

## Debug/Simulate an ASM Project in CCS without DSP Board

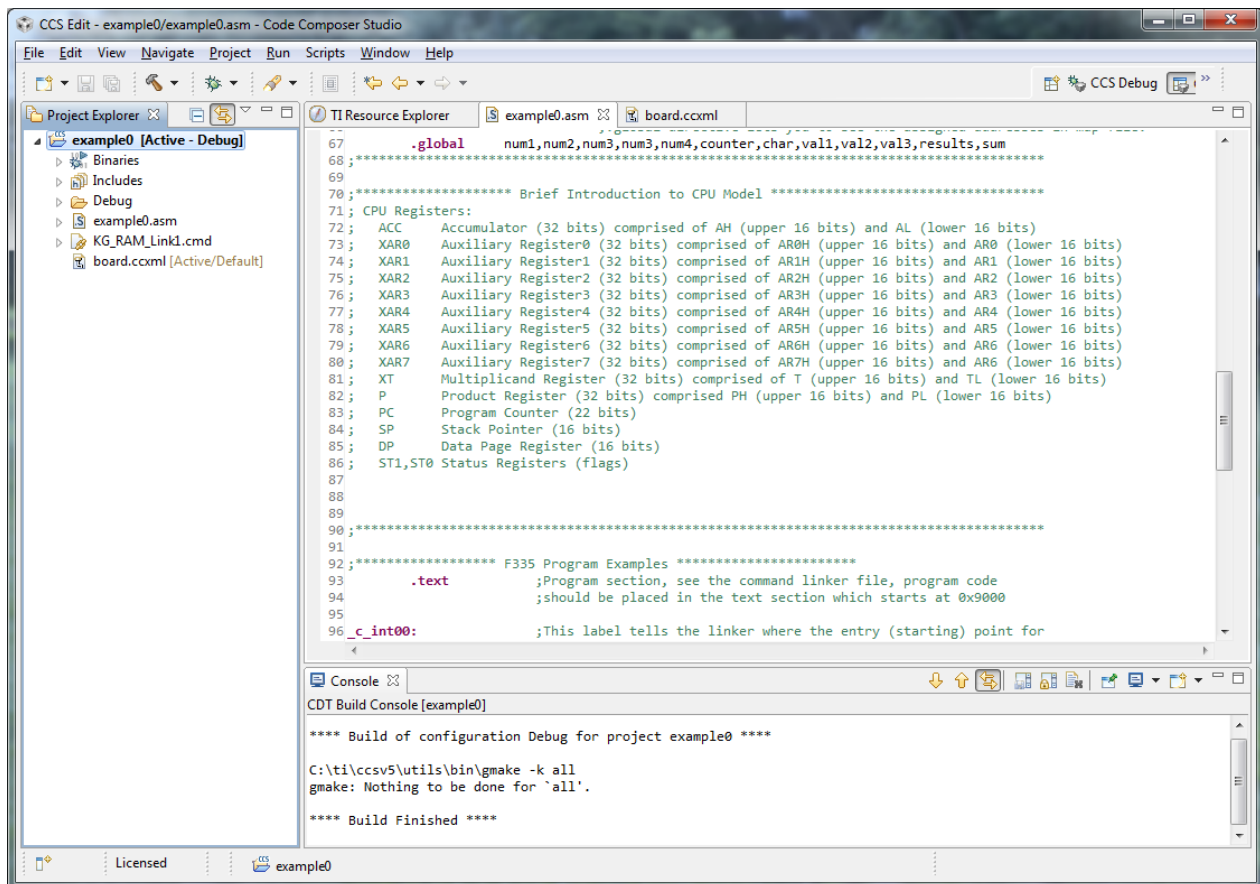
### Introduction

The purpose of this document is to quickly illustrate how to debug an assembly language file using the built in Code Composer Studio (CCS) debugger.

This tutorial will pick up where we left off in the last tutorial. Therefore you must have ex0.asm successfully assembled with no errors or warnings. This tutorial will illustrate the basic features of the CCS debugger and how to switch between simulated debugging and debugging on your actual lab board.

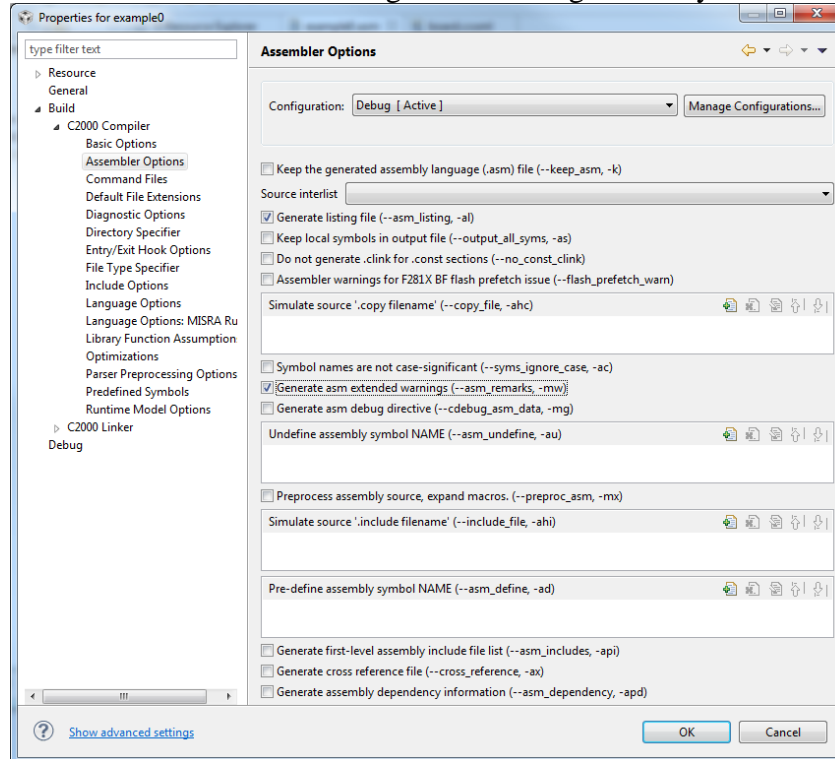
### Procedure

1. This tutorial assumes that you already have CCS installed and have set your workspace folder location on your hard disk. For more information on this topic see the tutorial [CCS\\_Installation\\_Instructions.pdf](http://mil.ufl.edu/4744/docs/CCS_Installation_Instructions.pdf) (at [http://mil.ufl.edu/4744/docs/CCS\\_Installation\\_Instructions.pdf](http://mil.ufl.edu/4744/docs/CCS_Installation_Instructions.pdf)). This tutorial also assumes that you have completed the tutorial on creating an ASM project, where we created a project **example0** which contained the files: **ex0.asm** and **KG\_RAM\_Link1.cmd**. For more information on this topic, see the tutorial [Create\\_ASM\\_CCS\\_Project.pdf](http://mil.ufl.edu/4744/docs/Create_ASM_CCS_Project.pdf) (at [http://mil.ufl.edu/4744/docs/Create\\_ASM\\_CCS\\_Project.pdf](http://mil.ufl.edu/4744/docs/Create_ASM_CCS_Project.pdf)).
2. Open up the example0 project. Select the Hammer icon (the rebuild the active project) on the toolbar to assemble and link the project. Do this to insure that there are no errors or warnings as shown below.



## Debug/Simulate an ASM Project in CCS without DSP Board

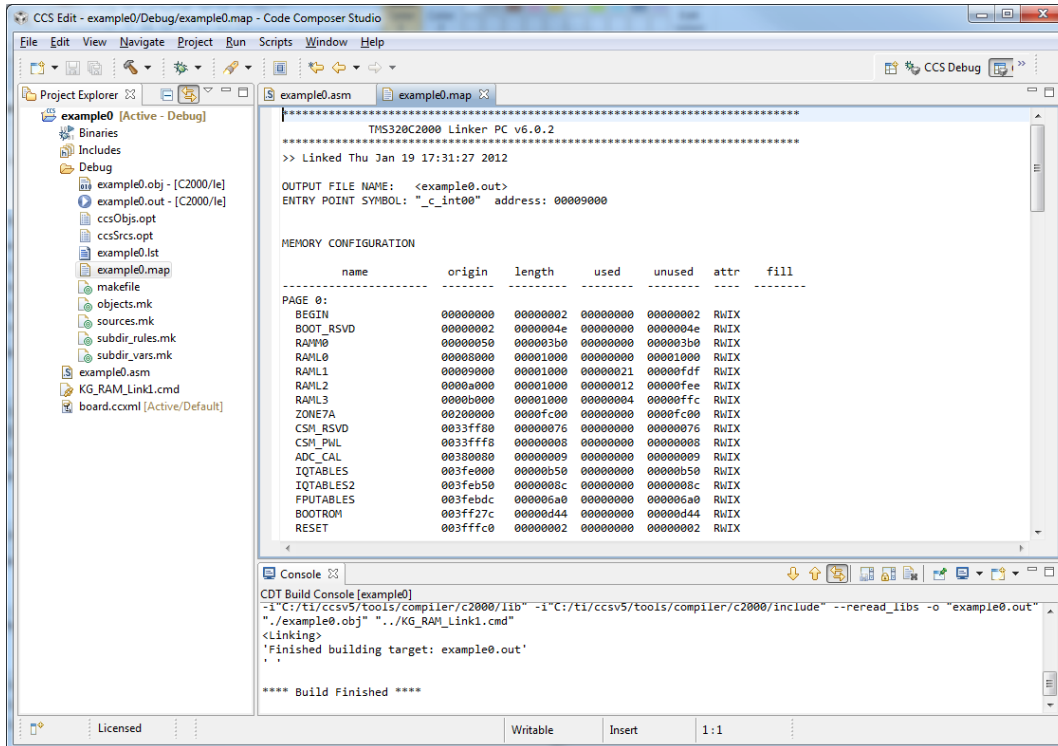
- Before we begin debugging the assembly code, we will first need to modify some of the project settings to produce map and list files. To do this, right click on the **example0 [Active – debug]** project icon that is located in the Project Explorer window. Scroll all the way down and select **Properties**. A new window should open showing all the project property tabs for project example0.
- Select **Assembler Options** under **C2000 Compiler**. Select **Generate listing file** and **Generate asm extended warnings** as shown in the next snapshot. This will tell the assembler to create a list file and also to give more information if a warning occurs during assembly or link time.



Select **OK** on the bottom right to effect the changes and exit the properties window.

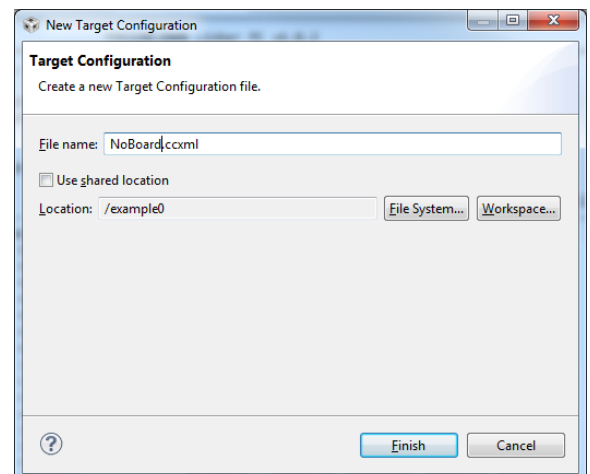
- Again select the **Rebuild Active Project** icon to assemble/link the code. Look in the Project Explorer window; open the map file by selecting the **arrow** to the left of **Debug** and then selecting **example0.map**. I have scrolled down a little in the map file for the snapshot below.

## Debug/Simulate an ASM Project in CCS without DSP Board



Repeat for the list file, **example0.lst**. The **map file**, **example0.map**, shows the physical addresses for each section of code placed in DSP internal SRAM: **.data**, **.text** and **.bss**. The map file also shows all the addresses of the variables that were set as global variables in the **example0.asm** file. The **list file**, **example0.lst**, identifies all the data and program (DSP instructions) placed in each individual section. Data variables are usually placed in the **.data** section while program is placed in the **.text** section. Run-time results are then placed in the **.bss** section. Each of these sections' physical address is determined by the information in the **command linker file**, **KG\_RAM\_Link1.cmd**. We are now almost ready to begin debugging (stepping through) code.

- There are two types of debugging modes that we can use: **real hardware** or **simulated**. Which one is used is dependent on the active target configuration file. In the Create CCS Project tutorial, we created a target configuration file to debug on the hardware. In the case of hardware debugging, we attach our lab board to our laptop running CCS and download and step through our assembly code *on the board*. In the case of simulated debugging, we step through our assembly code using the simulated debugger feature found in CCS. Now we will now create another target configuration file so that software debugging will be possible.
- To debug using the **simulator**, we will have to tell CCS that our target is simulation and not our lab board. Recall that in the last tutorial we created a new Target Configuration file to use the XDS100 programmer/emulator and TMS320F28335 DSP. If you double click on **board.ccxml** on the left you will see

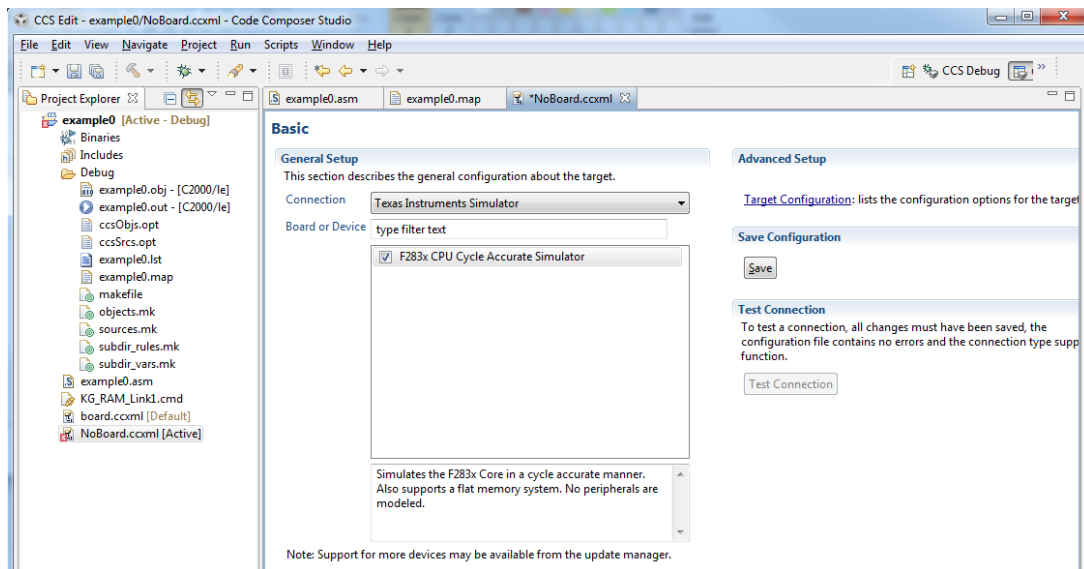


## Debug/Simulate an ASM Project in CCS without DSP Board

these settings. These are used to load code and debug using our lab board.

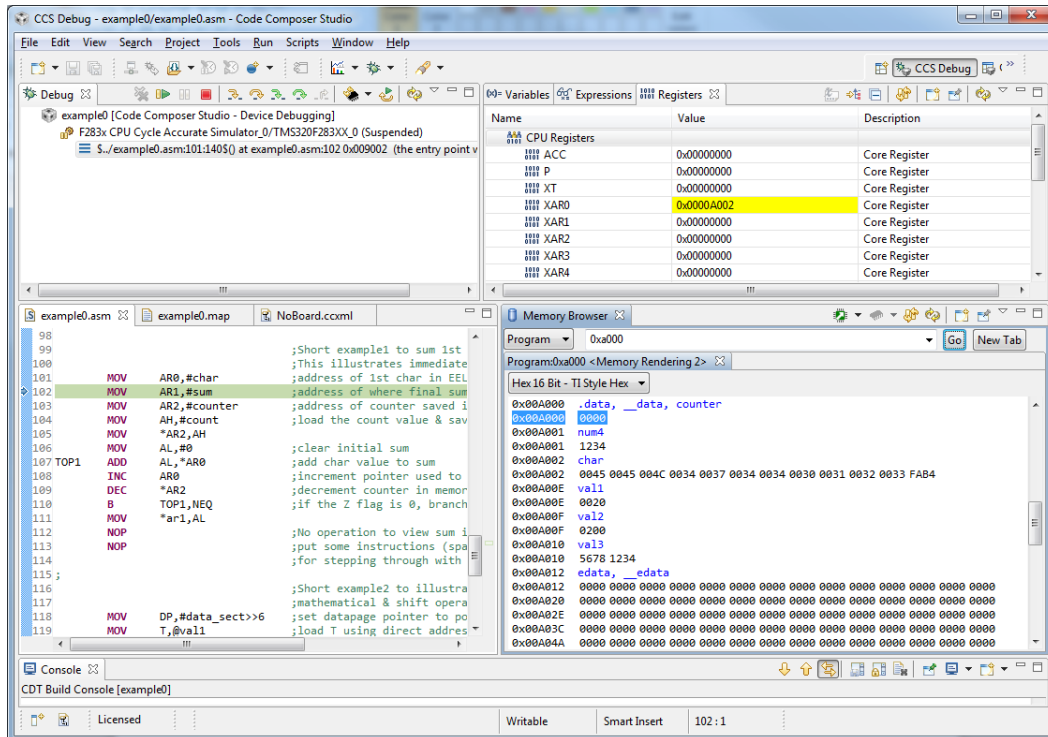
Now we will add a new Target Configuration File by pressing **File** → **New** → **Target Configuration**. Type in a filename **NoBoard** to remind us this target configuration is for simulation only. Verify that the NoBoard.ccxml file is placed in our example0 directory; if necessary, un-check the **Use default location** and select **/example0** as the destination directory. Now select **Finish**.

Set **Connection** to **Texas Instruments Simulator** and check the **Device** that says **F283x CPU Cycle Accurate Simulator** as shown in the snapshot below. Select **Save** and then you can close the file (by selecting the X to the right of the NoBoard.ccxml below the toolbar). NoBoard.ccxml should appear on the left in the Project Explorer window, at the bottom.



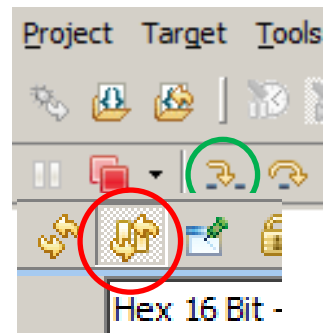
- Right-click the **NoBoard.ccxml** file and select **Set as Active Target Configuration** to tell CCS we are going to simulate debugging instead of using our actual DSP on our lab board. NoBoard.ccxml should now be the active target configuration shown on the left in the above snapshot.
- Finally we can start the **debugger** by pressing the small icon that looks like a six-legged insect shown to the right (or by selecting **Run** → **Debug** or with the F11 key). Do so now. A new window may pop up that says Variables | Expressions | Registers. If so, select **Registers**; if not, go to **View** to open a window for **Registers**. Go to **View** to open another window for **Memory Browser**. Expand the CPU registers icon to see the cpu registers. i.e., ACC, P, XT, ..., XAR7, PC, IC, etc. Type **0xa000** in the field on the **Memory Browser** window and set the pull-down next to this field to **program**. **Select the downward facing arrow in the Memory Browser window and select Default Rendering → Memory Rendering (not Traditional).** Then select **go**. You can see the variables that were placed into memory using the assembler directives in the data section of ex0.asm.



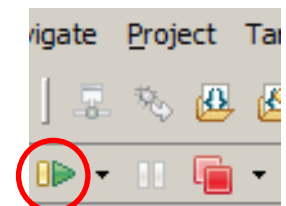


Scroll down to the **PC** (program counter) in the **Registers** window and verify that it is set to **0x9000**. This is where our code begins in ex0.asm. Begin stepping through the code by selecting the **step into** arrow (circled in green on the right in the **Debug** window) that is just to the right of the red **Terminate** square (used to exit debugging).

Scroll up to ACC, XAR0-XAR2 registers (in the **Registers** window) and observe how they change when the code is single stepped. Enter **0xb000** into the memory browser. You need to also select **Enable Continuous Refresh** (circled in red in the right snapshot) in the memory box (near the right edge) to see the memory changes when you step your code. Step through the first example that adds three ASCII characters. Notice that 0x00D6 (the sum of the three characters) is written into 0xb003 (which is the address associated with the sum label) after the **MOV \*AR1,AL** instruction is run.



10. Single step through the remaining code and again observe the results being written into memory locations 0xb000-0xb003. *This program will be covered in great detail during our class lectures so it is very important to attend class to learn about the F335 addressing modes and instruction set.*
11. To add a breakpoint, select the line number in the asm file. You can view and delete the break points by selecting **View → Breakpoints**. You can now run (the green arrow circled in red, as shown here) and it will stop at the breakpoint. You can either run again from there or start single stepping from this point forward.
12. To see the addresses for each of the instructions, **View → Disassembly**. This is very useful to confirm that everything is going where you want it to. To see the actual assembly language **including** the labels, select the icon in the disassembly window immediately to the right of the “Enter location here” box.
13. If you would like to run the program again, select the **Reset CPU** icon (shown to the right, circled in red) and then press the **Restart** icon



## Debug/Simulate an ASM Project in CCS without DSP Board

(immediately to the right that shows a green arrow with a yellow arrow underneath). Check to make sure the PC is starting again at 0x9000. A **Dissassembly** window may pop up which blocks the **Registers** window you opened earlier. You may close this but first notice that this shows the associated machine code for our program beginning at 0x9000. A **Console** window might also pop up; you can close this as well.

To terminate the debug session, press the red **Terminate** square. Note: Sometimes it takes a few seconds to exit debugging (due to CCS variables being removed from operation) so patiently wait until CCS returns to code assembly/link mode.

### FAQ Relating to ASM Debugging

1. When I re-run the debugger to execute my code, I noticed that the XAR0-XAR2 registers already had non-zero values in them. So I zeroed them out by clicking on them and then ex0.asm did not run properly. Why is this? What happened?

*When you re-run the debugger by pressing **Reset CPU** and **Restart**, several instructions are pre-fetched and executed due to the pipelining architecture of the CPU. Note: CPU pipelining will be further discussed in class. Instead you should reset & restart and simply leave the registers alone and step your code as before.*

2. When I step through my code, I noticed that it sometimes takes a couple more instructions to execute before results are actually written to memory. In other words, even though I stepped through an instruction that moves a register value to SRAM, I don't see memory change until a couple more instructions are stepped.

*Again I believe that this is due to the pipelining nature of the CPU. Most likely it was using the CPU address and data bus to pre-fetch new instructions and therefore waited several cycles later to perform the memory write cycle. This will effect when we view our results, however it is not a problem once we are running at full speed on our board.*

3. I am an engine-erd and I love this new tool. Someday I will create new products with this DSP so that others will have to work for me. Therefore I would like to know how to debug ex0.asm on my lab board so that I can begin experimenting with real hardware. How can I do this?

*Make sure the debugger is terminated by pressing the red **Terminate All** icon. Next plug in your lab board and right-click the **example0.ccxml** target configuration file and **Set as Active Target Configuration**. We are telling the CCS application that we now want to use our lab board as the target device for download/debug. Now press the debug (insect) icon as was performed earlier in simulated mode and step code as was done earlier in this tutorial.*