Lab 3 Software Quality Measurement

Table of Contents

IP Core Design
2002.9

1. Introduction

This Lab gives instructions to perform a variety of measurement skills of you ARM programs. The following instructions are based on the demonstration program that runs the **Dhrystone** test software, which is the same to that used in Lab 2.

The skills you will learn:

- Memory requirement of the program
- Profiling: Build up a picture of the percentage of time spent in each procedure.
- Evaluate software performance prior to implement on hardware

2. Memory requirement of the program

1. Make the directory C:\ARMSoC\Lab_03\

2. Copy ARM Project file **My_Poeject.mcp** from C:\ARMSoC\Lab_01\My_Poeject\ to **C:\ARMSoC\Lab_03** or copy **dhryansi.mcp** from the **dhryansi** directory in the C:\Program Files\ARM\ADSv1_1\Examples\dhryansi\ and rename it as **My_Poeject.mcp**.

3. Double click on **My_Poeject.mcp**.

4. ***Make*** **My_Poeject.mcp**
   - 4.1 A compiling and linking status window would appear to indicate making progress.
   - 4.2 After finishing compiling and linking, the ***Errors and Warnings window*** would appear, as shown in Figure 1.

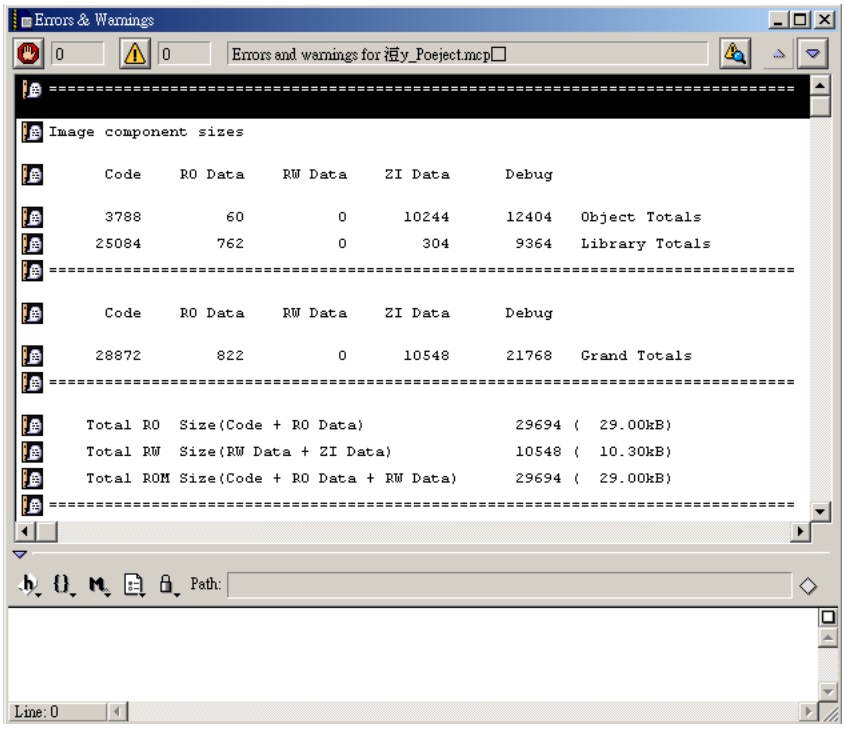

Figure 1. Show Code and Data Size by using -info totals.

5. Select ***DebugRel Settings*** from ***Project Window***

5.1 Change the ARM Linker option from ***–info totals*** to ***-info sizes*** in Equivalent Command Line, as shown in Figure 2.

5.2 Make the project again and then check the ***Errors and Warnings window***. A much detailed memory requirement for each object file and library file is listed.

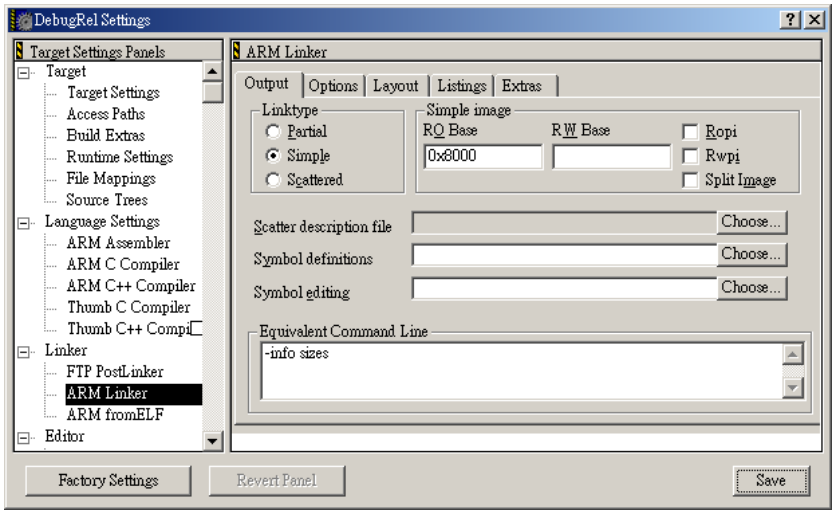

Figure 2. DebugRel Settings for ARM Link output

3. Profiling

1. After making the project, launch AXD.

1.1 Select ***File➔Load Image*** to load image file from ***C:\ARMSoC\Lab_03\My_Poeject_Data\DebugRel\My_Poeject.axf.***

1.2 Check the ***Profile*** checkbox, as shown in Figure 3.

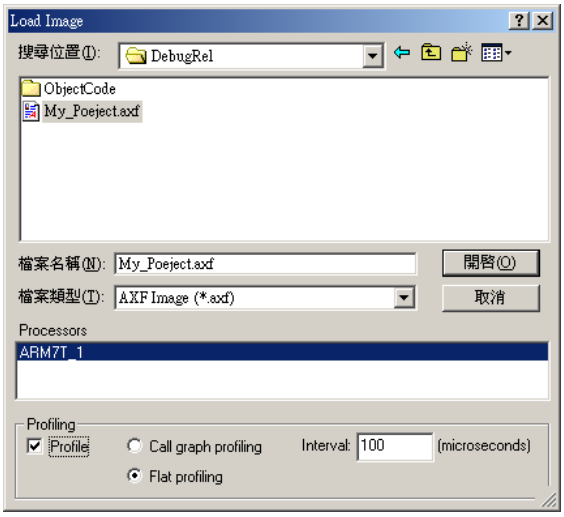

Figure 3. Load image with Profiling functionality.

1.3 Select ***Options ➔ Profiling ➔ Toggle Profiling*** if necessary to ensure that Toggle Profiling is checked in the Profiling submenu of the Options menu.

1.4 Select ***Options➔Profiling➔Clear Collected*** to clear previous profiling data if necessary.

1.5 Execute your program (Hit the ***Go*** button). Type ***8000*** in the ***Console Window***

when be asked.

1.6 When the image terminates, select ***Options ➔ Profiling ➔ Write to File***.

1.7 A ***Save*** dialog appears. Enter a file name (e.g., ***dhryansi***) and a directory as necessary. Click the ***Save*** button.

1.8 Type ***armprof dhryansi.prf*** under command line to view the profiling information.

Note: You cannot display profiling information in AXD. Use the Profiling functions on the Options menu to capture profiling information, then use the ***armprof*** command-line tool.

To collect information on a specific part of the execution:

1. Load (or reload) the program with profiling enabled.
2. Set a breakpoint at the beginning of the region of interest (e.g., start of for loop at line number ***155***), and another at the end (e.g., end of for loop at line number ***199***).
3. Execute the program as far as the beginning of the region of interest.
4. Clear any profiling information already collected by selecting ***Options ➔ Profiling ➔ Clear Collected***, and ensure that Toggle Profiling is checked.
5. Execute the program as far as the breakpoint at the end of the region of interest.
6. Select ***Options ➔ Profiling ➔ Write to File*** and specify the name of a file in which to save the profiling information.

4. Performance benchmarking

This section is based on ARM Application Note 93: Benchmarking with ARMulator, March 2002.

4.1 Cycle counting example: Dhrystone using the ARM7TDMI

1. Select ***File➔Load Image*** to load image file from C:\ARMSoC\Lab_03\My_Poeject_Data\DebugRel\My_Poeject.axf.

2. Within AXD ***select Options ➔ Configure Target…***

   2.1 Select ***ARMUL*** as the target and click on the ***Configure*** button.

   2.2 Select the ***ARM7TDMI*** as the processor variant and ensure that the check box for ***Floating Point Emulation*** is cleared, then click ***OK***.

   2.3 Choose ***OK*** in the configuration dialog.

   2.4 Click ***Yes*** when asked to reload the last image

3. Select ***Processor Views ➔ Low Level Symbols*** and locate ***Proc_6*** in the Low Level Symbols window.

   3.1 Right-click on it and select ***Locate Disassembly***.

   3.2 Place a breakpoint on this line in the ***Disassembly window***.

4. Click on the ***Go*** button (or press F5) to begin execution, the program will run to ***main***.

   4.1 Click on ***Go*** again, the program will run, when prompted, request at least ***two*** runs through Dhrystone. The program will then run to the breakpoint at ***Proc_6*** and stop.

5. Select ***System Views*** ➔ ***Debugger Internals*** and click on the ***Statistics tab*** in the Debugger Internals window.
   5.1 Right-click in the Statistics pane and select ***Add New Reference Point***.
   5.2 Enter a suitable name (e.g., ***Proc_6_cycle***) when prompted and click on ***OK***.

6. Click on the ***Go*** button.
   6.1 When the breakpoint at ***Proc_6*** is reached again, the contents of the reference point are updated to reflect the number of instructions and cycles consumed for one iteration of the loop.
   6.2 The result shown in Console window also reveals some information about running the benchmark program

4.2 Estimate the execution time

1. Clear all breakpoints

2. ***Select Options*** ➔***Configure Target…***then click on the ***Configure*** button.

3. Mark the clock as ***Emulated*** and set ***Speed*** as ***10MHz***

4. Reload the executable image

5. Click on the ***Go*** button.

6. When prompted, request ***30,000*** runs through Dhrystone.

7. Check the result on ***Console window***.
   The information reveals the ***Microseconds for one run through Dhrystone*** (the smaller the better) and ***Dhrystones per Second*** (the larger the better). Record these values.

8. Check internal variable ***$sys_clock***, which records the number of ***centiseconds*** since the simulation started.
   8.1 To display this value, select ***System Views*** ➔***Debugger Internals*** ➔***Internal Variables***)
   8.2 You may change the format of ***$sys_clock*** to decimal. Record this value, said sys_time.

9. Alternatively, the ***execution time = Cycle count / Cycle Frequency***. As we set the bus frequency to 10MHz, we can calculate the total execution time = (total cycle count/ (10×10^6)) in ***second***.
   9.1 Record the total cycle count shown in the ***Statistics tab*** and calculate its time, said ***cyc_time***, by above equation. Then (***sys_time/100***) should approximate to ***cyc_time***.

10. Reload the executable image, repeat the steps ***5~7*** except that request ***40,000*** runs through Dhrystone. The results shown on ***Console window*** should be the same that in step 6.

11. Reload the executable image, repeat the steps *2~7* except that set the ***Emulated Speed*** as *20MHz*.
    11.1 What message is shown on ***Console window***?

12. Repeat the actions at step 11 except that request *60,000* runs through Dhrystone.
    12.1 What is the difference between this result and the result at step 7?

Note:
1. If the system clock is set to Real-time, then *$sys_clock* will return actual time using the host computer's real-time clock rather than simulated execution time. This will benchmark the performance of the host computer!
2. Note that entering a speed without specifying units assumes for example 50 assumes 50Hz rather than 50MHz. Speeds given in kHz and GHz are also acceptable.

4.3 Performance estimation using different Memory models

The default setting for the ARMulator is to model a system with 4GB of zero wait state 32bit memory. However, real systems are unlikely to have such an ideal memory system! Hence an alternative memory model called mapfile can be used. The mapfile memory model reads a memory description file called a map file which describes the type and speed of memory in a simulated system.

Note: ARMulator accepts a map file of any name. The file must have the extension ***.map*** or ***.txt*** for the browse facility to recognize it; however, any extension may be used if you are entering the path and filename explicitly in the map file text entry field.

To calculate the number of wait states for each possible type of memory access, the ARMulator uses the ***values supplied in the map file*** and the ***clock frequency specified to the model***.

For cached cores, the clock frequency specified is the ***core clock frequency***. The ***bus clock frequency*** is calculated by dividing the specified core clock frequency by the ARMulator constant - ***MCCFG***. The derived bus clock frequency is used to calculate ***wait states*** in cached cores.

Note: ARMulator constant - ***MCCFG*** is specified in *install_directory\Bin*.ami*. ***Default.ami*** specifies the processor to use if no other processor is specified. The default setting in ***default.ami*** is ***MCCFG=3***. See ***ARM® Developer Suite Debug Target Guide*** for more information.

In the following steps, we will use armsd.map located at C:\Program Files\ARM\ADSv1_1\Examples\dhry\ as the map file. This map file describes a system

00000000 80000000 RAM 4 RW 135/85 135/85

- A section of memory starting at address 0x0
- 0x80000000 bytes in length
- labeled as RAM
- a 32-bit (4-byte) bus
- read and write access

- read access times of 135ns nonsequential and 85ns sequential
- write access times of 135ns nonsequential and 85ns sequential

1. Clear all breakpoints

2. ***Select Options*** ➔ ***Configure Target…*** then click on the ***Configure*** button.
   2.1 Mark the clock as ***Emulated*** and set ***Speed*** as ***10MHz***
   2.2 Specify the memory file through browsing to C:\Program Files\ARM\ADSv1_1\Examples\dhry\armsd.map

4. Reload the executable image

5. Click on the ***Go*** button.

6. When prompted, request ***30,000*** runs through Dhrystone.

7. Check the result on ***Console window***.
   The information reveals the ***Microseconds for one run through Dhrystone*** (the smaller the better) and ***Dhrystones per Second*** (the larger the better). Record these values and compare with those values you recorded at step 7 in Section4.2 Estimate the execution time.

8. Check internal variable ***$sys_clock*** and compared with that you got at step 8 in Section 4.2 Estimate the execution time. Remember the data format should be the same, i.e., decimal. The performance should be worse.

9. Read the memory statistics
   9.1 Open ***Command Line Interface Window***
   9.2 Enter command ***di*** (short form of ***dbginternal***), and press any key (e.g., enter) until ***$memstates*** are displayed. In this case, only single memory is used and therefore ***$memstates[0]*** is displayed.

4.4 Benchmarking cached cores

1. Edit ***default.ami*** located at ***C:\Program Files\ARM\ADSv1_1\Bin***
   1.1 If MCCDG≠3, set ***MCCFG=3*** and quit AXD and launch it again.

2. ***Select Options*** ➔ ***Configure Target…*** then click on the ***Configure*** button.
   2.1 Select Processor variant as ***ARM940T***
   2.2 Mark the clock as ***Emulated*** and set ***Speed*** as ***10MHz***
   2.3 Specify the memory file through browsing to ***C:\Program Files\ARM\ADSv1_1\Examples\dhry\armsd.map***

3. After the step 2, check the message shown in ARMulator startup banner: ***System Output Monitor - RDI Log***. An example result is displayed in Figure 4.

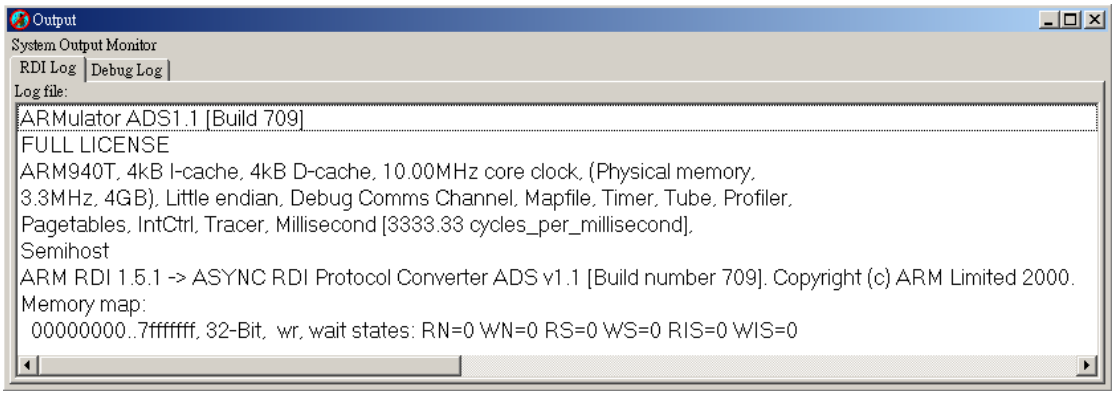

Figure 4. ARMulator startup Message.

4. Load or reload C:\ARMSoC\Lab_03\My_Poeject_Data\DebugRel\My_Poeject.axf.

5. Click on the ***Go*** button.

6. When prompted, request ***40,000*** runs through Dhrystone.

7. Open ***Command Line Interface Window*** and enter ***print $statistics***. An example result is displayed in Figure 4.

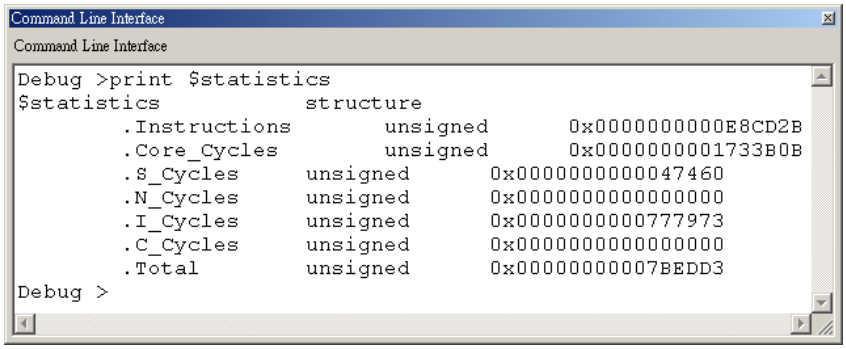

Figure 5. Brief statistics.

8. Edit ***default.ami***
   8.1 For ADS 1.2, set Counters=True after the line setting MCCFG=3;
   for ADS 1.1, add Counters=True after the line setting MCCFG.

   Choose 8.2a or 8.2b step:
   8.2a ***Select Options➔Configure Target…***then click on the ***Configure*** button. Select ***OK***
   8.2b Quit AXD and then restart it.

9. Open ***Command Line Interface Window*** and enter ***print $statistics***. Additional statistics for cached core is displayed, as one example displayed in Figure 6. Because we do not start the execution of the program, all values are zero.

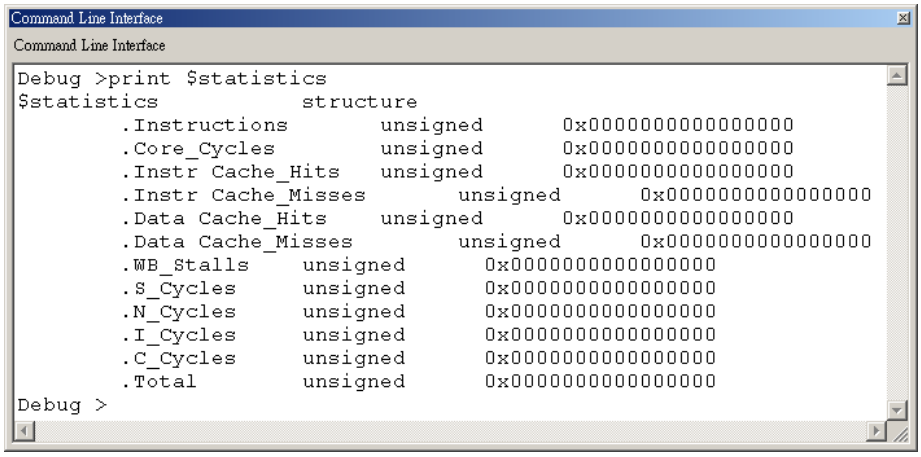

Figure 6. Cached core additional statistics.

10. Click on the ***Go*** button.
    10.1 Set a breakpoint on the line ***158*** of the source file, the ***Proc_5()***.
    10.2 Select ***System Views*** ➔***Debugger Internals***➔***Statisticss***.

11. Click on the ***Go*** button.
    11.1 When prompted, check the values at ***Statistics tab***.
    11.2 Right Click on ***Statistics tab*** and select ***Add New Reference Point***. Enter ***iter_1*** in the pop-up window. The new reference point will appear with zero values.
    11.2 Request ***40,000*** runs through Dhrystone in the ***Console Window***.

12. When the debugger halts at the breakpoint, check the values of ***iter_1*** and record ***Total cycle*** count.
    12.1 Add another new reference point, named as ***iter_2***.
    12.2 Resume the program.

13. When the debugger halts at the breakpoint, check the values of ***iter_2*** and record ***Total cycle*** count.
    12.1 Add another new reference point, named as ***iter_3***.
    13.2 Resume the program.

14. When the debugger halts at the breakpoint, check the values of ***iter_3*** and record ***Total cycle*** count.
    14.1 Clear or disable the breakpoint on the line ***158*** and resume the program.

The change of the total cycle of iter_1, iter_2 and iter_3 could be 10307➔ 901➔ 277. For the first iteration of the loop, the loop instructions and data would not be held in the cache memory, hence there are many cache misses and the total cycle is large. After several iterations, the Dhrystone loop will be held in cache memory and therefore the total cycle for each iteration is reduced.

5. Efficient C programming

1. Edit a copy of loop.c shown in Figure 7. Build it and record its memory requirement.

```
 1  #include <stdio.h>
 2
 3  int acc(int n) {
 4     int i; //loop index
 5     int sum=0;
 6
 7     for (i=1; i<=n ;i++)
 8        sum+=i;
 9     return sum;
10
11  }
12
13
14  int main(void) {
15     int acc_val;
16
17     acc_val=acc(10);
18     printf("%d\n",acc_val);
19
20     return 0;
21
22  }
```

Figure 7. loop.c

2. Load the executable image of ***loop.c*** into AXD.
   - 2.1 Open ***Processor Register View*** and extend the ***Current Register***.
   - 2.2 ***Select Options*** ➔ ***Configure Target…***then click on the ***Configure*** button.
   - 2.3 Select Processor variant as ***ARM7TDMI***
   - 2.4 Mark the ***Clock*** as ***Real-time***
   - 2.5 Specify the ***Memory Map File*** as ***No Map File***

3. Click on the ***Go*** button, Stepping Mode in Strong Source
   - 3.1 Right click on ***Source window*** and set ***Stepping Mode*** in ***Strong Source***.
   - 3.2 ***Step in*** through the program, check how the argument is passed to ***acc()***.
   - 3.3 During the execution of ***acc()***, which registers are used?
   - 3.4 Check which register is used to pass result to ***main()***.

4. Click ***Go*** button to finish the rest of the program.

5 Reload the imgae.
   - 5.1 Set two breakpoints, one on ***for (i=0; i<=n ;i++)*** and the other on ***return sum;***
   - 5.2 Click ***Go*** button. When the program halt at breakpoint on ***for (i=0; i<=n ;i++)***, add a new reference point, named as ***loop_time***.
   - 5.3 Click ***Go*** button again. When the program halt at breakpoint on ***return sum***, record the ***Total*** cycle count of ***loop_time***.
   - 5.4 Click ***Go*** button to finish the rest of the program. Record the ***Total*** cycle count of ***&statistics***.

6. Copy ***loop.c*** to a new file named as ***loop_opt.c***
   - 6.1 Change the statement ***for (i=1; i<=n ;i++)*** to ***for (i=n; i!=0 ;i--)***

6.2 Build it and compare its memory requirement with ***loop.c***

7. Load the executable image of ***loop_opt.c*** into AXD.
   7.1 Repeat step 3 and 4. Compare the results with those of ***loop.c***.
   7.2 Repeat step 5. Compare the result with that of ***loop.c***.