Codewarrior for ColdFire (Eclipse) 10.1 Setup

1. Goal

This document is designed to ensure that your CodeWarrior for Coldfire v10.1 environment is correctly setup and to orient you to it basic functionality of programming and debugging C and assembly based programs.

2. Introduction

The Codewarrior for Coldfire v10.1 is an integrated development environment (IDE) for the Freescale Coldfire line of microcontrollers. This microcontroller uses a core processor that implements the Coldfire instruction set. In addition, the Coldfire MCF52259 processor has 512KB of Flash memory for non-volatile program storage, a small amount of RAM (64KB), integrated interrupt controller, timers, UART (serial) ports, USB and Ethernet ports, several channels of A-to-D converters, and many other peripheral devices including the ability to use several dozen I/O pins as general purpose I/O. You will learn more about these features throughout the semester. Information regarding the control of 52259 is available online through its reference manual. The reference manual is broken into chapters for each I/O block and provides bit-by-bit explanations for relevant control registers. To configure and test embedded software, the MCF52259 processor has been integrated onto the 52259 board. The board includes some on-board I/O such as a few pushbuttons, a potentiometer, and LED's. It also provides a power supply and regulator, a debugging control chip (BDM chip), and a place for a 40-pin I/O connector.

To develop software for the Coldfire microcontroller, Freescale provides an Eclipse-based IDE (Integrated Development Environment), named Codewarrior. This IDE provides a 52259 C compiler and assembler along with integrated debugging tools to allow you to run and debug your application code by single-stepping, setting breakpoints, and viewing register and memory contents.

3. Codewarrior Setup and Installation

a. The Codewarrior IDE has been installed on the computers in the RTH classrooms. However, it is not currently installed in any of the computer user rooms on campus. Instead, you can install Codewarrior on your own PC/laptop by going to the following link: http://cache.freescale.com/lgfiles/devsuites/HC08/CW_MCU_v10.1_B110726_SE.exe?fpsp=1. We highly encourage you to install this software on your own PC so you can work on your lab outside of discussion times. You should be able to use the standard install options without issue. At the end of installation, it will ask to install several hardware drivers to connect to your board. Say yes to all of these driver install packages. Though it is Windows-based we have had previous versions running via Parallels and/or Bootcamp on Mac's. There is also a linux

version available. The software should work on either 32-bit or 64-bit OS version but if you have any issues, please consult the TA and discussion boards.

4. Creating Projects in Codewarrior

Usually you will be provided with an incomplete (skeleton) set of source files. You will then create a project through Codewarrior and then copy and add those source files. Details for creating a new project are described below.

a. Start Codewarrior and Select "File > New > Project". The New Project dialog box appears. Select "Bareboard Project" and click Next. The Create an MCU Bareboard Project page appears. Type your desired project name and in location set the parent directory where you would like your project folder.

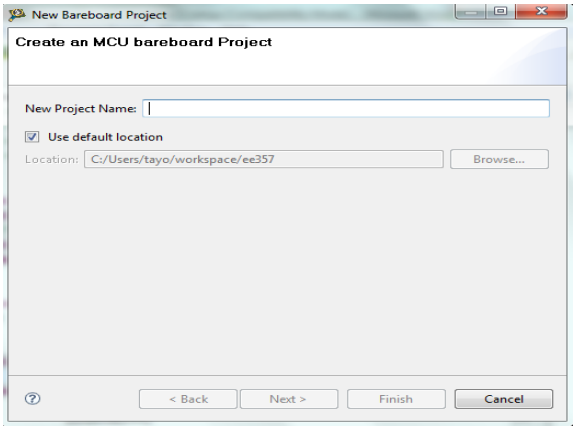


Figure 1 - New MCU Project Page

b. Next, we will tell Codewarrior what part we are using. In the window pane find 'ColdFire Vx Tower Boards'… 'ColdFire V2'… 'TWR-MCF5225X'. Click Next. The connections page appears. Check the "P&E Universal/USB MultiLink" box. Click Next.

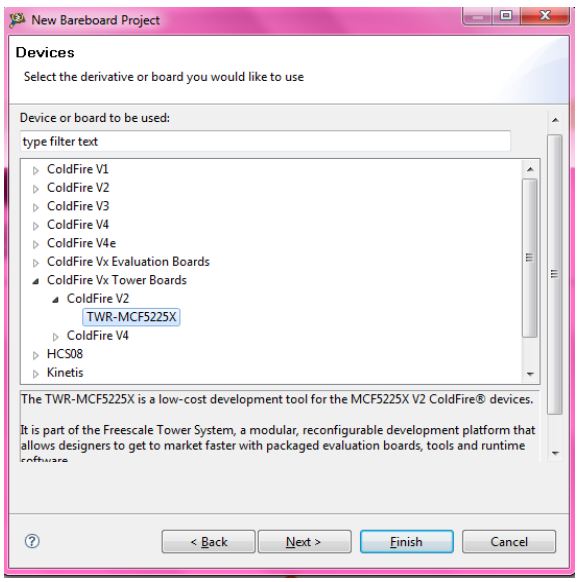


Figure 2 - Setting the Part

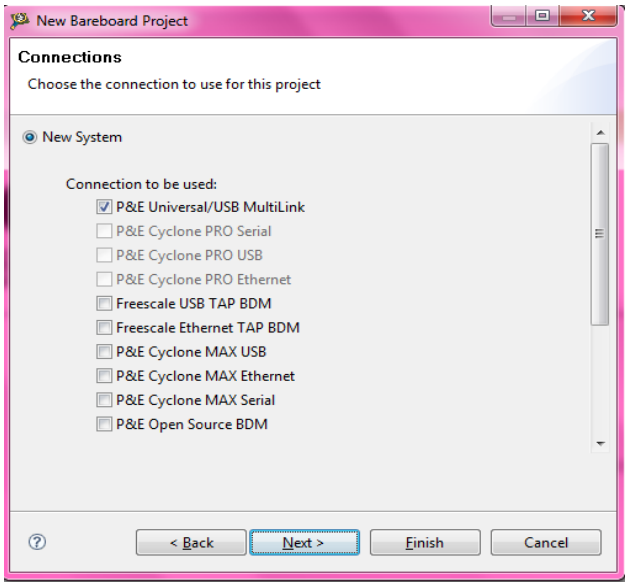


Figure 3 - Setting the Connection Type

c. Skip through the next window (we will add source files later).
d. In the next window select “Minimal Hardware Support” for Hardware Startup. Select “Easy Debug” for the Optimization Level. Click Next.

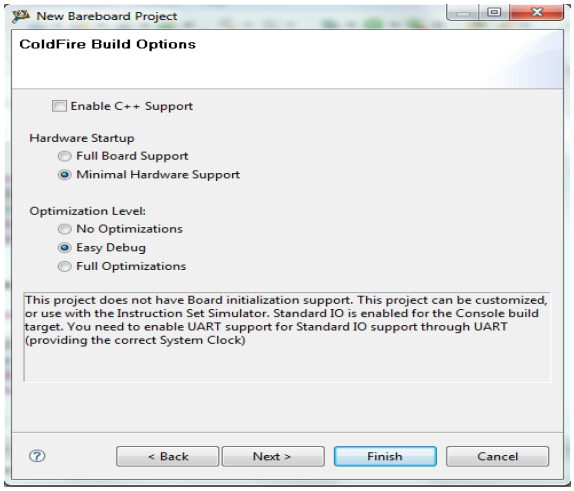


Figure 4 - Startup code generation option

e. In the next window regarding the “Processor Expert”, click “None” and then Click Finish.
f. The CodeWarrior view should look similar to the screenshot below. If it doesn’t, Go to the menu “Window -> New Window”.

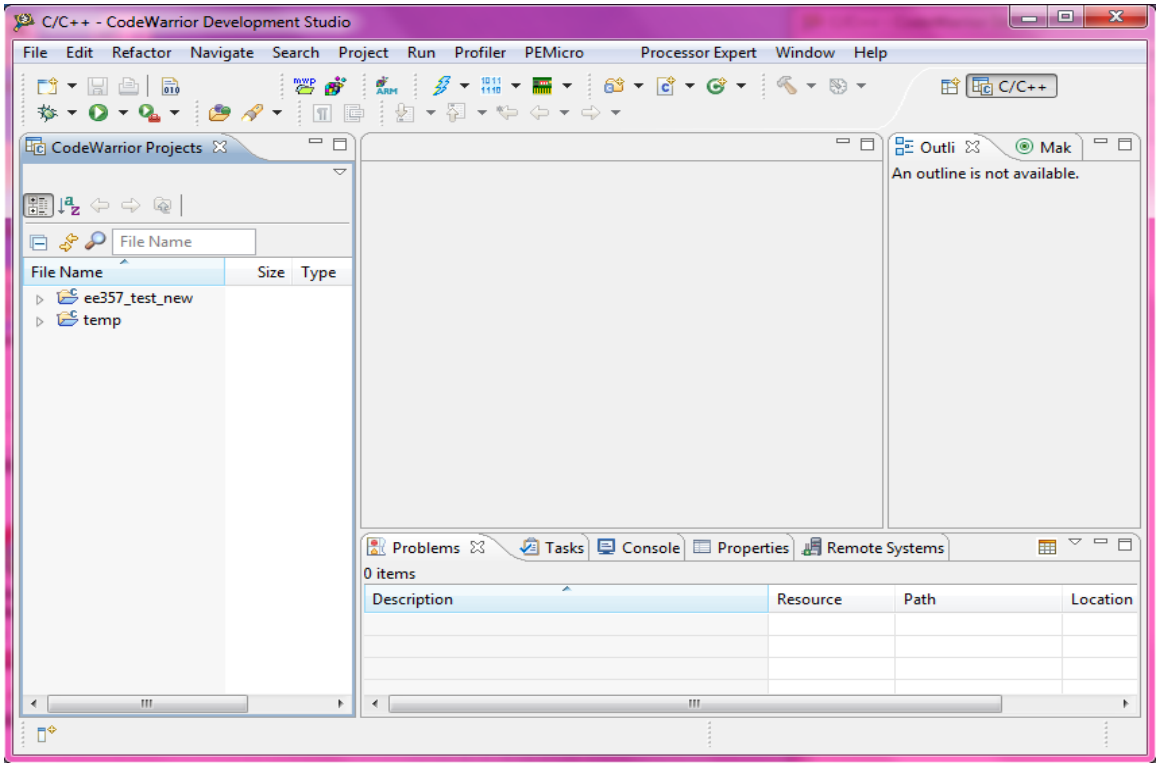


Figure 5 - CodeWarrior Development Window

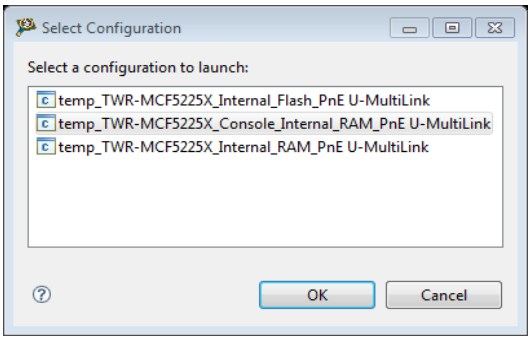
5. IMPORTANT: Updating ColdFire Board Firmware

Read sections 6 and 7 of this document to learn more about compiling and debugging your projects.

a. Create a new project as done in section 4 above.
b. Connect your ColdFire board.
c. Click on the debug icon:

Debug Icon

d. Select the option that has "…Console_Internal_RAM_PnE..." Click "OK". If the Firmware is out of date the following dialog box will appear. **DO NOT CLICK ON OK YET.**



e. Disconnect your ColdFire board. Connect a jumper on your ColdFire board to the 2-pin header pin close to the USB port that says BTLD. The TA will provide a 2-pin jumper to the students if needed.

Figure 6 - ColdFire Board BTLD 2-pin jumper

f. Reconnect the ColdFire board. A new driver will automatically be installed for the device. When this is done, click OK on the previous dialog box. Wait for the Firmware to complete installation.

g. **When installation is complete, disconnect your ColdFire board. Remove the jumper (please return it to the TA, if you collected it from your TA). Reconnect you board. Your board should be ready to use.**
h. **Get help from your TA, if you are not able to start debugging your projects in your ColdFire board.**

6. C and Assembly Projects

a. We have provided several files that include other helpful functionality in both C-based or assembly-based projects. On Blackboard..Assignments..Labs, download the "Codewarrior 52259 Assembly Template Files" .zip file or "Codewarrior 52259 C Template Files". Unzip it to a temporary location. **Follow the procedure below for EVERY new project you create in this course.**
b. **For C projects**: In Windows Explorer, take the source files you unzipped and copy the "usc_support.h" file to "*Project_name/*Project Headers" folder and the "usc_support.c" file to the "*Project_name/*Sources" folder. All files in the sources folder are automatically included in the project. Now from within the Eclipse project, open/edit 'main.c'. To use our USC specific functionality, please add the following line to 'main.c' just under the `#include <stdio.h>` line:

```
#include "usc_support.h"
```

You can begin editing 'main.c' and write your program. For I/O, we have written our own version of 'printf' and called it 'myprintf' while 'scanf' remains unchanged. You can use normal 'printf' and 'scanf' I/O functions but please reference the note below.

Note: There are known problems with Codewarrior's Console I/O that require every string and integer input and output to be terminated or followed by a newline (LF).

c. **For ASSEMBLY-ONLY projects**: In Windows Explorer, take the source files you unzipped, copy "usc_support.h" and "ee357_asm_lib_hdr.s" and "ee357_exceptions.h" to "*Project_name/*Project Headers". Then copy "usc_support.c", "ee357_asm_lib.s", "main.s", " to the "*Project_name/*Sources" folder. Next, from within the Eclipse project, find the 'main.c' file in the *Project_name/*sources folder. Right-click on the file. Select "Exclude from build". Click "Select all" then "Ok". Your main program is now ready to be edited in 'main.s'. From within your main.s code, you can perform I/O from the HW board to your PC. In assembly this can be followed by calling the appropriate subroutines via the `jsr` instruction and pre-loading certain register with the necessary arguments. See the table below and reference the note above in the C section which also applies to assembly projects.

I/O Operation	Calling Instruction	Arguments	Return Value
Print a null-terminated ASCII string	`jsr ee357_put_str`	A1 = Pointer to first character of string	None
Print an integer	`jsr ee357_put_int`	D1 = Value to print	None
Get the specified number of	`jsr ee357_get_str`	A1 = Pointer to	D1 = Number of

ASCII characters from the PC keyboard. Must terminate with the 'Enter' key (ASCII 'LF' char.)		memory buffer to place string D1 = Max characters to receive	characters received (including the 'LF'). Unused characters in the buffer will be zeroed (NULL).[1]
Get an integer from the PC keyboard. Must terminate with the 'Enter' key.	`jsr ee357_get_int`	None	Integer from keyboard

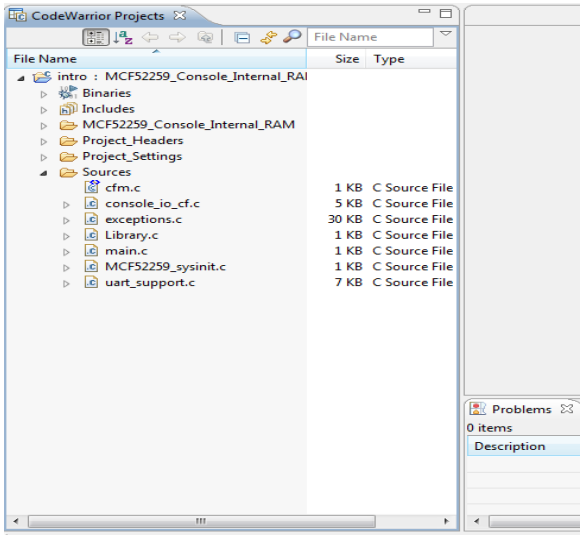


Figure 7 - The codewarrior project navigation pane shows all the source files of the project.

[1] When getting a string (ASCII characters), make sure the value passed in D1 is 1 less than the size of the buffer so that a NULL character can be added (which is necessary if this will be printed or processed as a traditional string). If more characters are entered than the value passed in D1, characters beyond the limit will be discarded (including the newline/LF character)

7. Compiling and Debugging Projects

a. A few buttons will be useful when compiling and debugging your code.

	
Build (Compile) Icon	Debug Icon

b. First, you must select the target I/O system and code location. This can be done by right-clicking on the project name in the project navigation pane (shown above). Select "Build Configurations" > "Set Active" > "MCF52259_CONSOLE_INTERNAL_RAM".

You should normally ensure this is set to "CONSOLE_INTERNAL_RAM." Information about each option is listed below:

i) "CONSOLE_INTERNAL_RAM": Unless otherwise directed be sure that this option is selected. This allows you to execute I/O functions (either the assembly versions or C I/O functions like `myprintf()` and have the output be redirected to the Codewarrior Console window.

ii) "INTERNAL_RAM": This option does not include the standard C I/O libraries and places code in system RAM.

iii) "INTERNAL FLASH": This option should **NEVER** be selected. (Because we are only debugging our code, we will always want it loaded into RAM. In the real world, once you have debugged your code and want to embed it in the 5211 processor, the final code can be permanently burned to ROM.)

c. Select your project in the project navigation pane. Then click the "build" button.

d. If you are running an assembly-based project, it is highly recommended that you set a breakpoint at the "main" label / first instruction by clicking in the left margin of the line. A red dot will appear indicating the breakpoint. If you do not set the breakpoint, the program will simply run and not allow you to step through your code. C-based projects will automatically load the program and stop at the "main()" function and allow you to run the program or step through it.

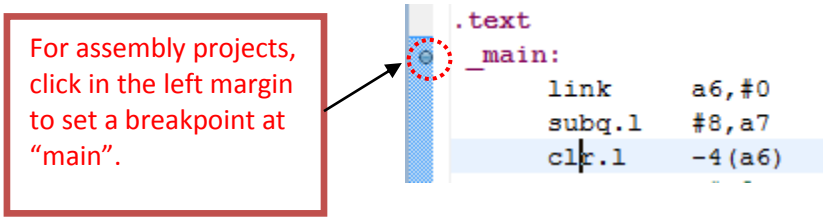


Figure 8 - Set a breakpoint in assembly projects to prevent run-away execution or your code when the debugger is started

e. You will be running the code on your board, first connect the USB cable from the PC to the M52259 board. (Note: Be aware that the first time you connect your board, the PC may need to install a driver.) The PC should recognize the USB

device at which point you can click the “Debug” button. Select the configuration “<projectname>_Console_Internal_RAM…” to download and run your code to the board.

f. The “Debug” perspective will appear and execution should break at the breakpoint in your assembly code or, for C projects, the first line of your “`main`” routine. Note you can switch between the Debug perspective and the default perspective(C/C++) by selecting the desired one in the upper right-hand corner.

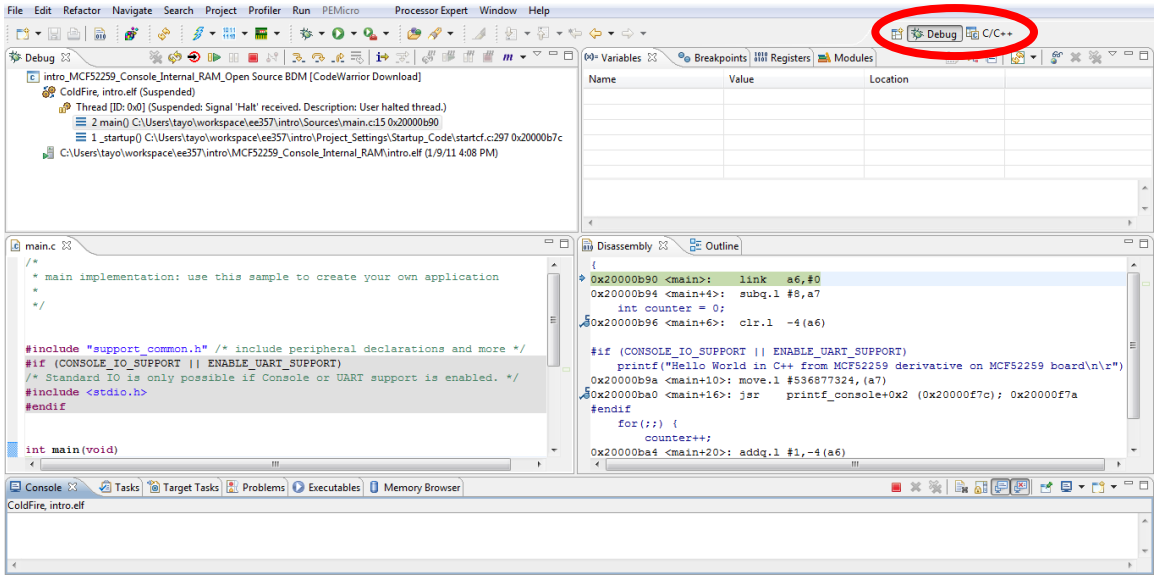


Figure 9 - The Debug Perspective

g. The debug perspective by default shows you the assembly code that is generated by the C compiler. As shown in figure 7.

h. The available commands are accessed using the toolbar at the top of the debug window. The commands are summarized below.

	Run/Resume	Runs the program until it is stopped or encounters a breakpoint.
	Suspend	Breaks/stops program execution.
	Kill / Terminate	Kills the execution and stops debug mode
	Step Over	Steps through code and over function calls (executes the entire function moving on to the next statement.)
	Step Into	Steps through code and into function calls.
	Step Out	Completes execution of the current function and back to the calling routine.

Figure 10 - Debug Toolbar Commands

i. **Breakpoints**: Breakpoints can be set by double-clicking on the far left of each statement. When a breakpoint is enabled, a check mark appears indicating the

execution will stop at that statement. Double-clicking on the check mark will remove the breakpoint and the check mark will disappear.

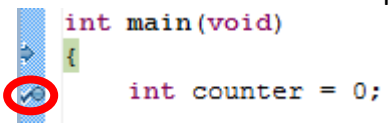


Figure 11 - Setting Breakpoints

j. **Watching Variables (C programs only):** If you set a breakpoint or step through your code, you can see the value of live variables in the upper right window pane of the debug perspective.

k. **Viewing Registers**: Register values including most I/O control registers can be viewed by selecting "Registers" in the upper right window pane. The register window will appear and can be used to navigate to the desired processor and I/O register. When using the actual boards, all I/O registers for controlling integrated peripherals will also be visible.

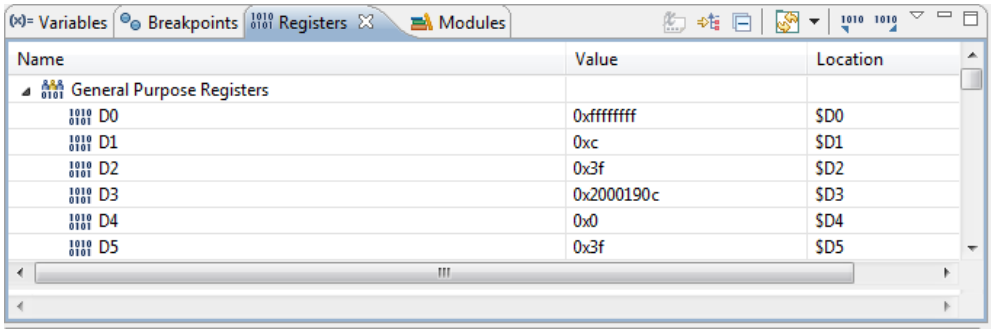


Figure 12 - Register Window

l. **Viewing Memory**: Memory can be viewed by selecting "Memory Browser" in the bottom window pane. You can navigate to the desired address range by typing the address in the Display textbox. Note that addresses should be typed in C hex format (i.e. prepended with 0x40000000). Alternatively, you can right click on address registers (An) in the register window and view the memory contents starting at the address to which they are pointing. Also, the .data section of memory for the 52259 starts at 0x20003004.

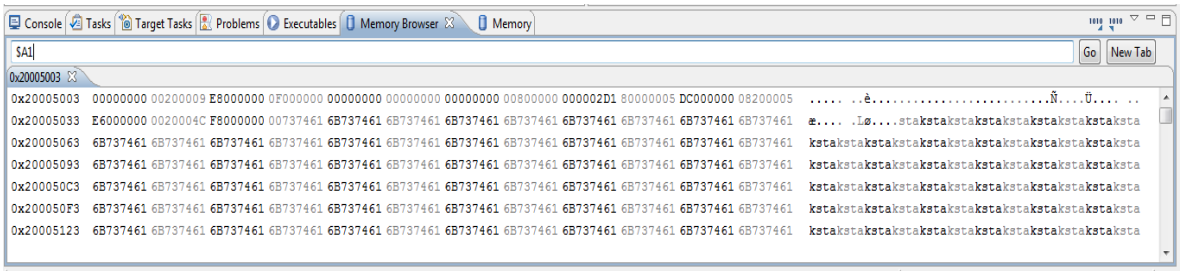


Figure 13 - Memory Window