



Get Start with CCS and eZdsp USB Stick



# EXPERIMENT 1.1

# Propose of the experiment

- Use CCS to create a workspace
- Create a new project
- Write a simple C program using CCS tools
- Setup CCS build environment
- Connect eZdsp USB Stick Hardware to CCS
- Set target configuration
- Build and run a C program using CCS

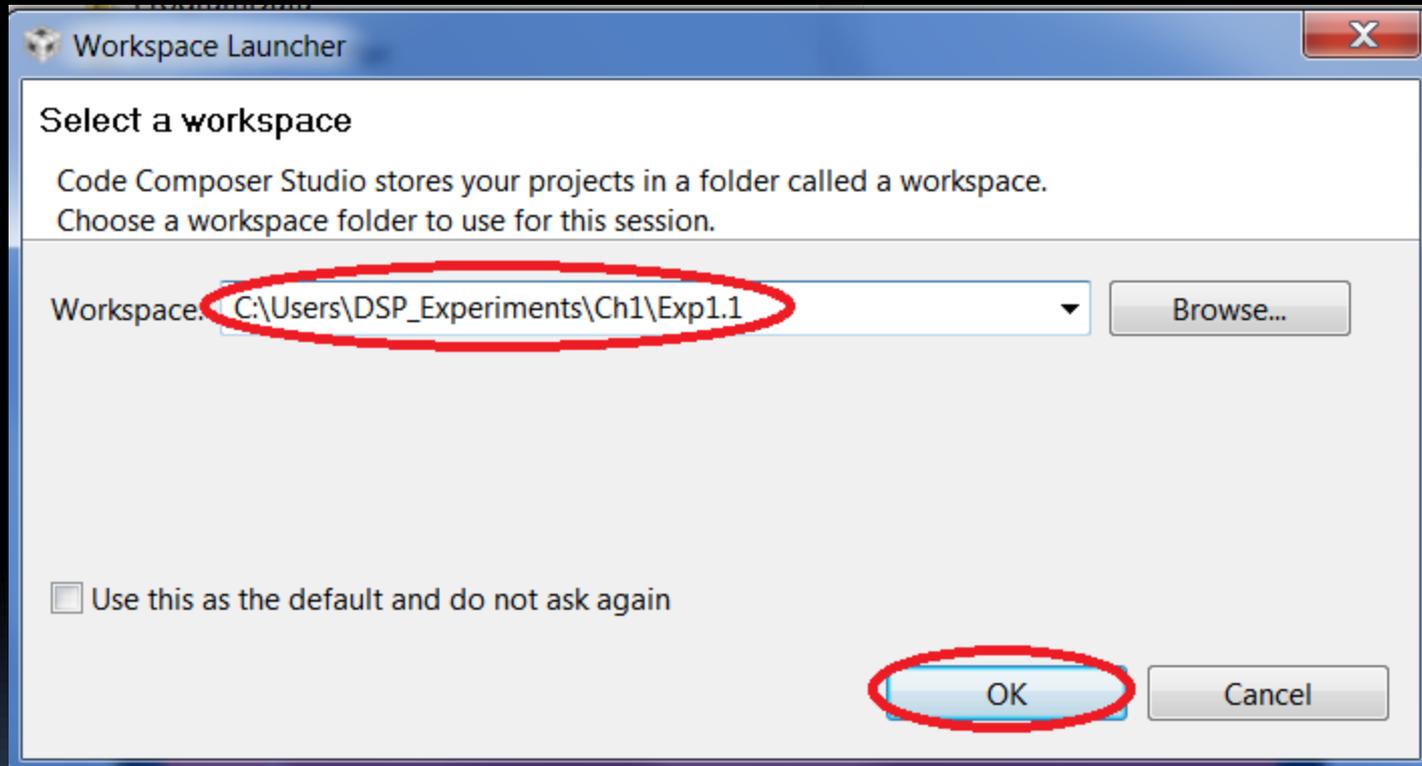
# Start CCS

*(Example: Code Composer Studio Version 5)*



# Create workspace

(Example: `C:\User\DSP_Experiment\Ch1\Exp1.1`)

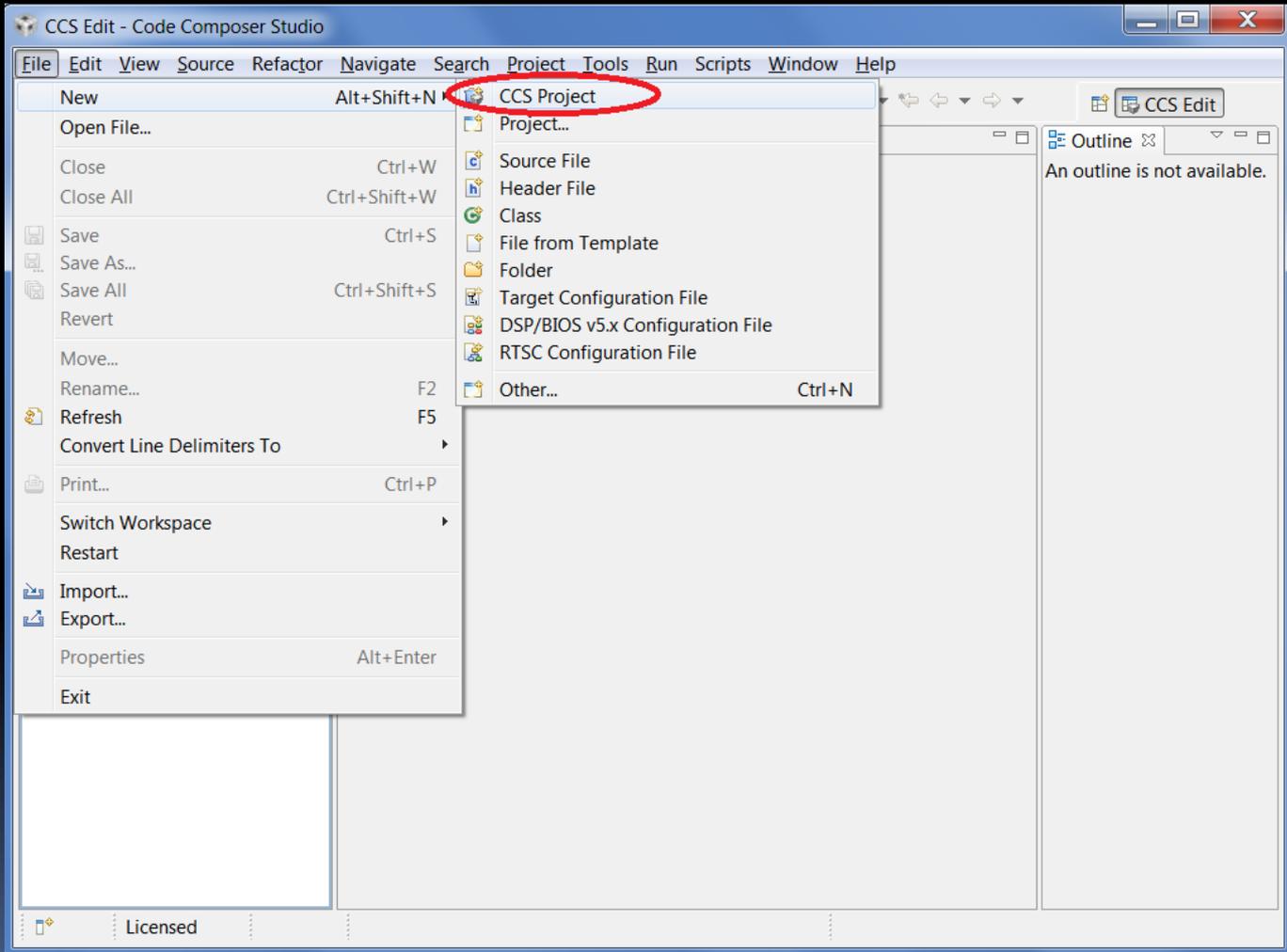


# Go to CCS



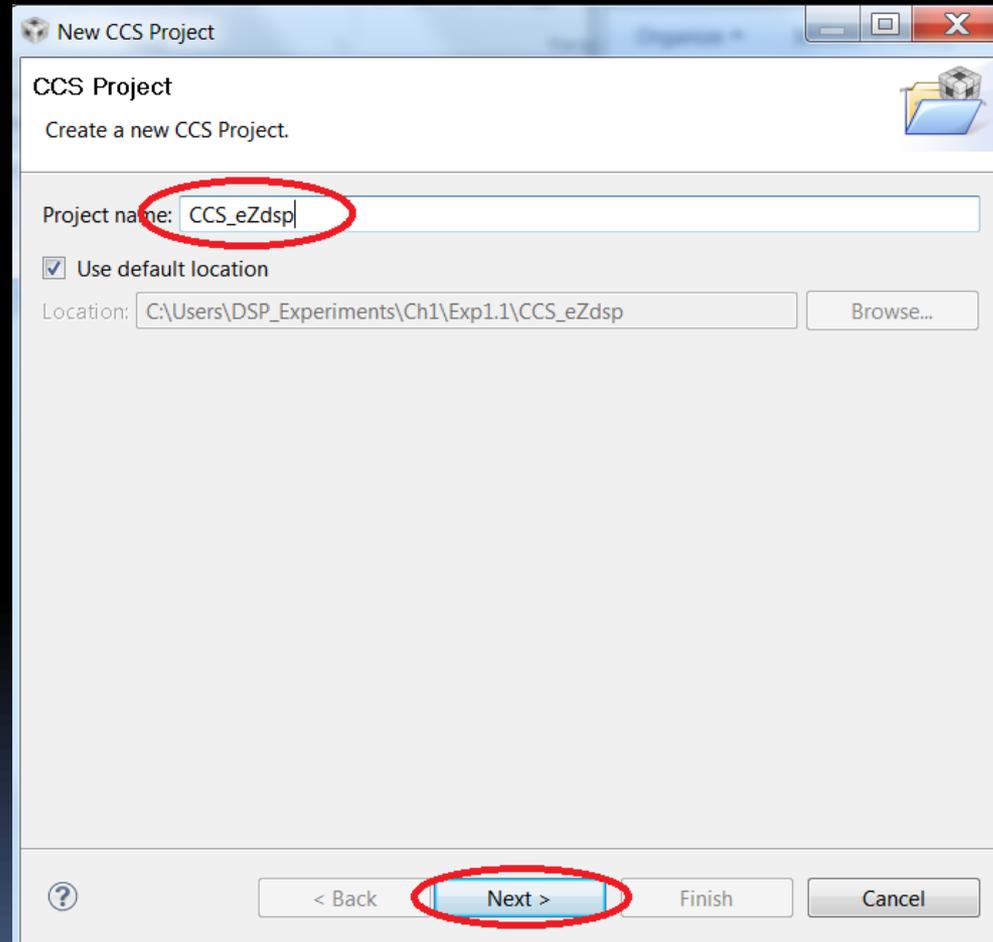
# Create a new project

(File -> New -> CCS Project)

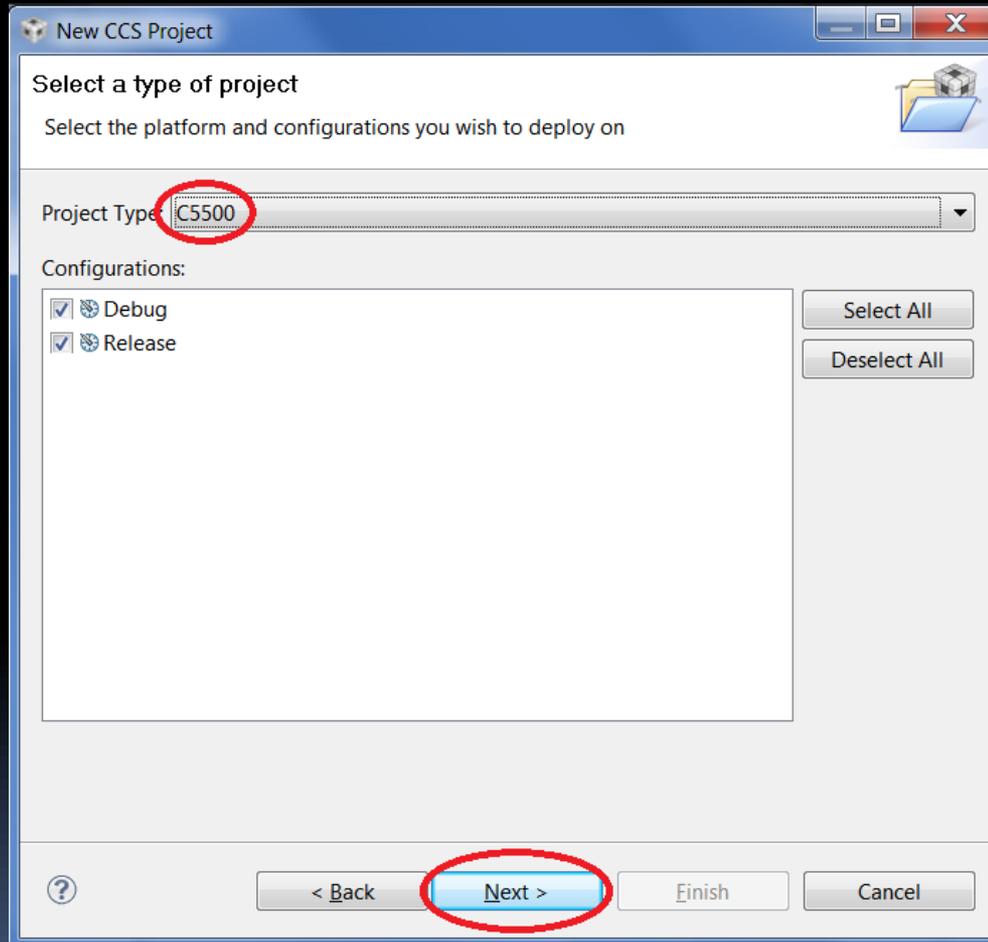


# Create a new project name

(Example: CCS\_eZdsp)

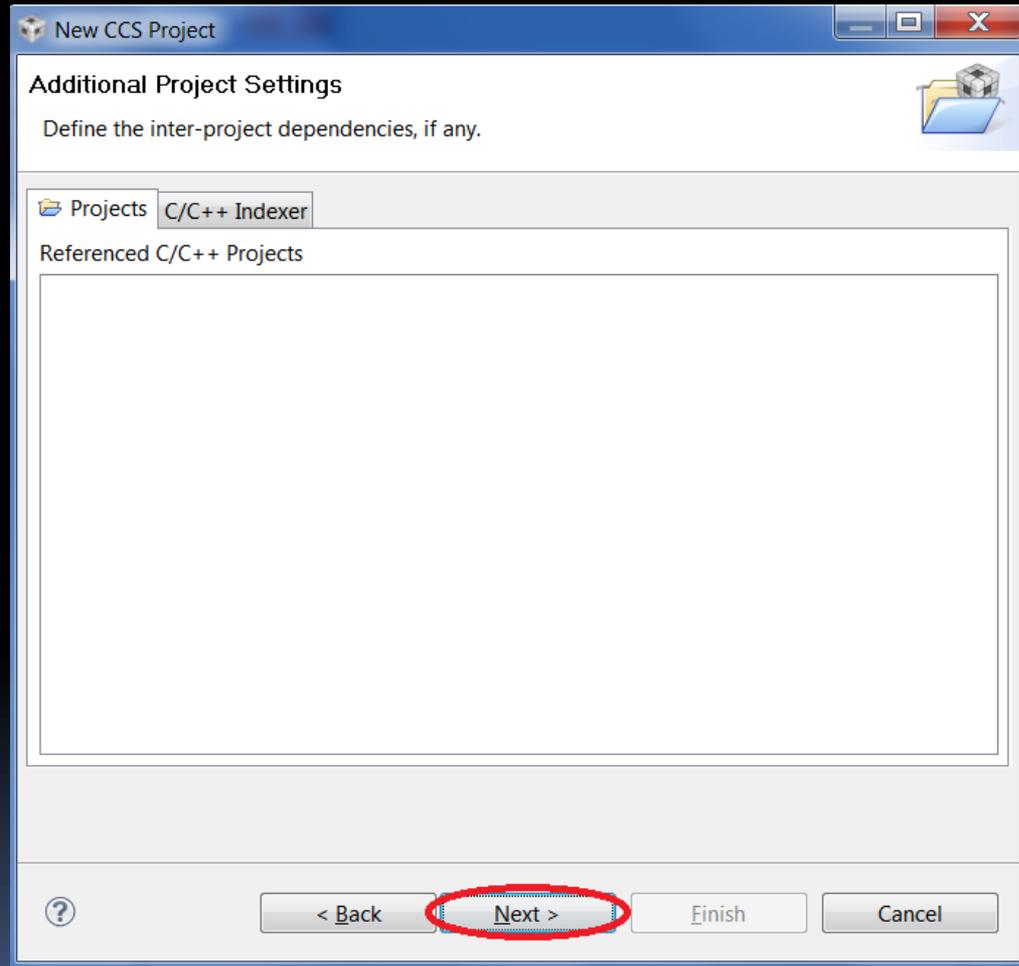


# Select C5500 as the new project



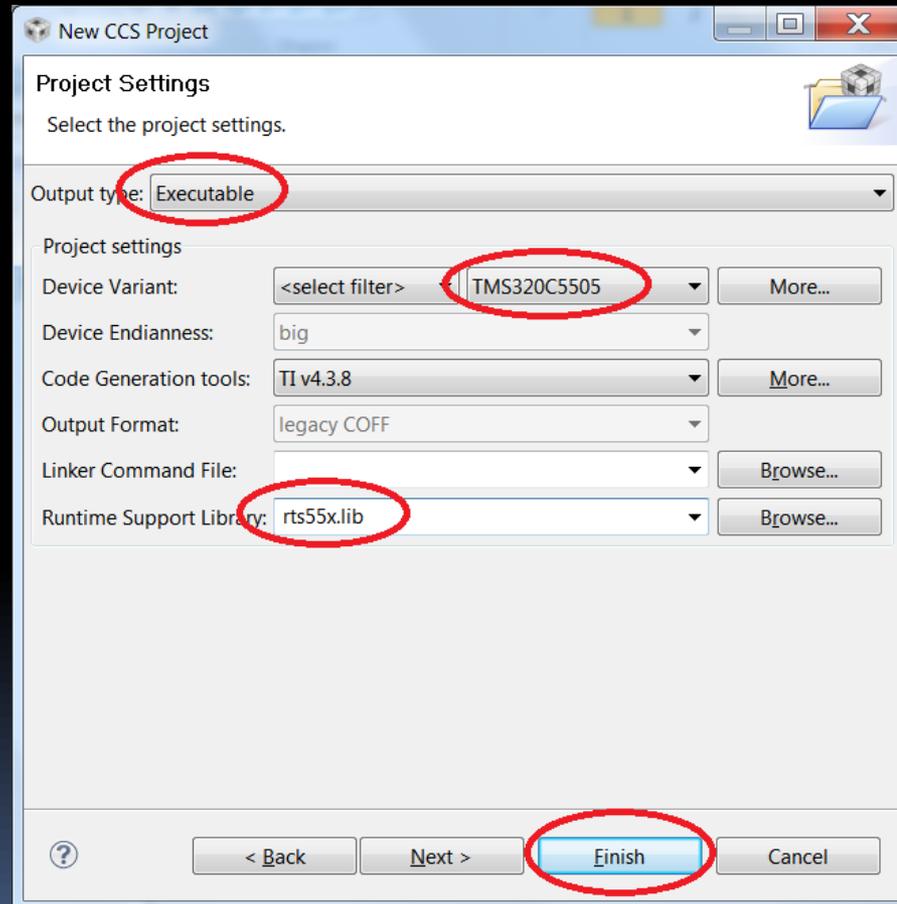
# Go to next

*(We do not need any additional settings)*

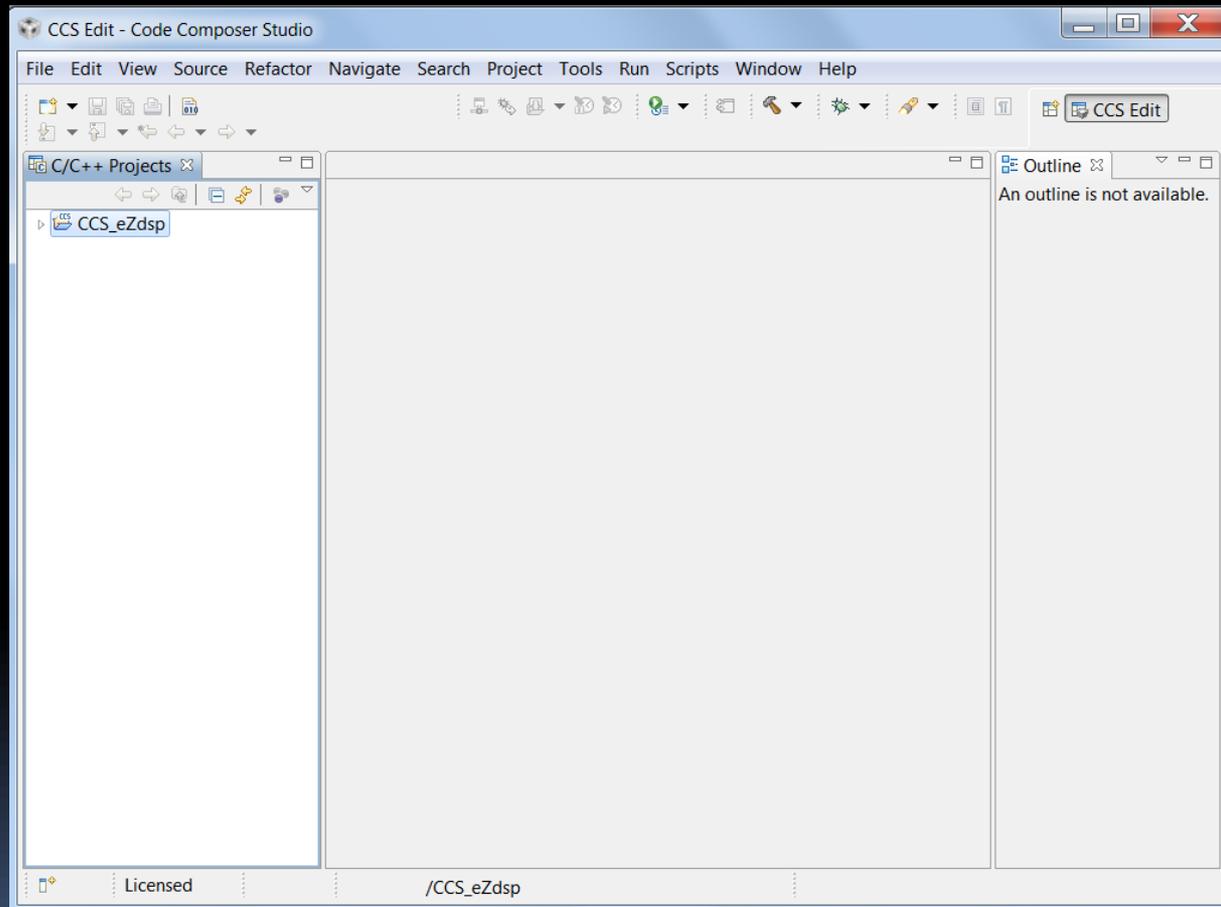


# Set up the project

*(Executable output, Device C5505, Library rts55h.Lib)*

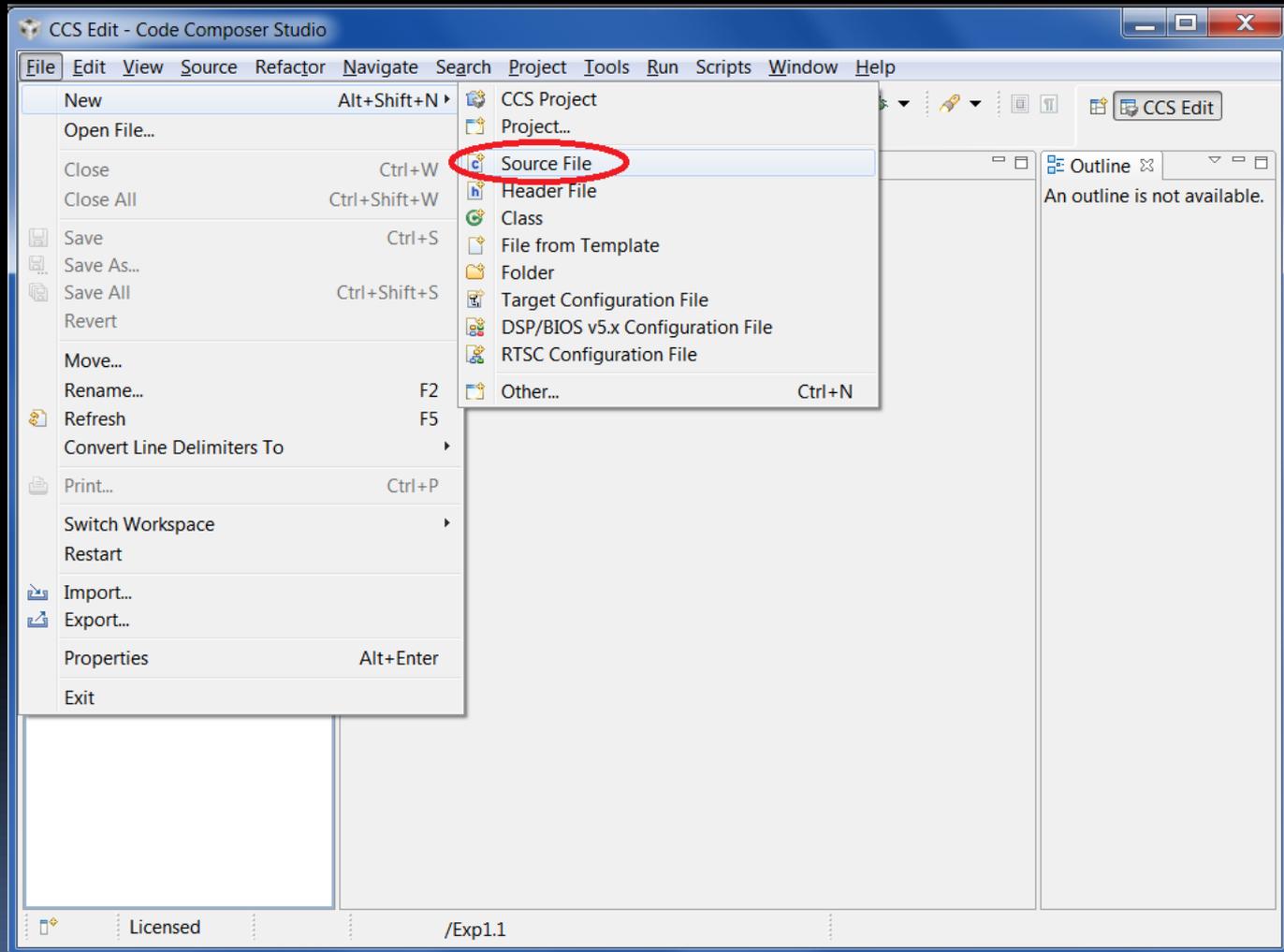


# Project: CCS\_eZdsp

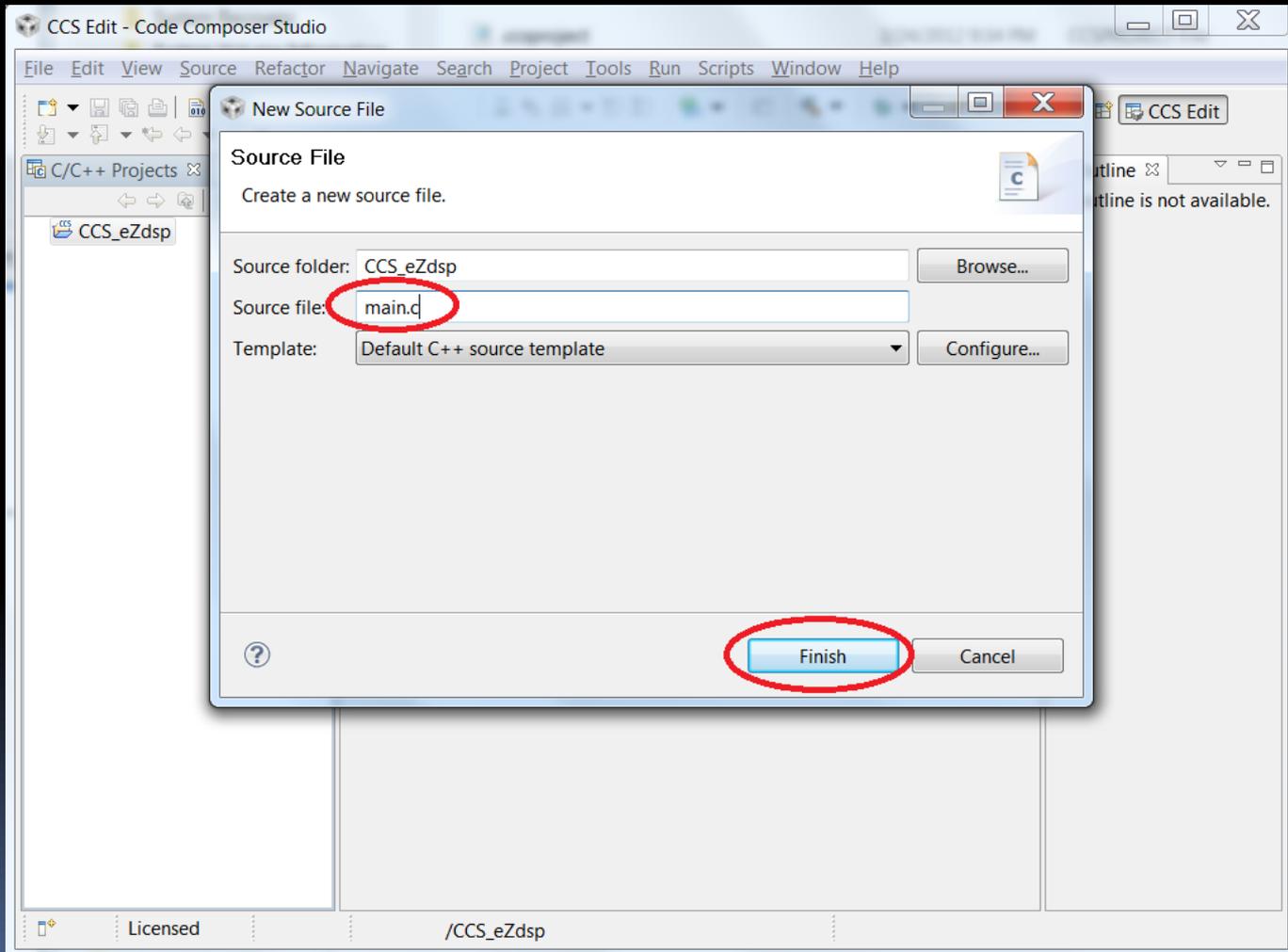


# Add Source File

(Example: File->New->Source File)



# Create C file main.c (1)

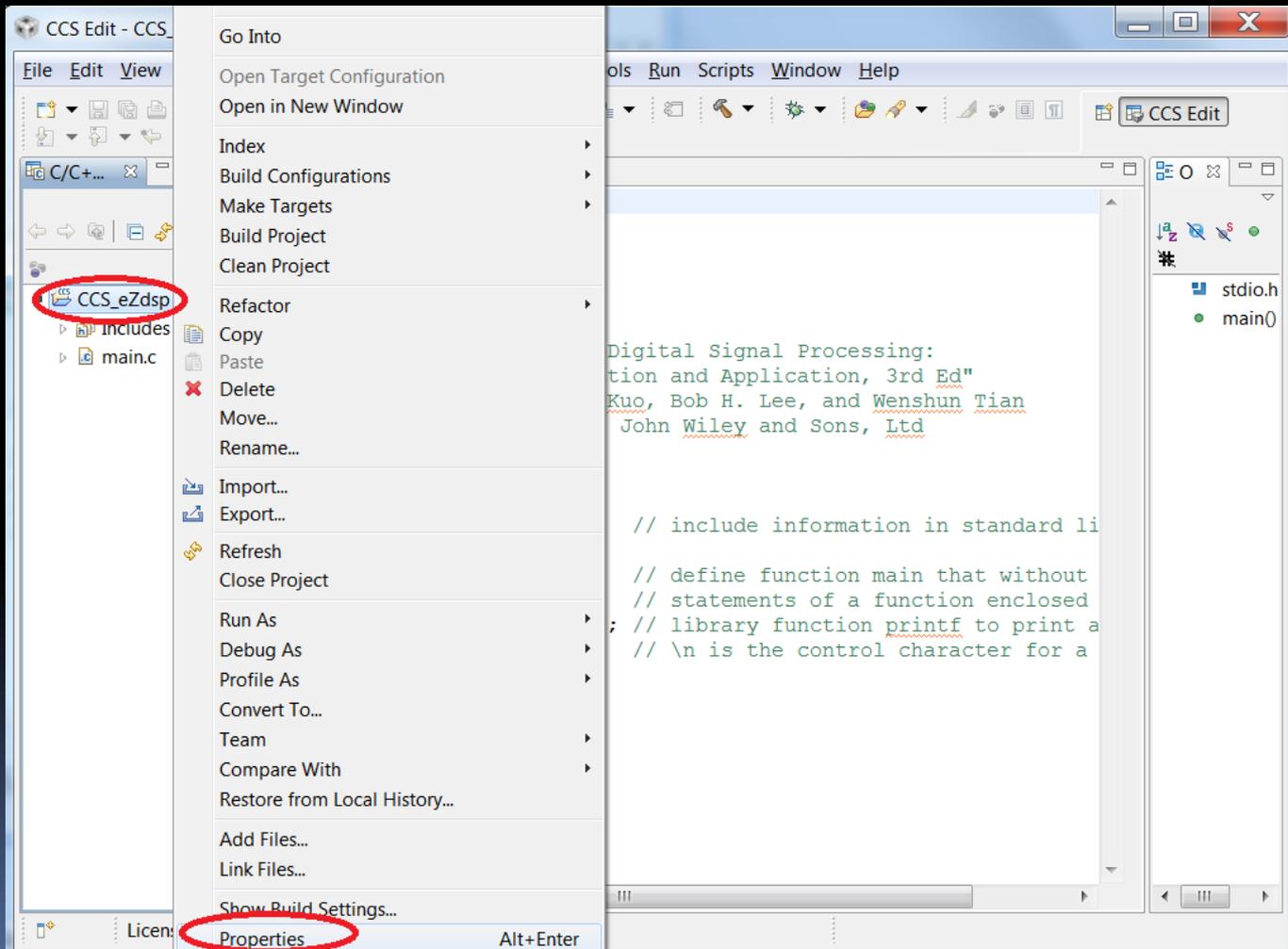


# Create C file main.c (2)

```
1 /*
2 * main.c
3 *
4 * Created on: MAR 10, 2012
5 * Author: BLEE
6 *
7 * For the book "Real Time Digital Signal Processing:
8 * Implementation and Application, 3rd Ed"
9 * By Sen M. Kuo, Bob H. Lee, and Wenshun Tian
10 * Publisher: John Wiley and Sons, Ltd
11 */
12
13
14 #include <stdio.h> // include information in standard li
15
16 void main() // define function main that without
17 { // statements of a function enclosed
18     printf("Hello World!\n"); // library function printf to print a
19 } // \n is the control character for a
20
21
```

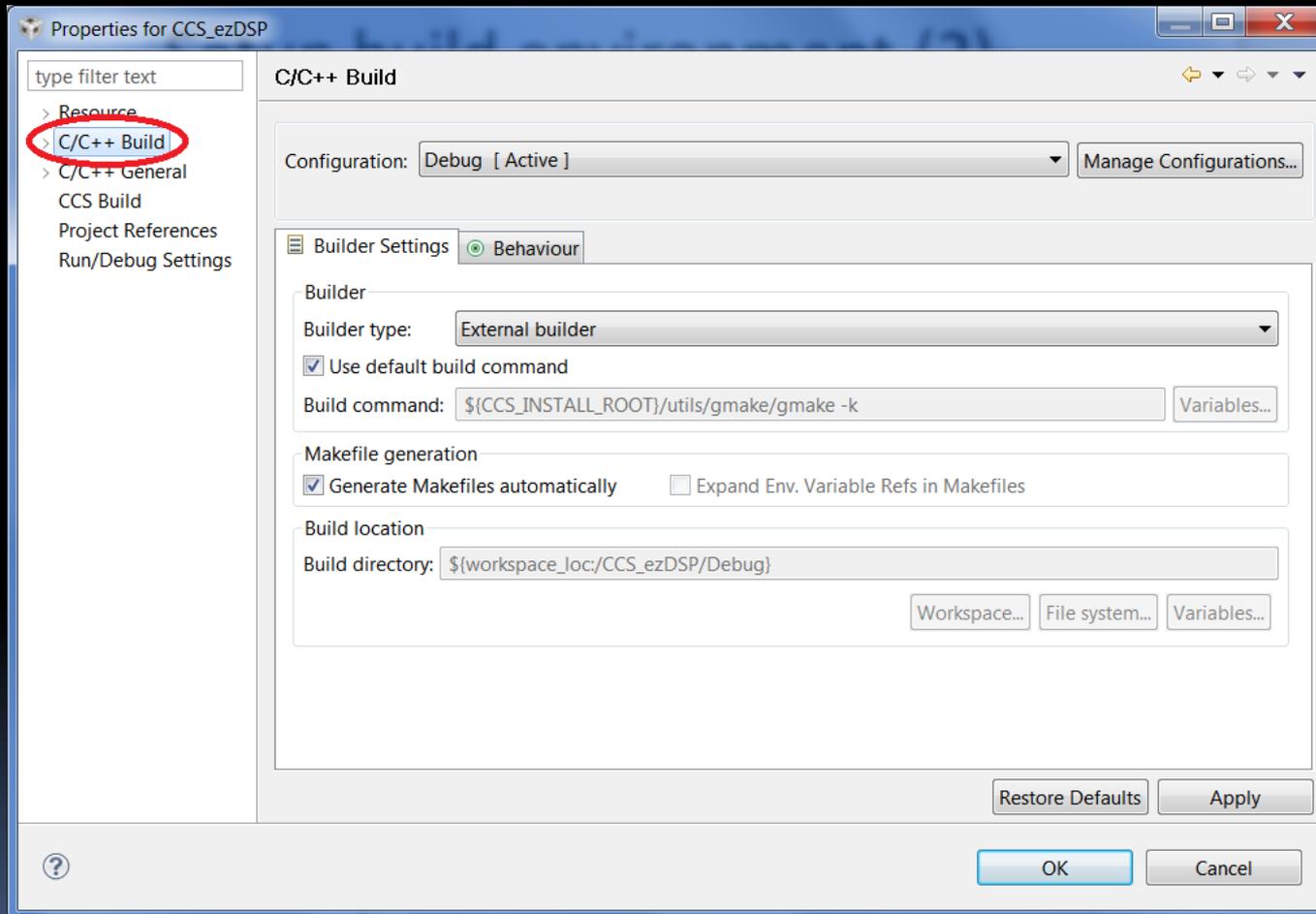
# Setup build environment (1)

*(Right click on CCS\_eZdsp then select Property)*



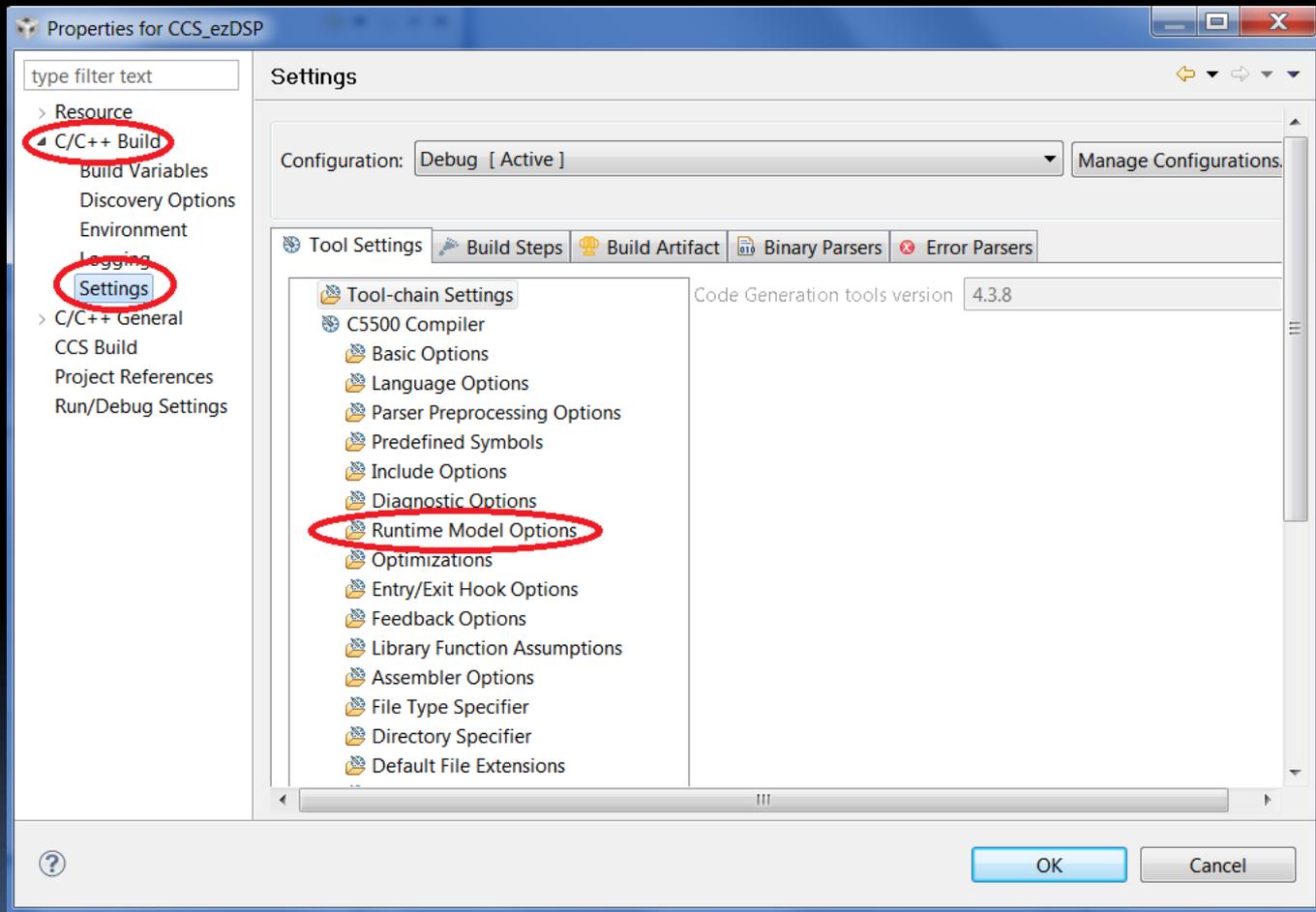
# Setup build environment (2)

*(Select and expand C/C++ Build option)*



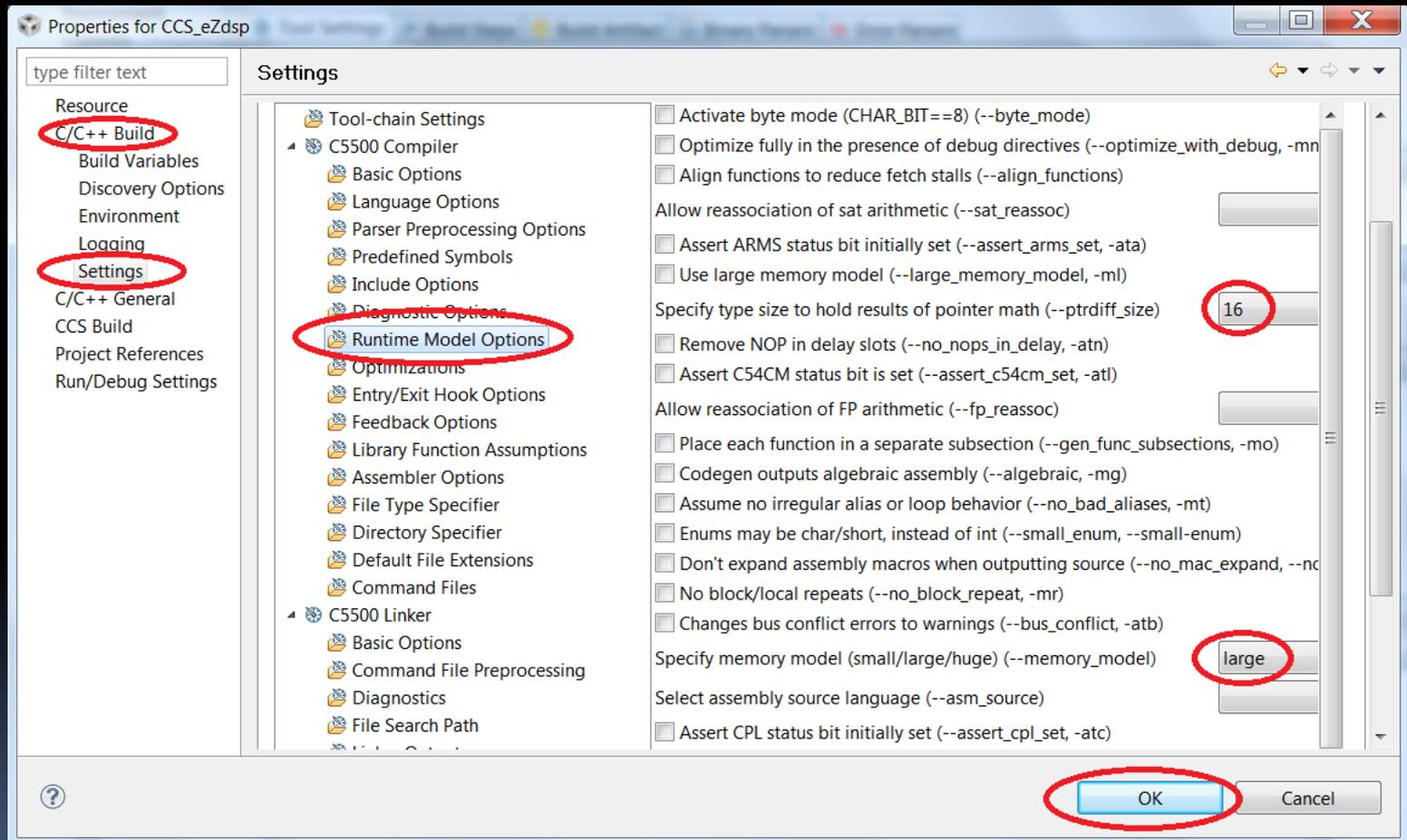
# Setup build environment (3)

*(Select Settings, then Runtime Options)*



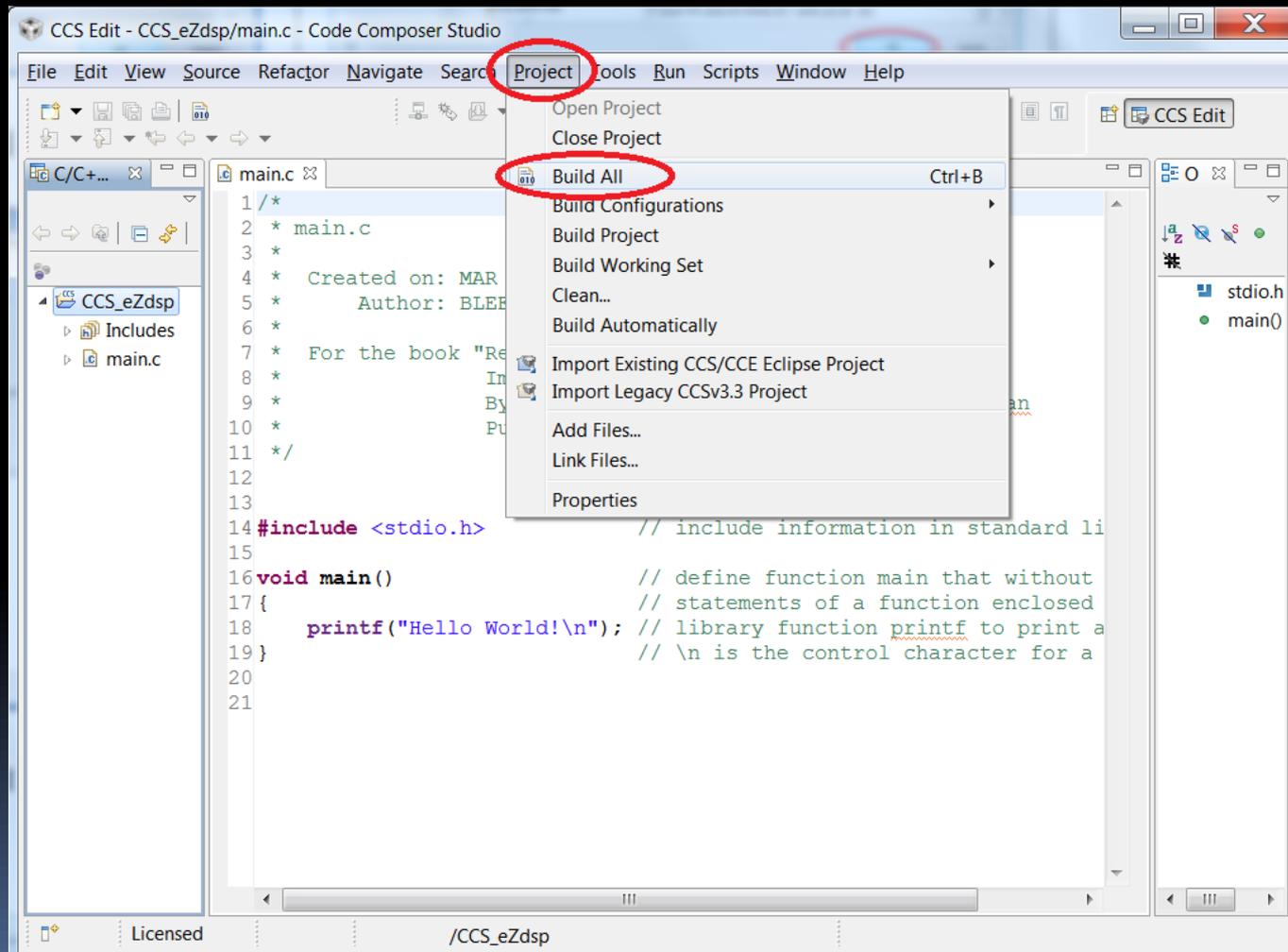
# Setup build environment (4)

(Set type size to 32 and memory model to huge)



# Build the project

(Project->Build ALL)



# Build result

(No errors, but there are 5 warnings)

The screenshot shows the Code Composer Studio (CCS) interface. The main editor displays a C program named `main.c` with the following content:

```
1 /*
2  * main.c
3  *
4  * Created on: MAR 10, 2012
5  * Author: BLEE
6  *
7  * For the book "Real Time Digital Signal Processing:
8  * Implementation and Application, 3rd Ed"
9  * By Sen M. Kuo, Bob H. Lee, and Wenshun Tian
10 * Publisher: John Wiley and Sons, Ltd
11 */
12
13
14 #include <stdio.h> // include information in standard li
15
16 void main() // define function main that without
17 { // statements of a function enclosed
18     printf("Hello World!\n"); // library function printf to print a
```

The Console window shows the build output for `C-Build [CCS_eZdsp]`:

```
size
warning: creating ".sysstack"
section with default size of
0x3e8; use the
-sysstack option to change the
default size
```

The Problems window shows the build result: `0 errors, 5 warnings, 0 others`. The warnings are listed as follows:

- Warnings (5 items)
- creating ".stack" section with default size o
- creating ".sysmem" section with default siz
- creating ".sysstack" section with default siz

The warnings are circled in red in the original image.

# Correct warnings

- The warnings are due to undefined symbols for the memory sections during the program link time.
- The linker uses default settings for these missing sections.
- To correct this, copy the file, `Ink.cmd`, from CCS C55x folder to the project folder.
- The Project folder is:  
*C:\Users\DSP\_Experiments\Ch1\Exp1.1\CCS\_eZdsp.*
- CCS C55x folder could be at (depending your CCS version):  
*C:\ti\ccsv5\ccs\_base\_5.0.3.00023\emulation\boards\usbstk5505.*
- Or copy the `Ink.cmd` file from the software came with the book.

# Create linker command file

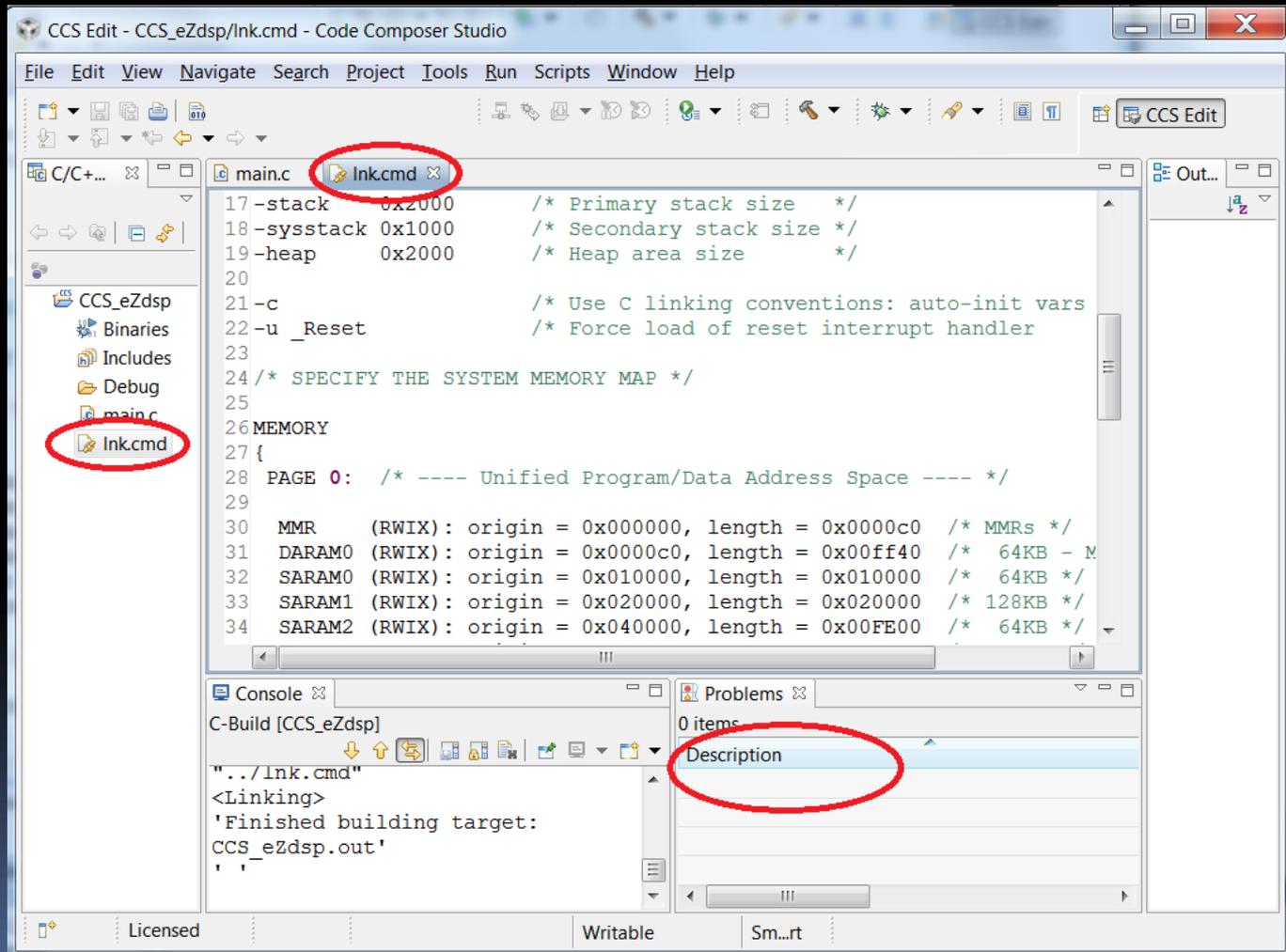
(Using a text editor create Ink.com as below)

```
/******  
/* C5505.cmd - COMMAND FILE FOR LINKING C PROGRAMS IN LARGE/HUGE MEMORY  
MODEL */  
/******  
-stack 0x2000 /* Primary stack size */  
-sysstack 0x1000 /* Secondary stack size */  
-heap 0x2000 /* Heap area size */  
  
-c /* Use C linking conventions: auto-init vars at runtime */  
-u _Reset /* Force load of reset interrupt handler */  
  
MEMORY  
{  
  MMR (RW) : origin = 0000000h length = 0000c0h /* MMRs */  
  DARAM (RW) : origin = 0000c0h length = 00ff40h /* on-chip DARAM */  
  SARAM (RW) : origin = 0030000h length = 01e000h /* on-chip SARAM */  
  
  SAROM_0 (RX) : origin = 0fe0000h length = 008000h /* on-chip ROM 0 */  
  SAROM_1 (RX) : origin = 0fe8000h length = 008000h /* on-chip ROM 1 */  
  SAROM_2 (RX) : origin = 0ff0000h length = 008000h /* on-chip ROM 2 */  
  SAROM_3 (RX) : origin = 0ff8000h length = 008000h /* on-chip ROM 3 */  
  
  EMIF_CS0 (RW) : origin = 0050000h length = 07B0000h /* mSDR */  
  EMIF_CS2 (RW) : origin = 0800000h length = 0400000h /* ASYNC1 : NAND */  
  EMIF_CS3 (RW) : origin = 0C00000h length = 0200000h /* ASYNC2 : NAND */  
  EMIF_CS4 (RW) : origin = 0E00000h length = 0100000h /* ASYNC3 : NOR */  
  EMIF_CS5 (RW) : origin = 0F00000h length = 00E0000h /* ASYNC4 : SRAM */  
}
```

## SECTIONS

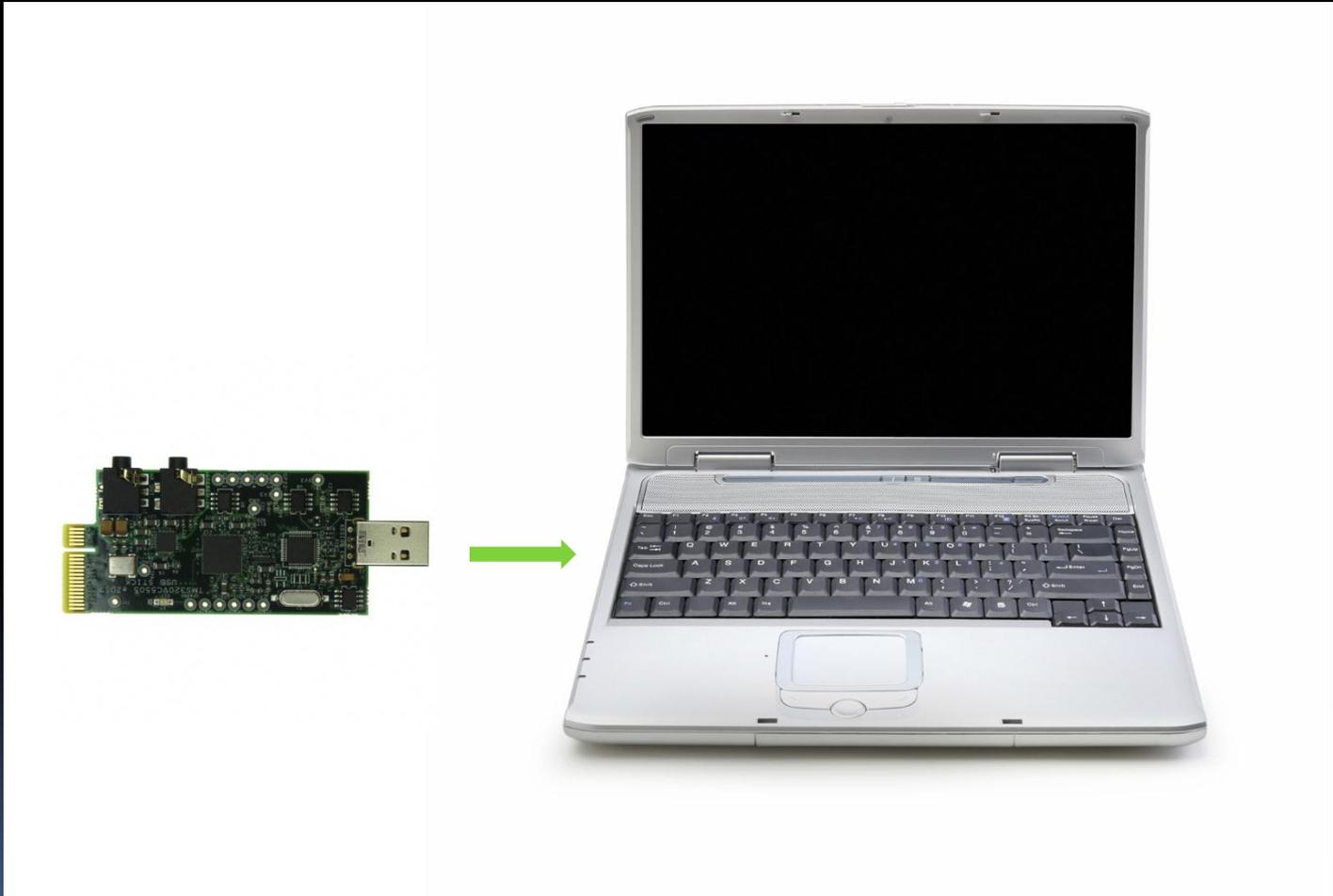
```
{  
  vectors (NOLOAD)  
  .bss :> DARAM /* fill = 0 */  
  vector :> DARAM ALIGN = 256  
  .stack :> DARAM  
  .sysstack :> DARAM  
  .systemem :> DARAM  
  .text :> SARAM  
  .data :> DARAM  
  .cinit :> DARAM  
  .const :> DARAM  
  .cio :> DARAM  
  .usect :> DARAM  
  .switch :> DARAM  
  .emif_cs0 :> EMIF_CS0  
  .emif_cs2 :> EMIF_CS2  
  .emif_cs3 :> EMIF_CS3  
  .emif_cs4 :> EMIF_CS4  
  .emif_cs5 :> EMIF_CS5  
}
```

# Once Ink.cmd file is copied (Rebuild the project and there are no warnings)



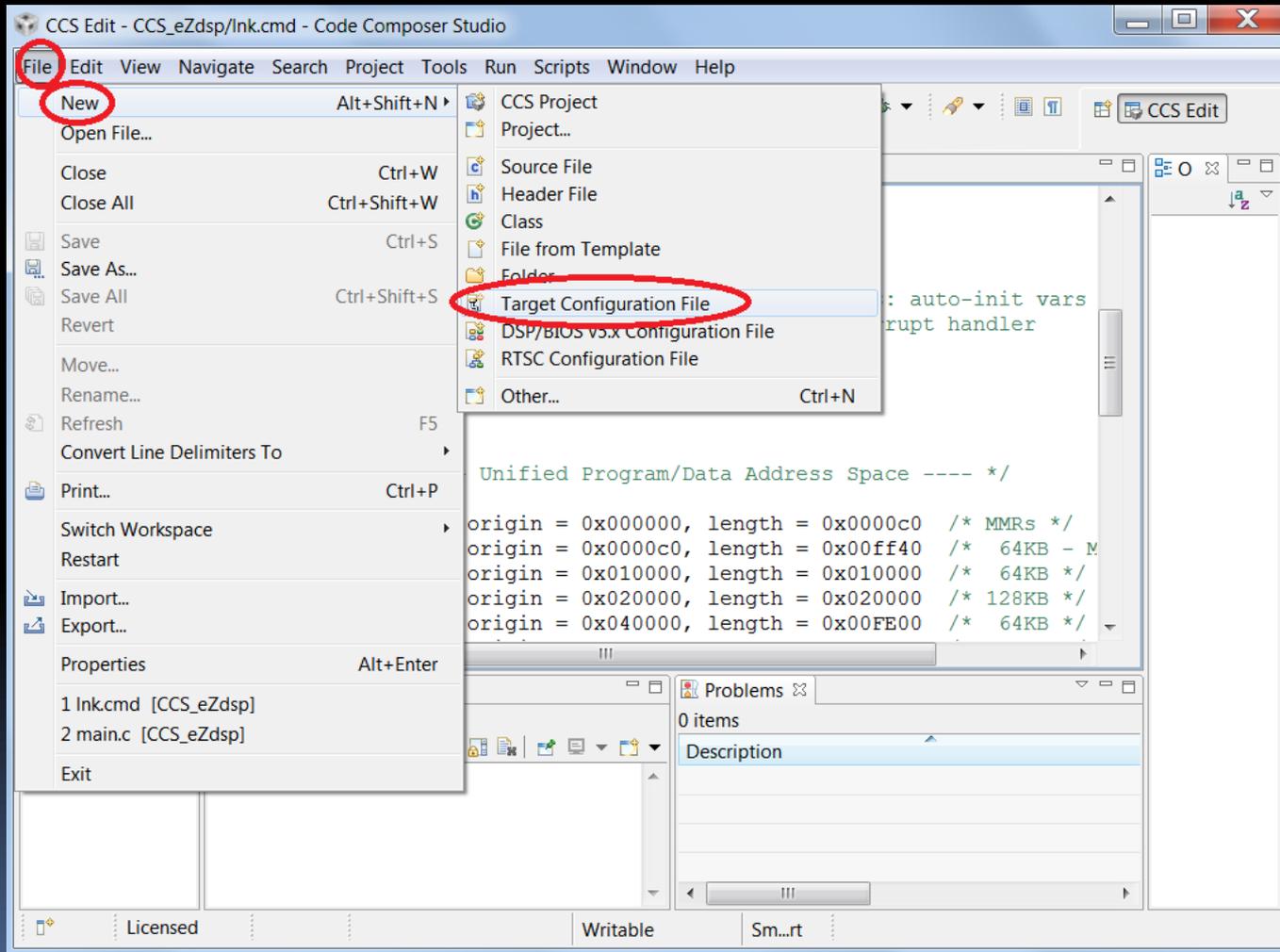
# Connect eZdsp to PC

(Refer to eZdsp Start Guide for settings)



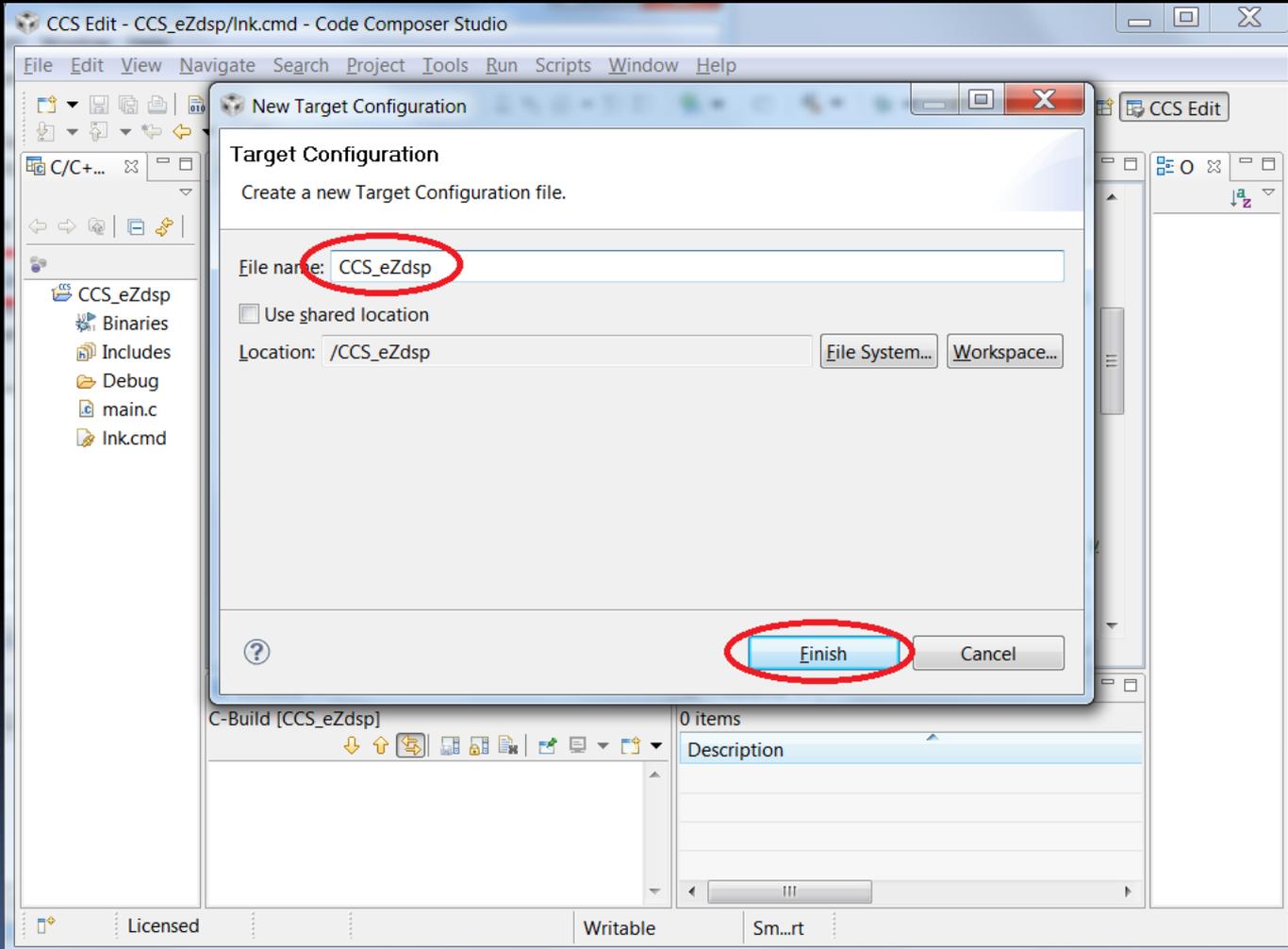
# Set target configuration (1)

(Right click on File->New->Target Configuration File)



# Set target configuration (2)

*(Create a target configuration file name)*



# Set target configuration (3)

(Select XDS100v2 USB Emulator, USBSTK5505, & Save)

The screenshot shows the CCS Edit interface for configuring a target. The 'Basic' tab is active, and the 'General Setup' section is expanded. The 'Connection' dropdown is set to 'Texas Instruments XDS100v2 USB Emulator'. Below it, the 'Device' list includes several options, with 'USBSTK5505' selected and checked. The 'Save' button in the 'Save Configuration' section is also highlighted. The 'Problems' window at the bottom right shows '1 error, 0 warnings, 0 others'.

CCS Edit - CCS\_eZdsp/CCS\_eZdsp.ccxml - Code Composer Studio

File Edit View Navigate Search Project Tools Run Scripts Window Help

Basic

**General Setup**

This section describes the general configuration about the target.

Connection: Texas Instruments XDS100v2 USB Emulator

Device: type filter text

- TMDSEVM6457L
- TMDSEVM6474L
- USBSTK5505
- USBSTK5515
- controlSTICK - Piccolo F28027
- AM1707
- AM1808

Advanced Setup

Target Configuration:

Save Configuration

Save

Console: C-Build [CCS\_eZdsp]

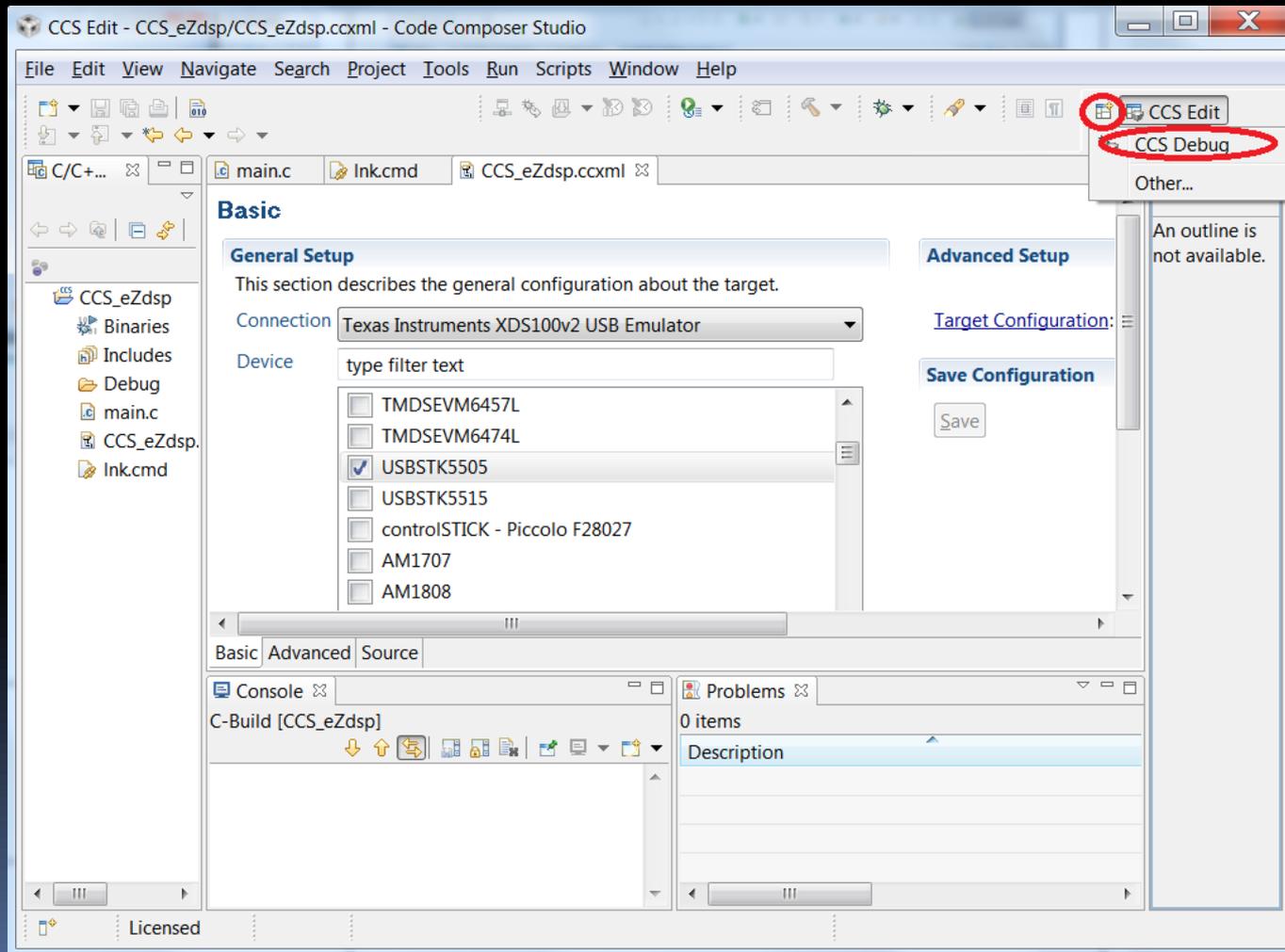
Problems: 1 error, 0 warnings, 0 others

Description

- Errors (1 item)

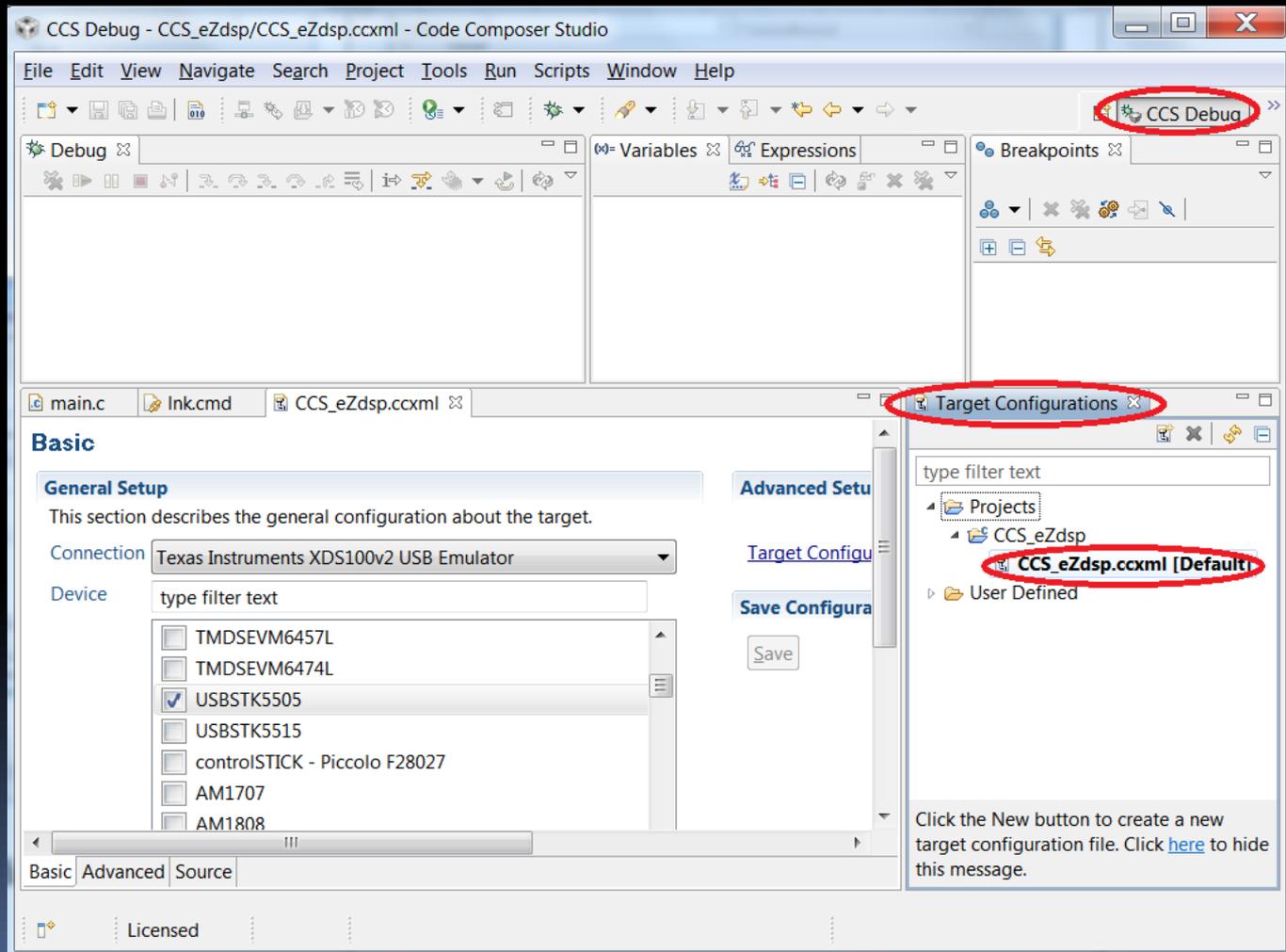
# Change edit mode to debug mode

(Change from CCS Edit to CCS Debug Mode)



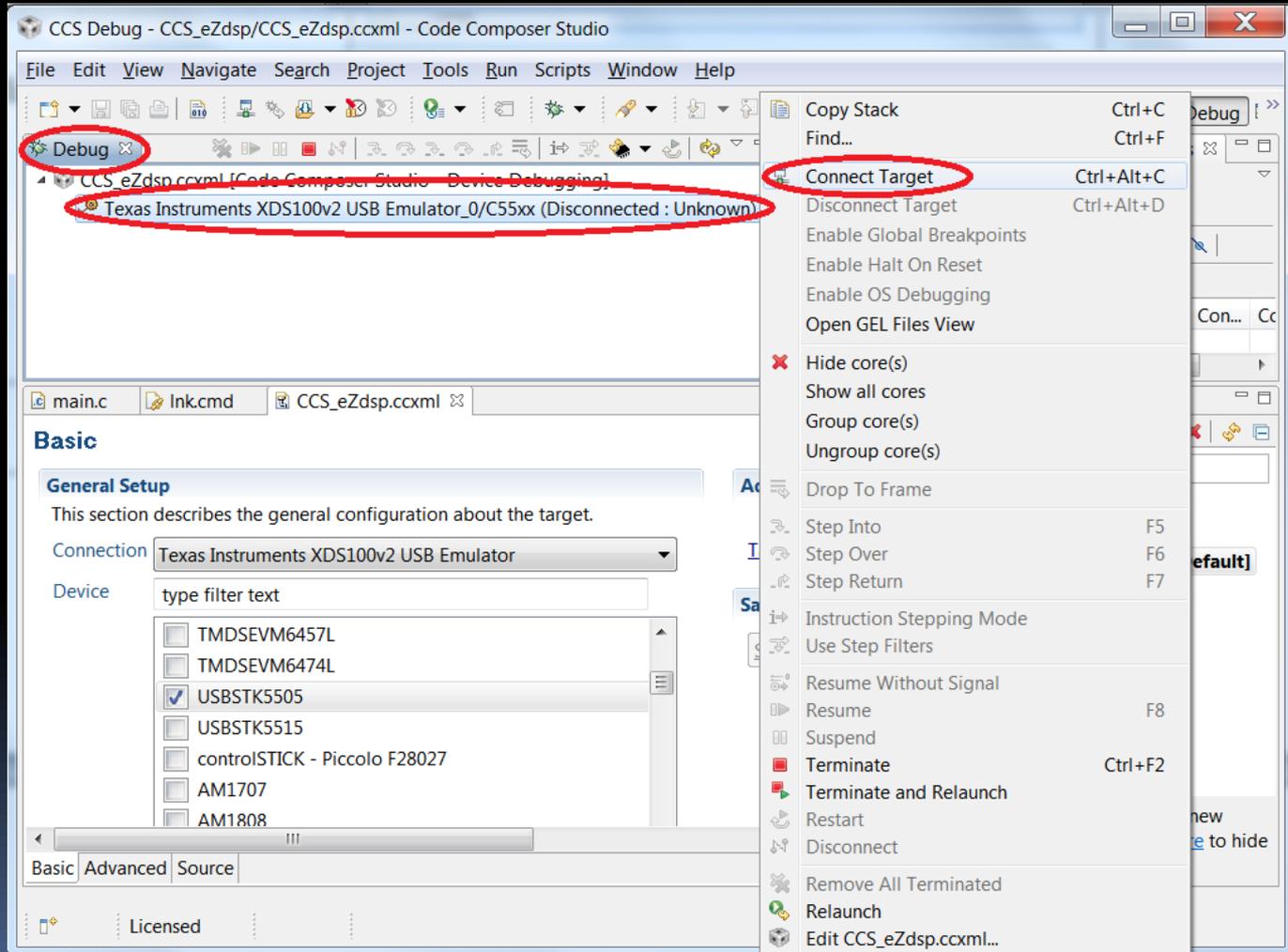
# Launch selected configuration

(Target Configuration->Project->right click on CCS\_eZdsp.ccxml)



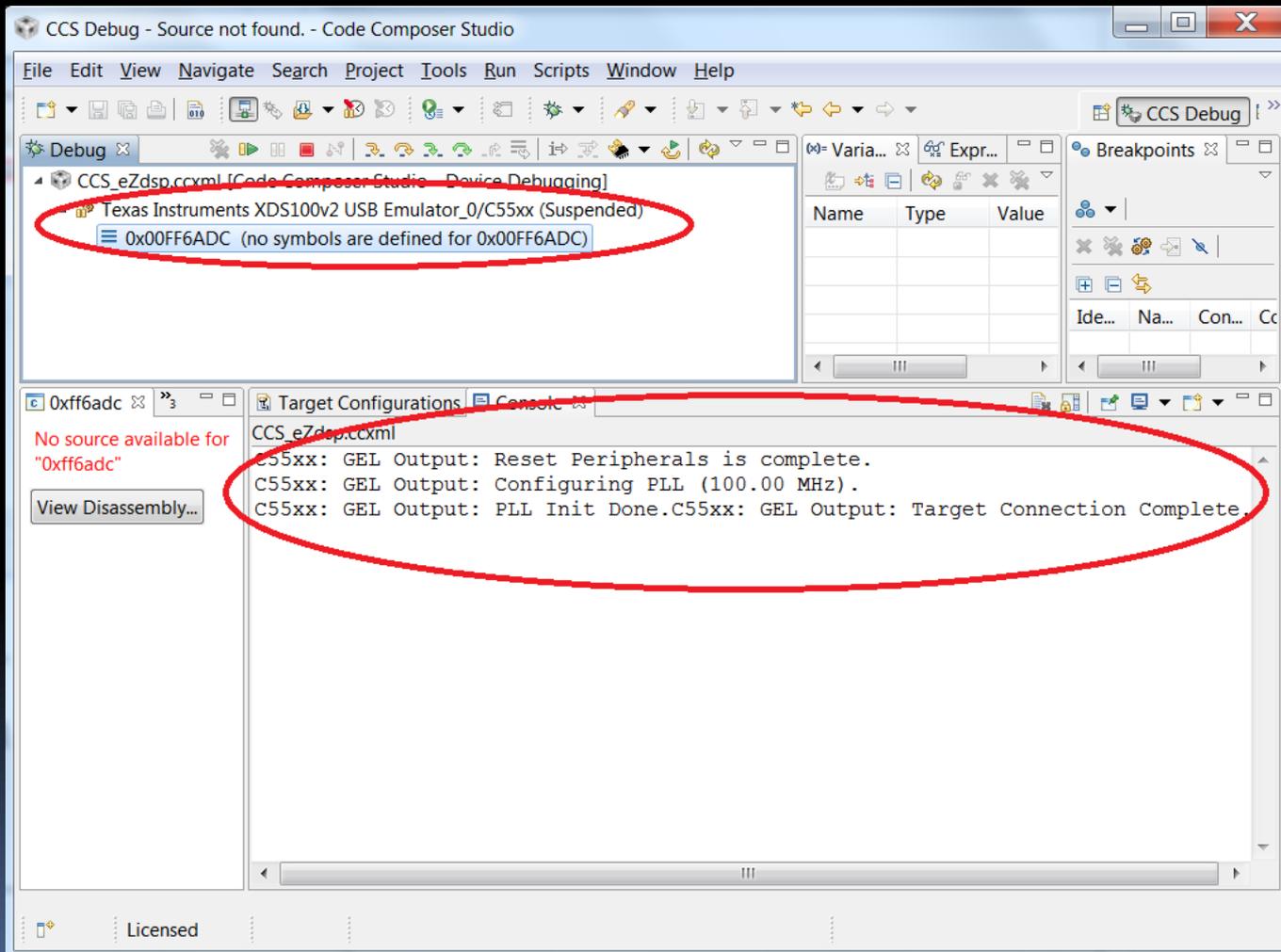
# Connect eZdsp

*(Right click on USB\_Emulator\_0, then Connect Tart)*



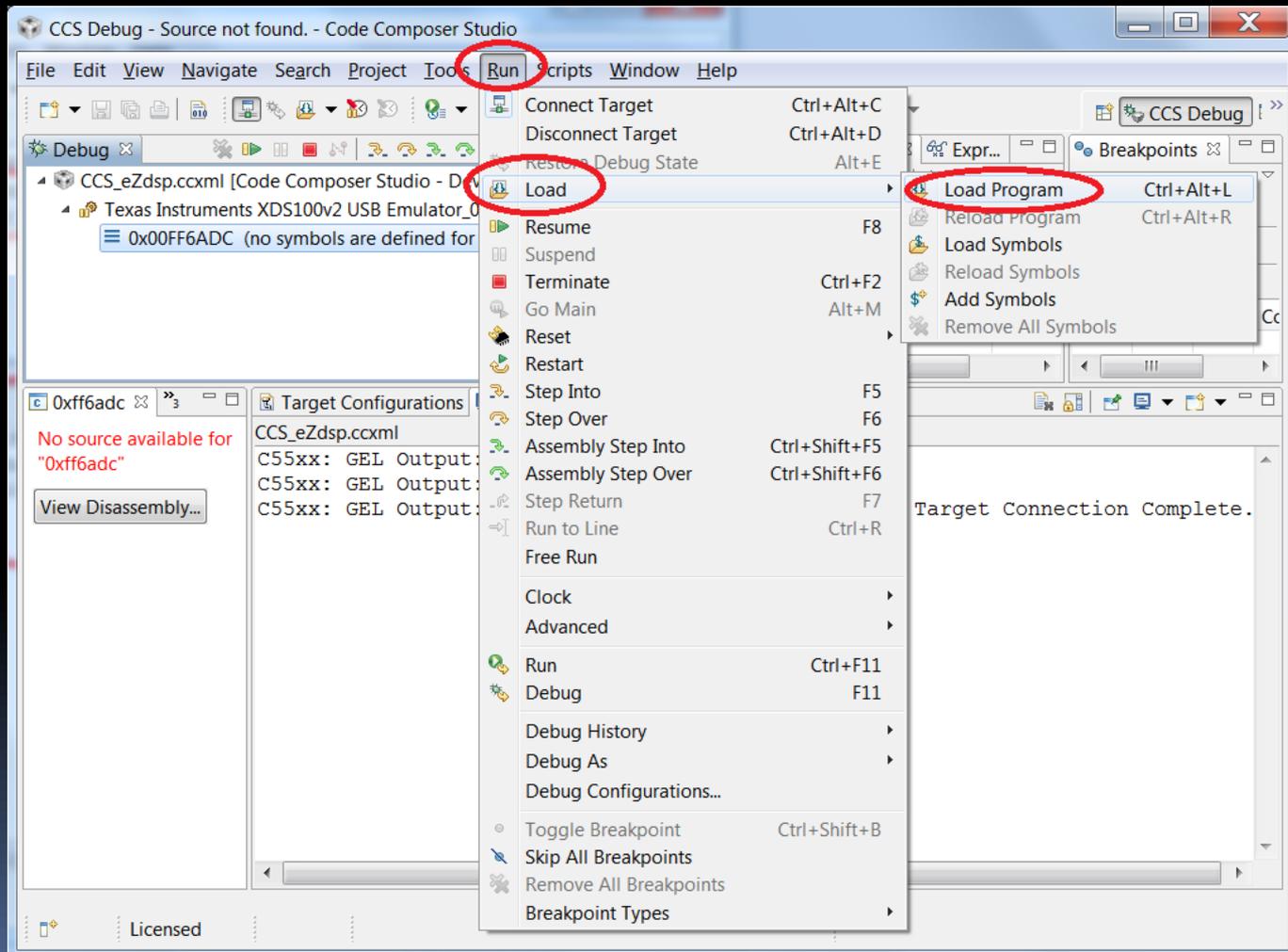
# eZdsp connected

*(Target reset and configured automatically)*



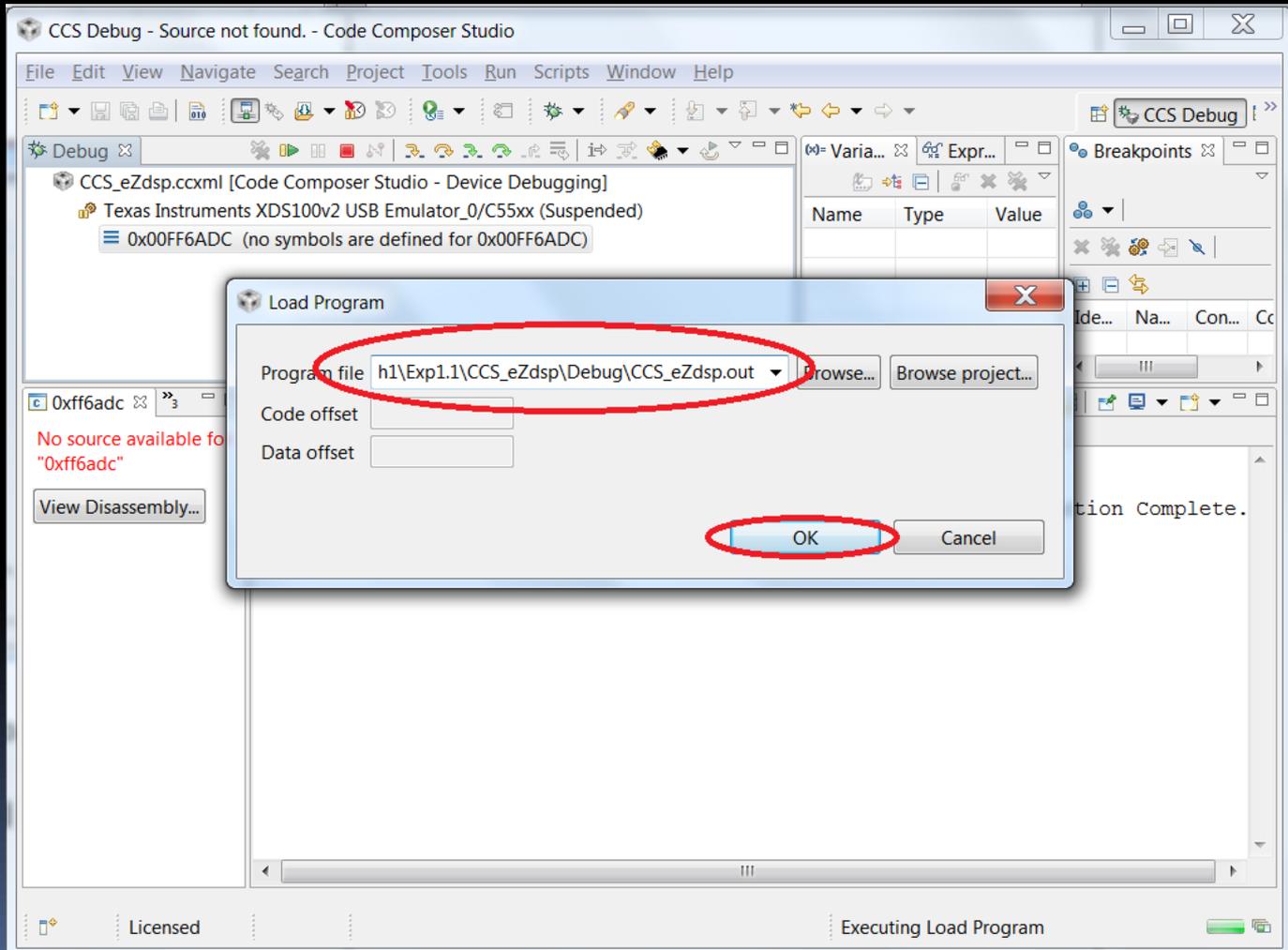
# Load program CCS\_eZdsp.out (1)

(From Run->Load->Load Program)



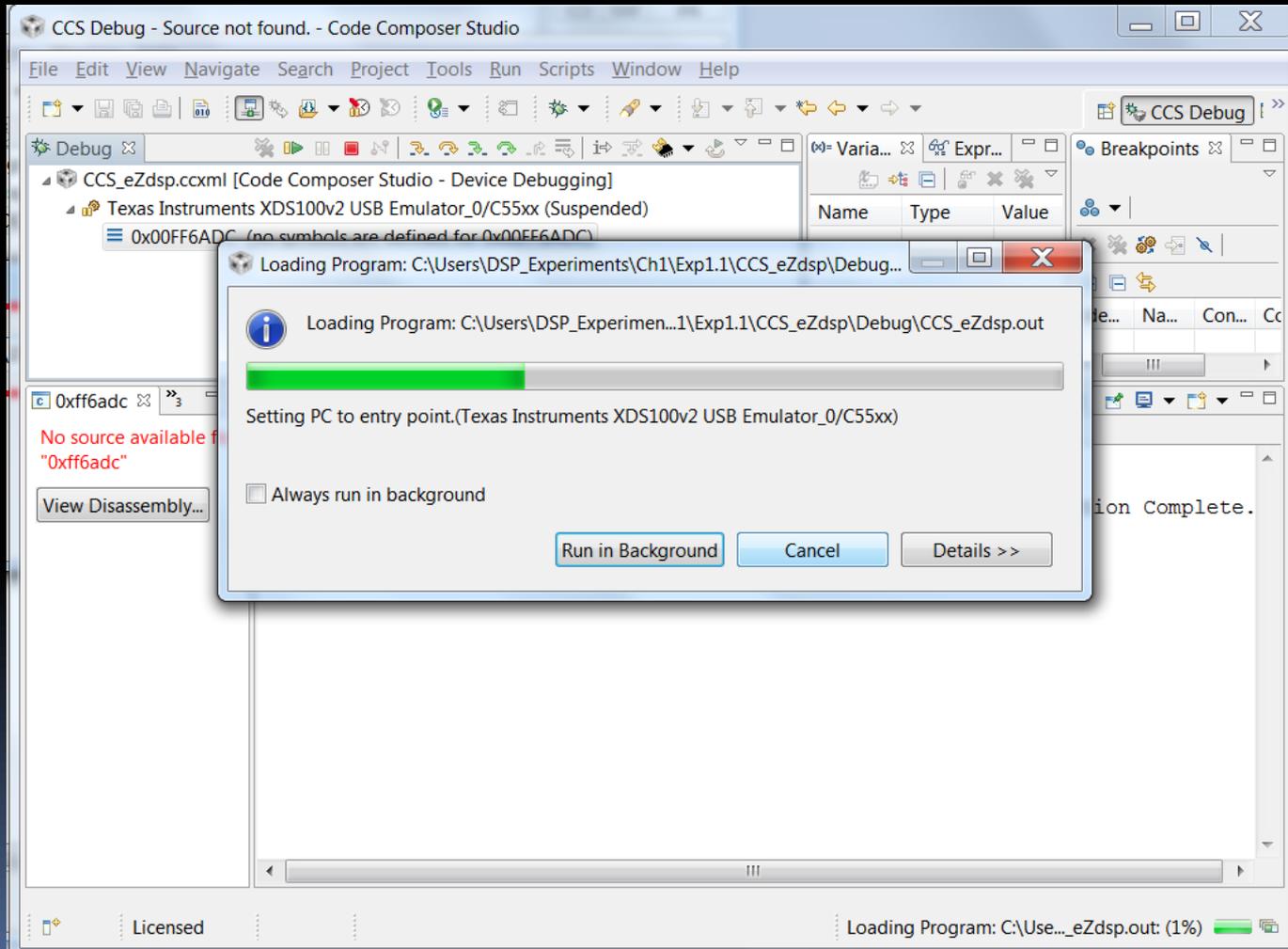
# Load program CCS\_eZdsp.out (2)

(...\DSP\_Experiments\Ch1\Exp1.1\CS\_eZdsp\Debug\CCS\_eZdsp.out)



# Load program CCS\_eZdsp.out (3)

*(CCS\_eZdsp program is Loading)*



# Once the program is loaded

(Program counter is at entry point of function main() )

The screenshot displays the Code Composer Studio (CCS) interface during a debug session. The main window shows the source code for `main.c` with the `main()` function highlighted. The console window shows the following output:

```
CCS_ezDSP.ccxml
C55xx: GEL Output: Reset Peripherals is compl
C55xx: GEL Output: Configuring PLL (100.00 MH
C55xx: GEL Output: PLL Init Done.C55xx: GEL O
```

The `main()` function in `main.c` is shown as follows:

```
2 * main.c
3 *
4 * Created on: Dec 31, 2011
5 * Author: YOUR NAME
6 */
7
8 #include <stdio.h>
9
10 void main()
11 {
12     printf("Hello World!\n");
13 }
```

# Once the program is loaded (Program counter is at entry point of function main() )

The screenshot shows the Code Composer Studio (CCS) interface. The main window displays the source code for `main.c`. The `main()` function is highlighted in green, and the program counter is at its entry point. The console window shows the output of the program, which is "Hello World!\n".

The CCS Debug window shows the following structure:

- CCS\_eZdsp.ccxml [Code Composer Studio - Device Debugging]
  - Texas Instruments XDS100v2 USB Emulator\_0/C55xx (Suspended)
    - main() at main.c:17 0x00023FFE
    - \_args\_main() at args\_main.c:25 0x00023E0F (\_args\_main has only skeletal debug)

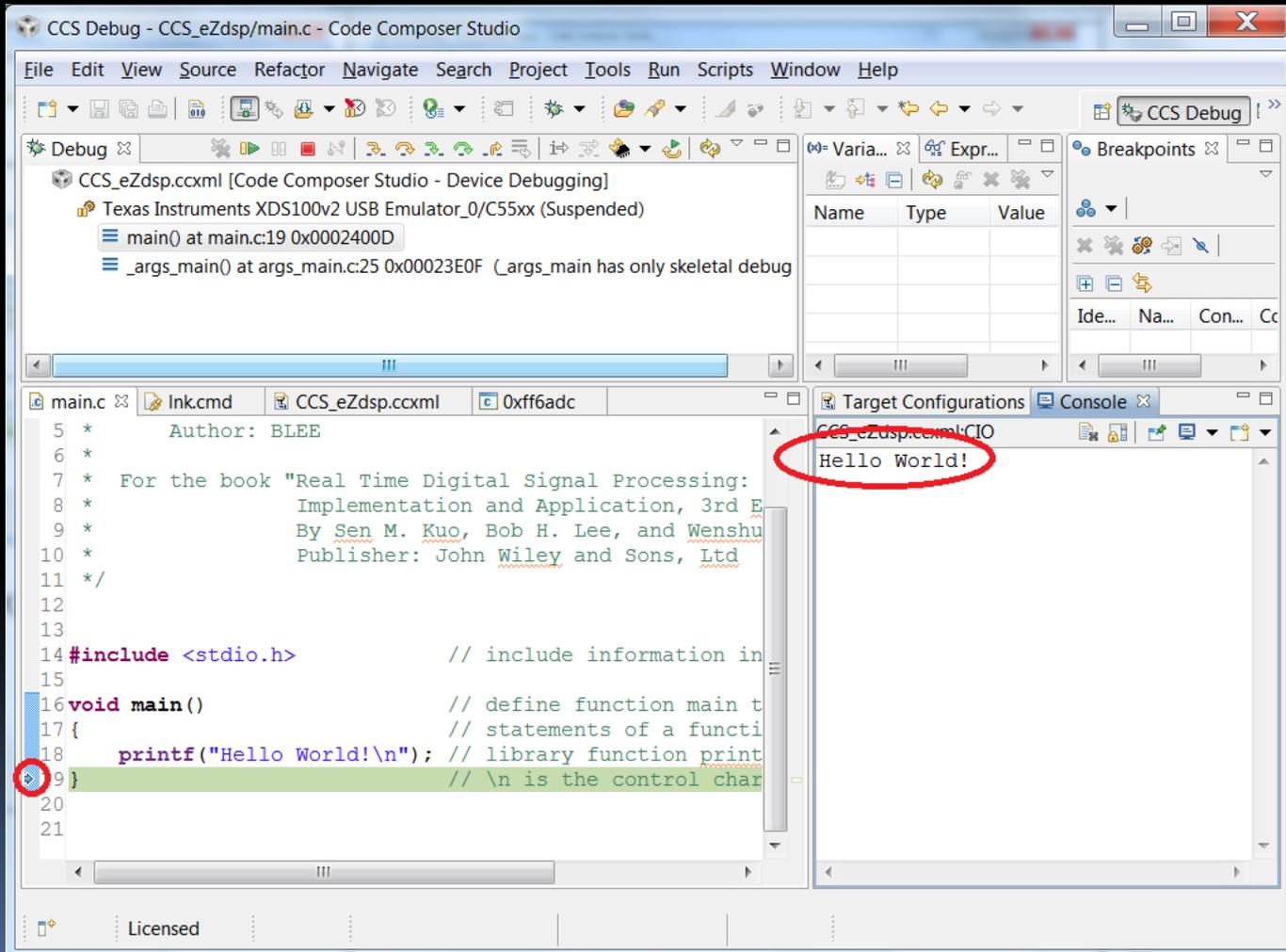
The console window shows the following output:

```
CCS_eZdsp.ccxml
C55xx: GEL Output: Reset Periphera
C55xx: GEL Output: Configuring PLL
C55xx: GEL Output: PLL Init Done.C
```



# Step Over (F6) the program

(Once PC passed printf( ), "Hello World!" is displayed)



# New experimental assignments

- Load the *CCS\_eZdsp.out*, *Step Over (F6)* through the program. Then, use *CCS to Reload Program* feature to load program again.

Q1: where does the program counter point at in the *main()* after program has been reloaded?

- Use *Resume (F8)*, instead of *Step Over*, to run the program again.

Q2: what will be showing on the Console display window?

- After running the program, use *Restart* and *Resume (F8)*, run the program again.

Q3: what will be showing on the Console display window?

- Exit CCS, then restart CCS and create your own program that will display the message "C55 DSP eZdsp Test" on the console.

# Programming quick review

- C program is one the most important programming languages for DSP applications.
- C program is written in a text file with file extension ".c", such as main.c in this experiment. The files with the ".c" extension are called C source files.
- C program start with a function called main, as in this experiment, *main()*.
- Each function has a prototype, the example *main()* function used in this experiment declares the prototype of the function *main()* as void.
- The *main()* usually calls other functions to perform tasks. Some of the functions are from the libraries provided by the C compiler tools include in CCS, and others are written by users like you.
- The *printf()* is a function provided by the library, *stdio.lib*. The function *printf* is an input/output (CIO) function. It prints messages to the C output device.
- The function is defined by a name with a pair of (). Inside (), it may have one or more arguments, or no argument at all. For example, *main()* does not have an argument while *printf()* has a string argument "Hello World!" in it. When the function *printf()* is called, it prints the string "Hello World!" on the screen.
- The experiment C file, main.c, has a line "#include <stdio.h>". The file *stdio.h* contains the function definition of the *printf* function. It tells C compiler to include the information from C I/O libraries. The files with ".h" extensions are called header files, which are also source files like C files. The standard include files are placed inside <>, as <stdio.h> with #include to indicate to the compiler this is an include file.
- The C program uses the "/\*" and "\*/" for comments. Anything inside the /\* and \*/ will be ignored by compiler. It is a good programming practice to use comments to document the data flow and logic of the program. Another way for adding comments is to use double slash //. Anything after the // on that same line will be ignored by C compiler.

# References

- <http://processors.wiki.ti.com/index.php/Category:CCS>
- [http://processors.wiki.ti.com/index.php/Category:Code\\_Composer\\_Studio\\_v5](http://processors.wiki.ti.com/index.php/Category:Code_Composer_Studio_v5)
- C Programming Language (2nd Edition), by Brian Kernighan and Dennis Ritchie, Prentice Hall