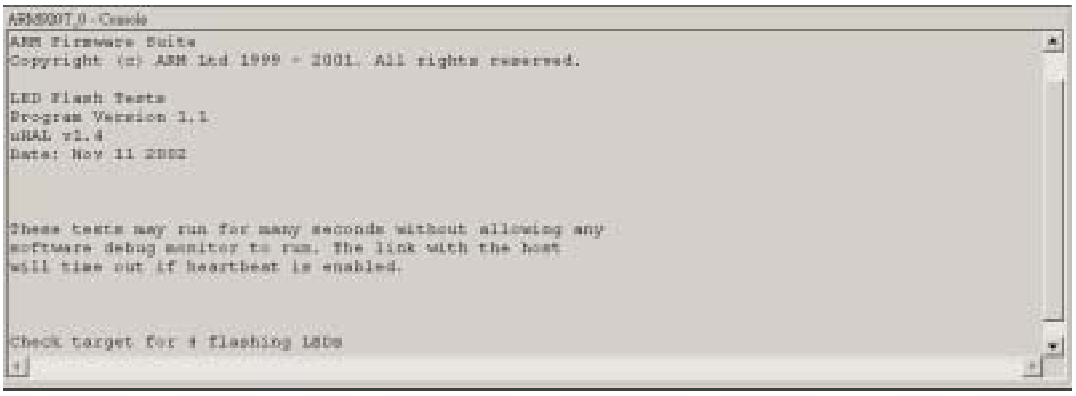


Figure 12 Debug the project from the project window

   (2) If using Multi-ICE with Integrator, you'll see LEDs on the integrator flashing.
   (3) The COM port of the Integrator is reserved when using with Multi-ICE. Hence data is transmitted through Multi-ICE.



```
ARM Firmware Suite
Copyright (c) ARM Ltd 1999 - 2001. All rights reserved.

LED Flash Tests
Program Version 1.1
uHAL v1.4
Date: Nov 11 2002


These tests may run for many seconds without allowing any
software debug monitor to run. The link with the host
will time out if heartbeat is enabled.


Check target for 4 flashing LEDs
```

Figure 13 Message on the CodeWarrior

7.4. 實驗要求

Modify the LED example. When it counts, we press any key to stop counting and then press any key to continue counting numbers.

7.5. 問題與討論

1. Explain the advantage and disadvantage of polling & interrupt.
2. A system can be divided into hardware, software, and firmware. Which one contains μHAL.

7.6. 參考文件及網頁

- SWI Interface [ADS_DebugTargetGuide 5.1.1]
- SWI Handling [ADS_DeveloperGuide 5.4]
- Semihosting [ADS_DebugTargetGuide 5]
- Building Semihosted application [ADS_CompilerLinkerUtil 4.2]
- Semihosting directly dependent functions [ADS_CompilerLinkerUtil Table4-1]
- Semihosting indirectly dependent functions [ADS_ComplierLInerUtil Table4-2]
- I/O supported functions using semihosting SWI [ADS_CompilerLInkerUtil Table4-13]
- uHAL API [DUI_0102D_AFS_REF 2] [DUI_0136_AFS_USER 2]