Get Start with CCS and eZdsp USB Stick

EXPERIMENT 1.1

Propose of the experiment

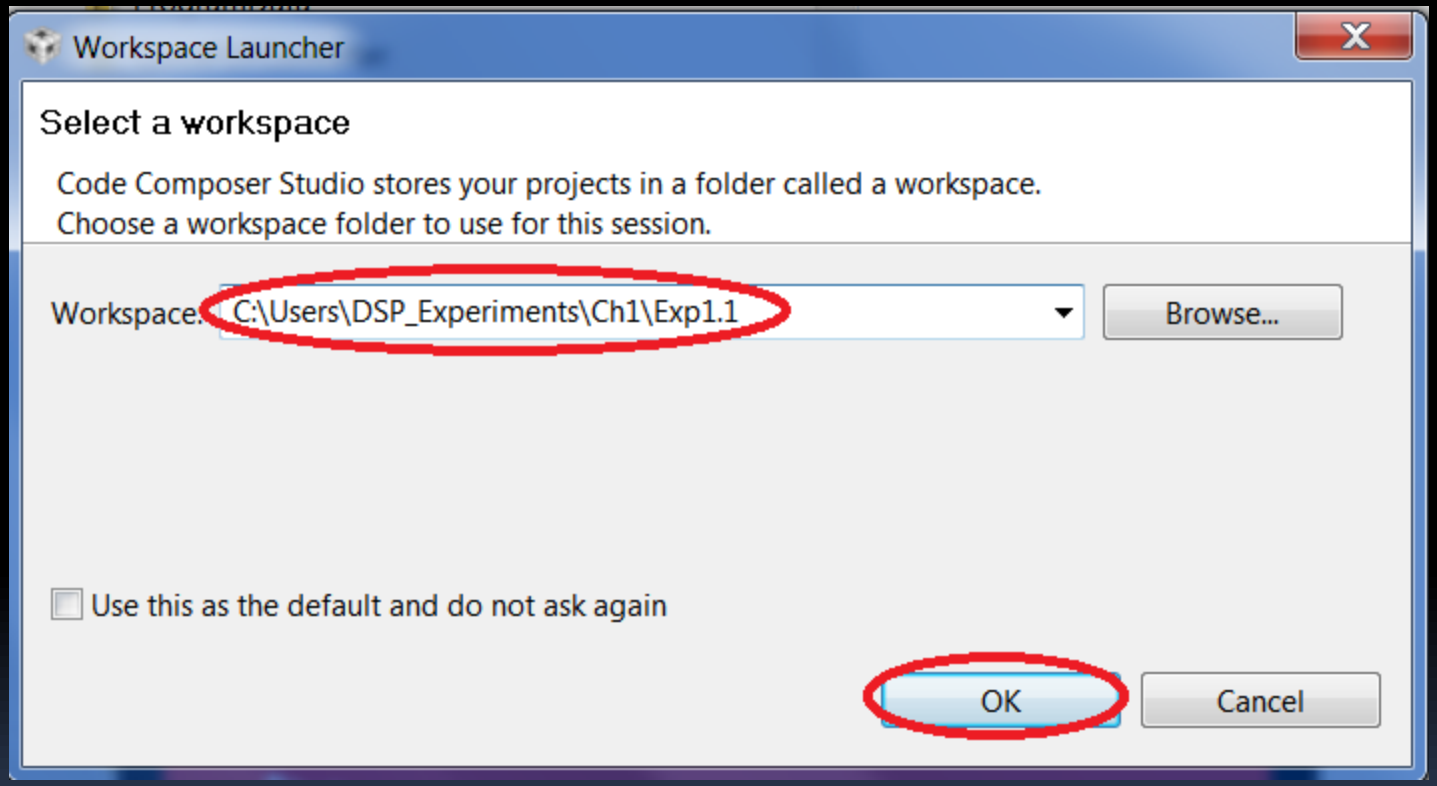
- Use CCS to create a workspace
- Create a new project
- Write a simple C program using CCS tools
- Setup CCS build environment
- Connect eZdsp USB Stick Hardware to CCS
- Set target configuration
- Build and run a C program using CCS

Start CCS

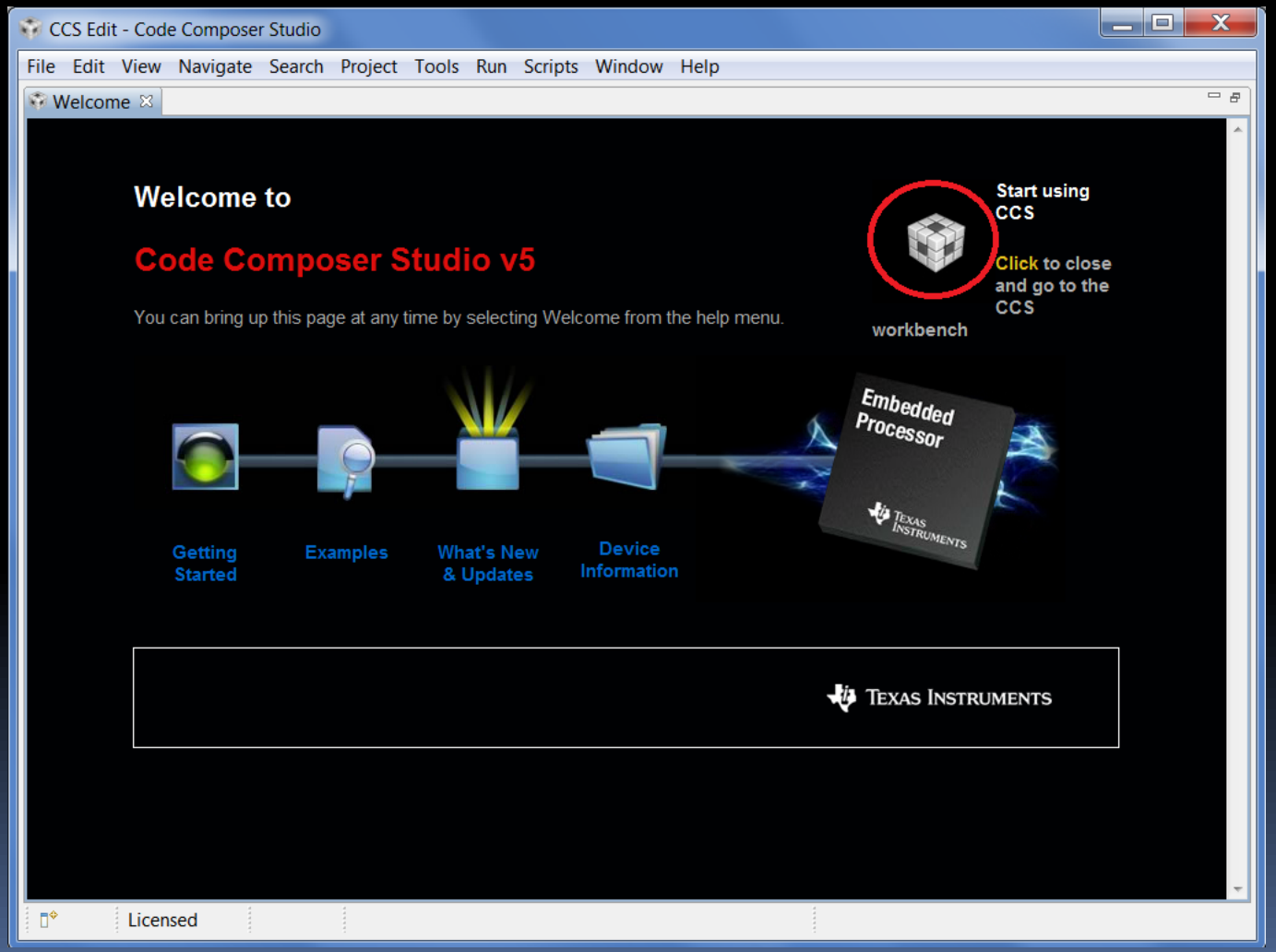
(Example: Code Composer Studio Version 5)

Create workspace

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

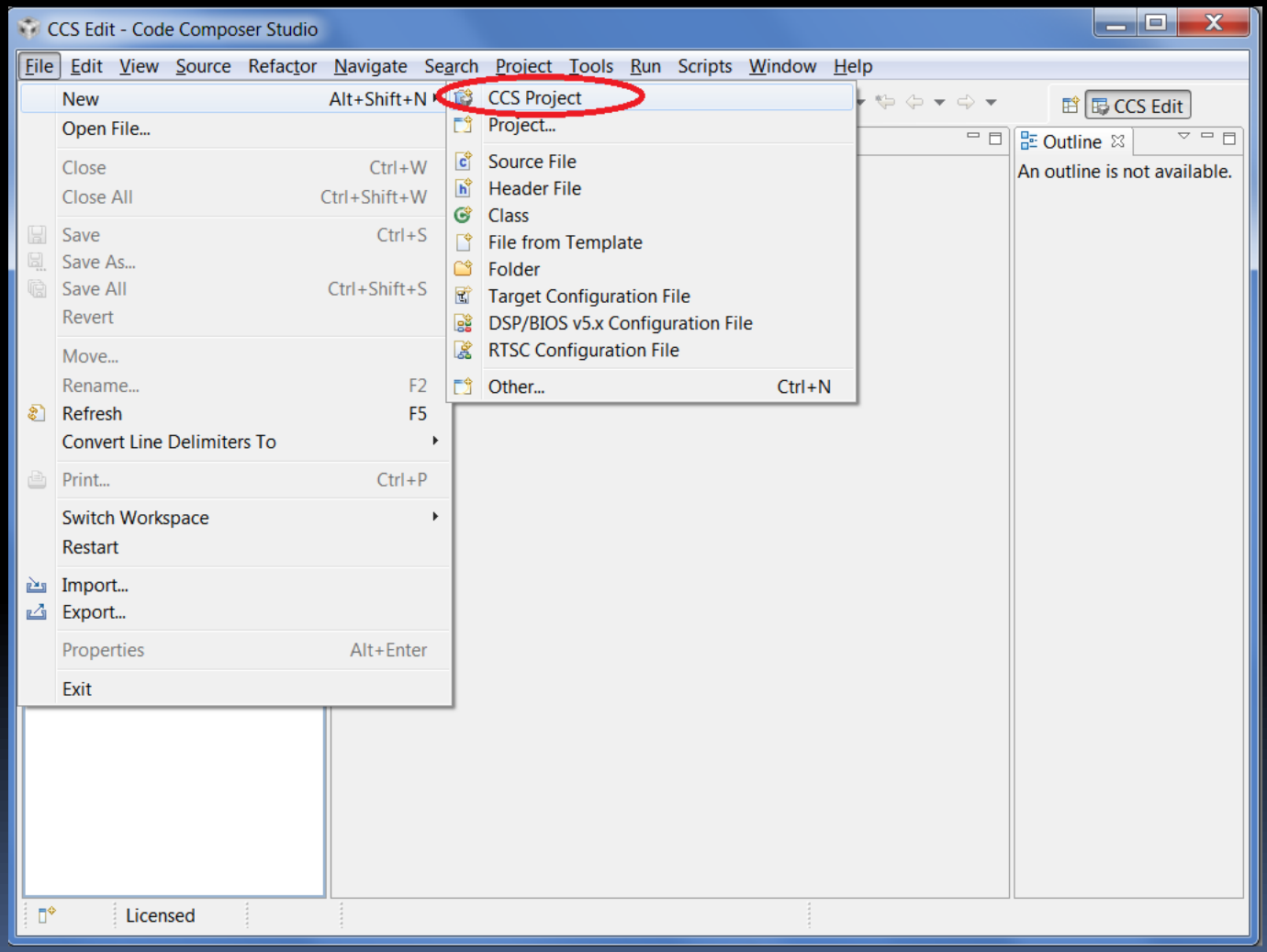


Go to CCS


CCS Edit - Code Composer Studio
File Edit View Navigate Search Project Tools Run Scripts Window Help
Welcome
Welcome to
Code Composer Studio v5
You can bring up this page at any time by selecting Welcome from the help menu.
Start using CCS
Click to close and go to the CCS workbench
Embedded Processor
Texas Instruments
Getting Started
Examples
What's New & Updates
Device Information
Texas Instruments
Licensed

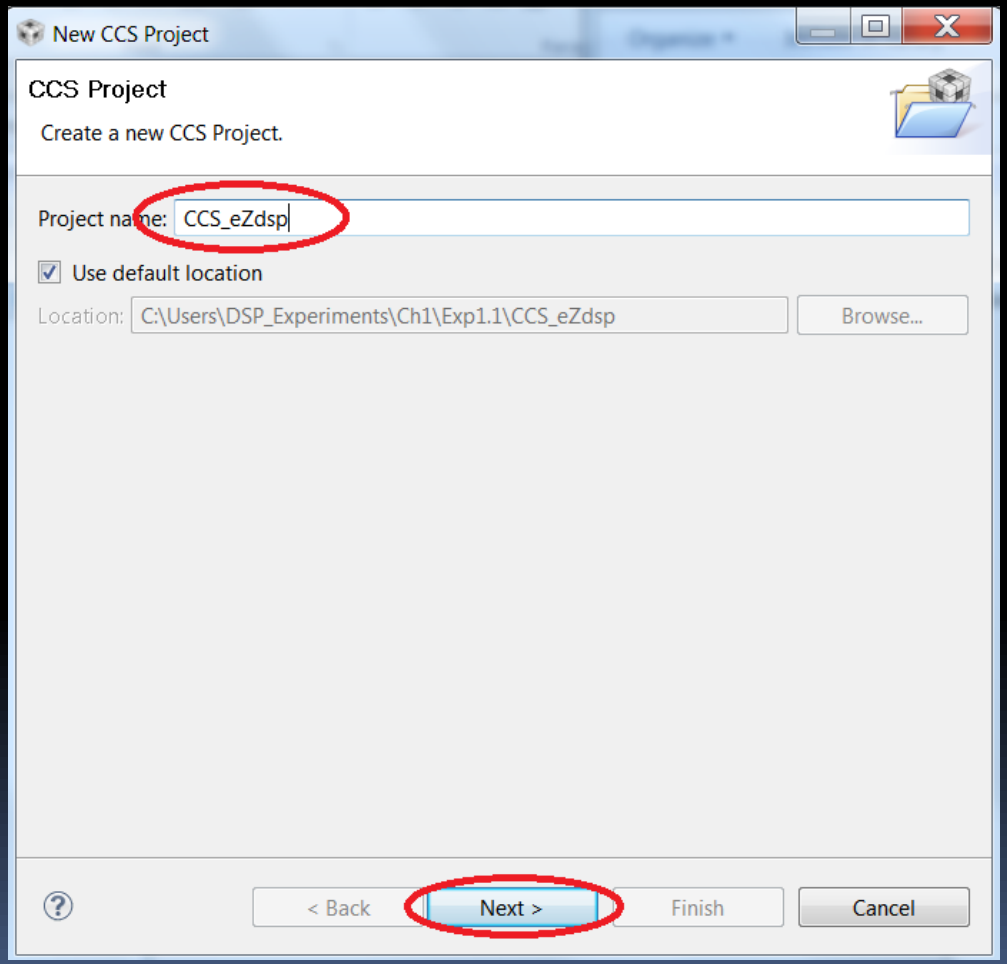
Create a new project

(File -> New -> CCS Project)

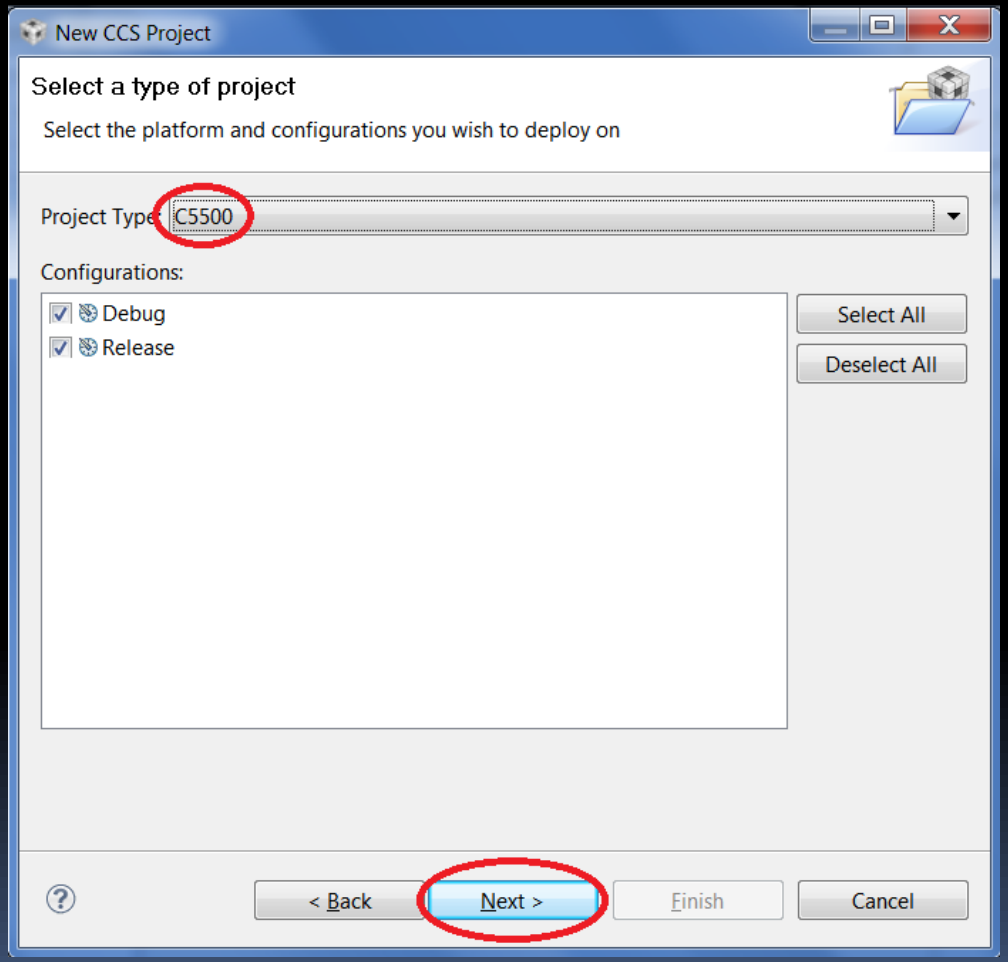


Create a new project name

(Example: CCS_eZdsp)

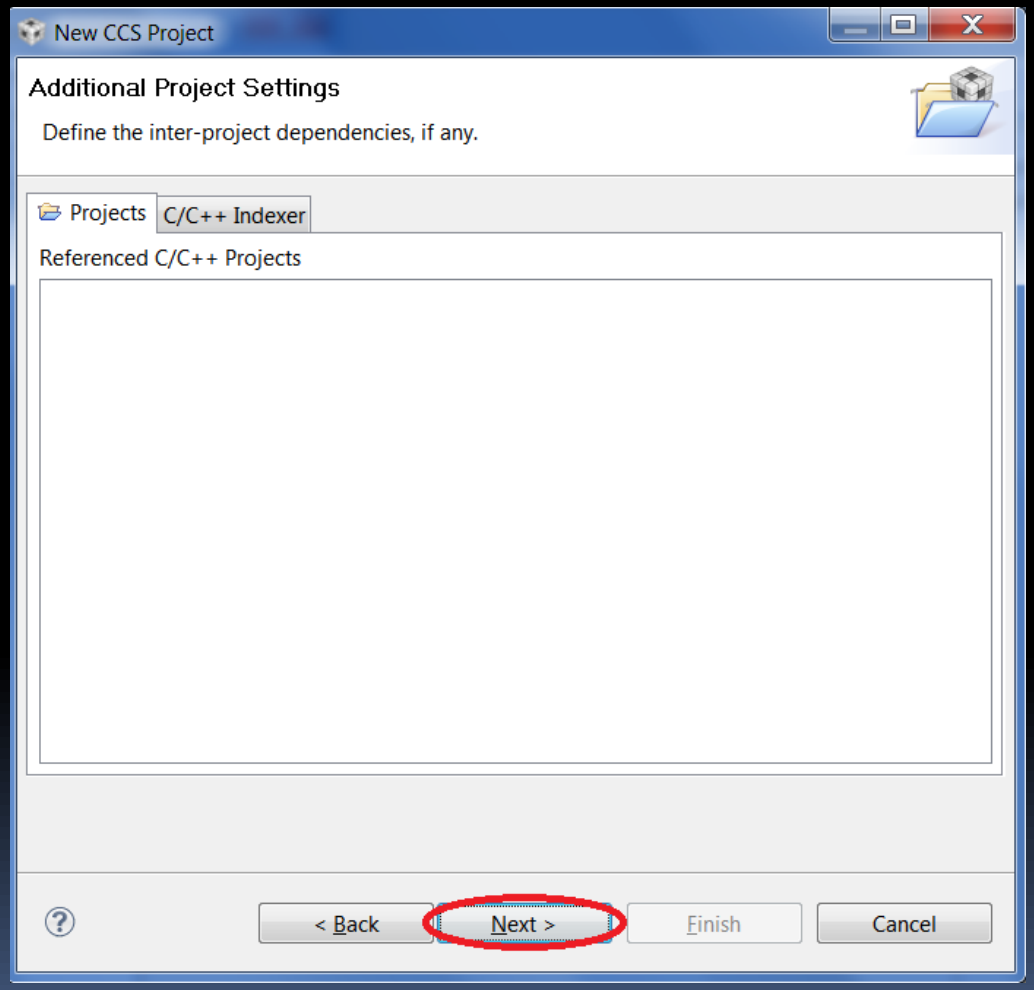


Select C5500 as the new project



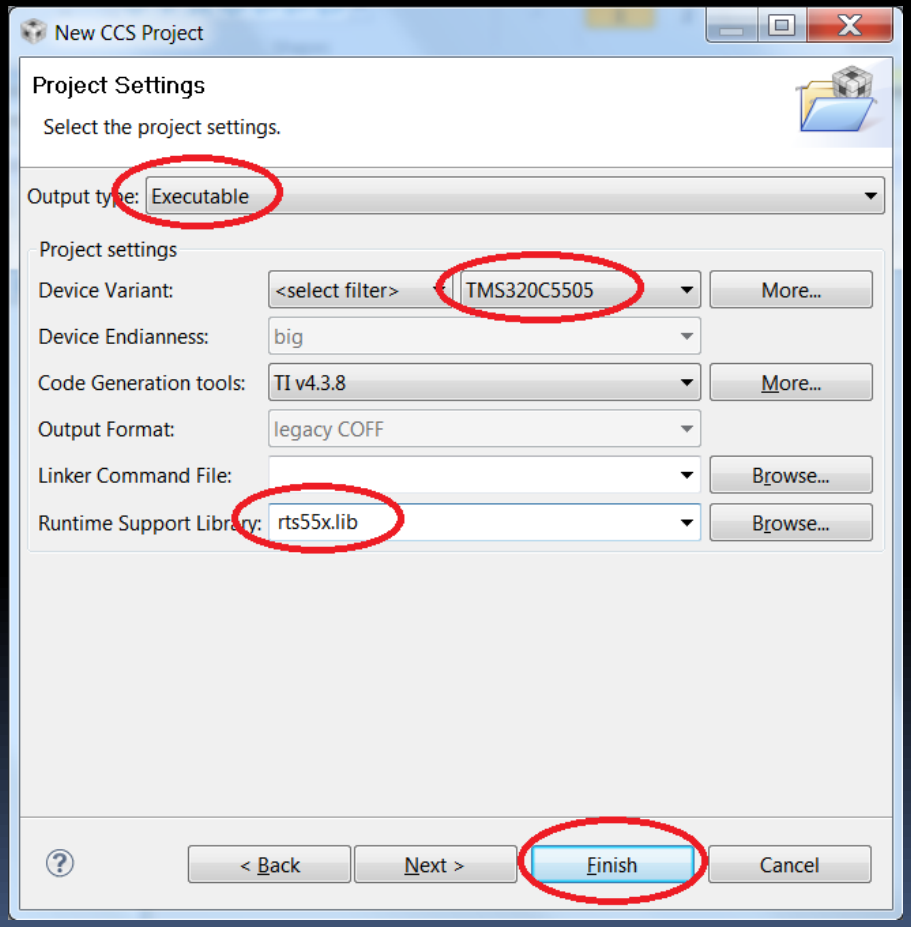
Go to next

(We do not need any additional settings)

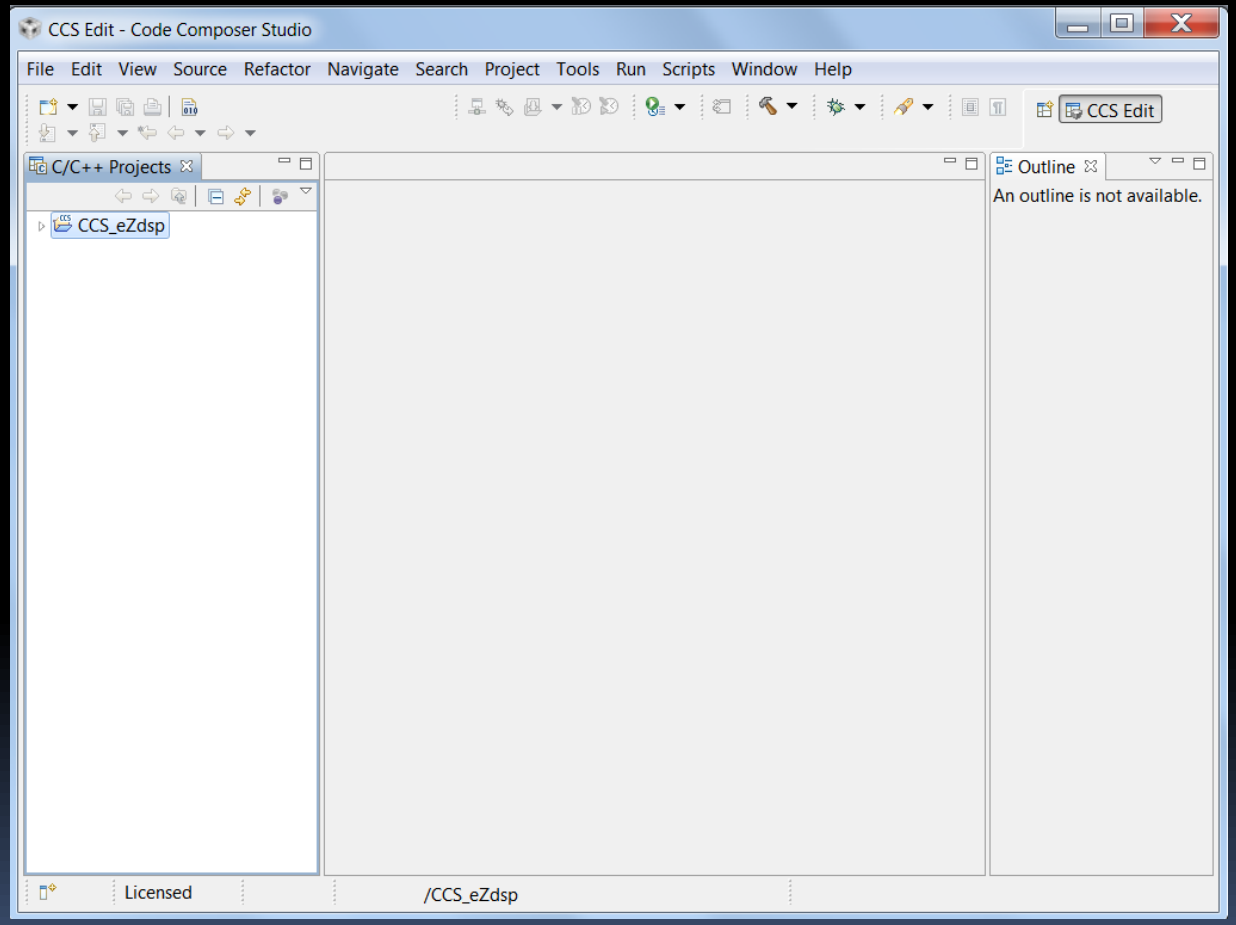


Set up the project

(Executable output, Device C5505, Library rts55h.lib)

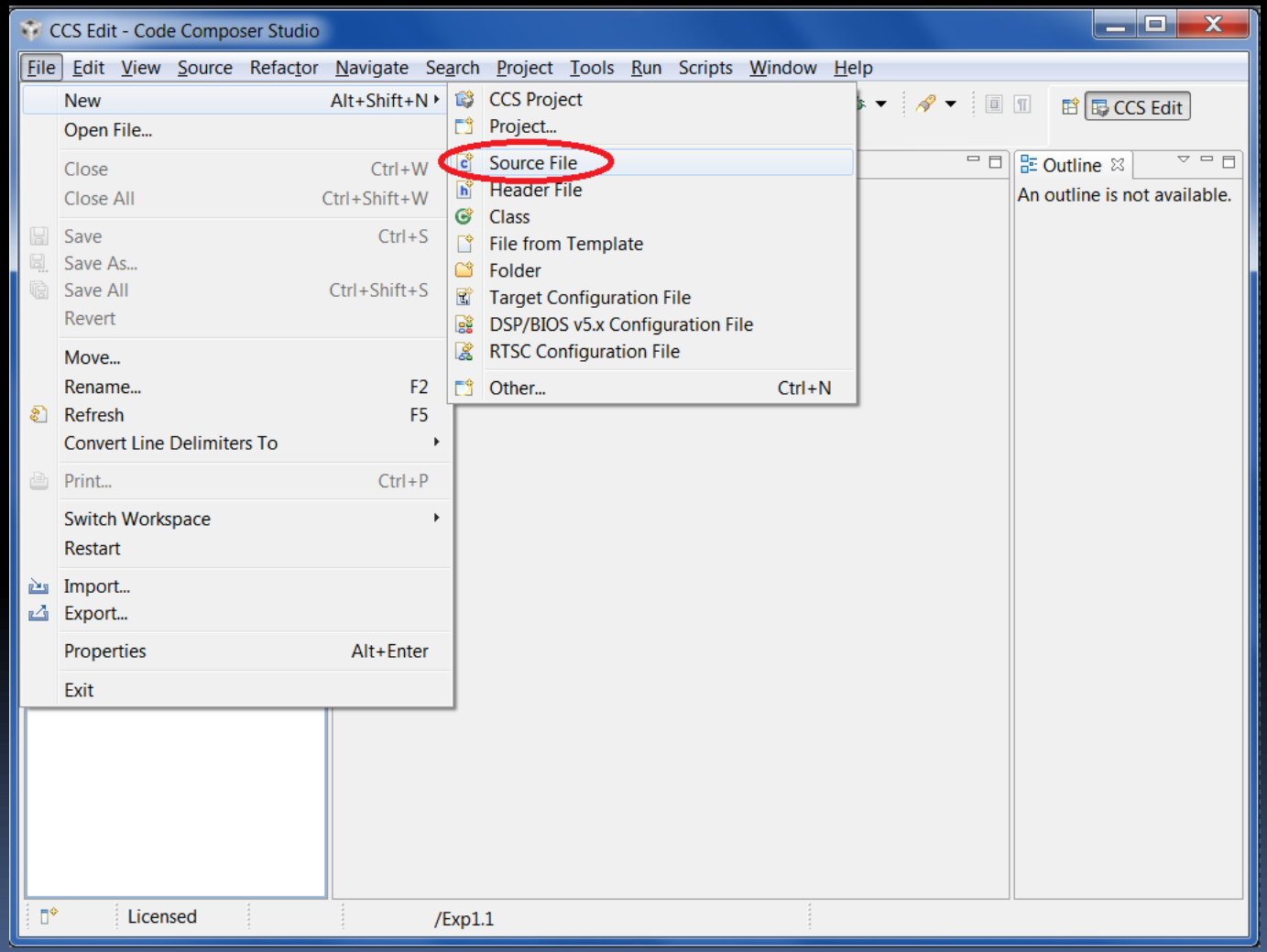


Project: CCS_eZdsp

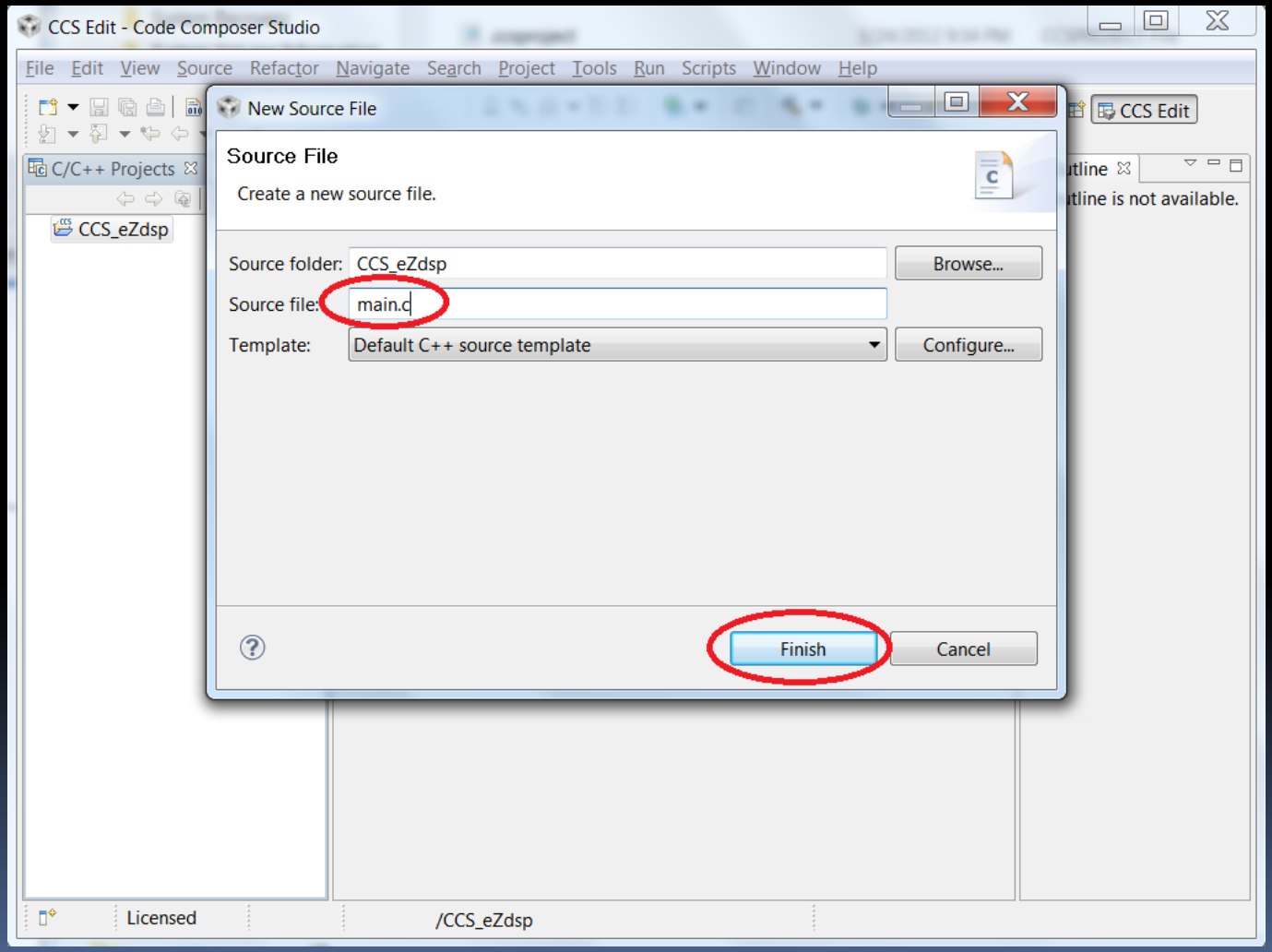


Add Source File

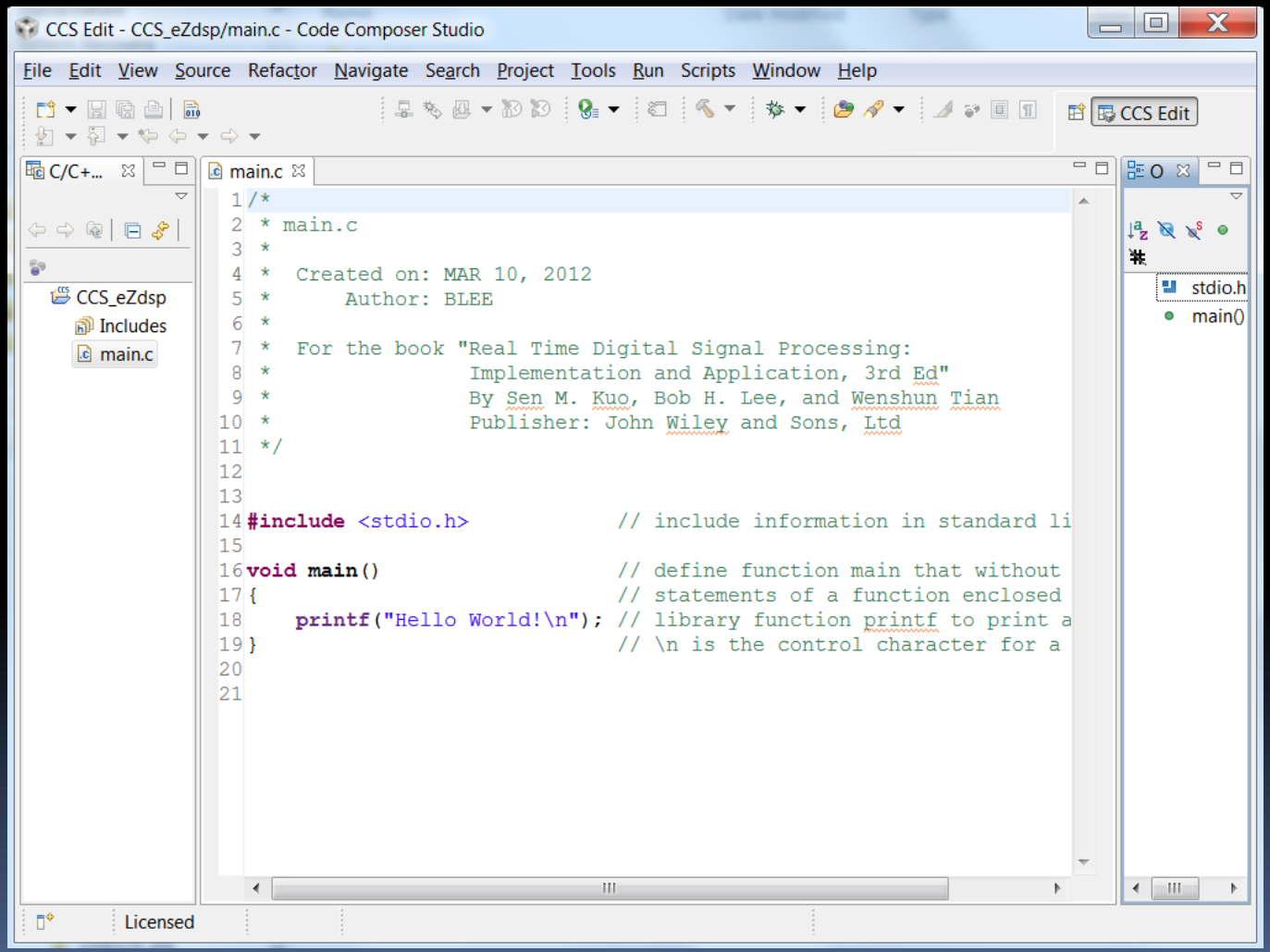
(Example: File->New->Source File)



Create C file main.c (1)

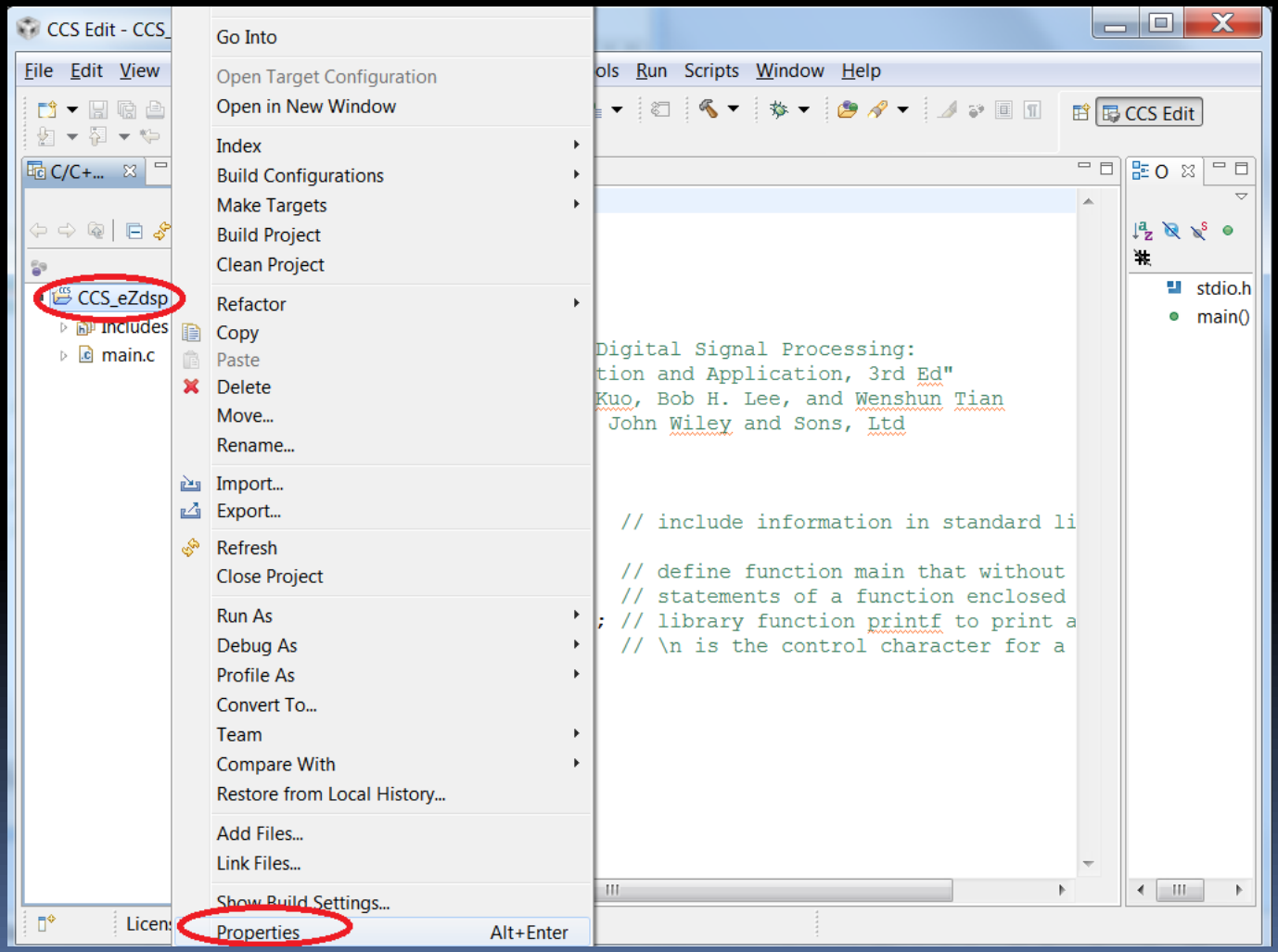


Create C file main.c (2)



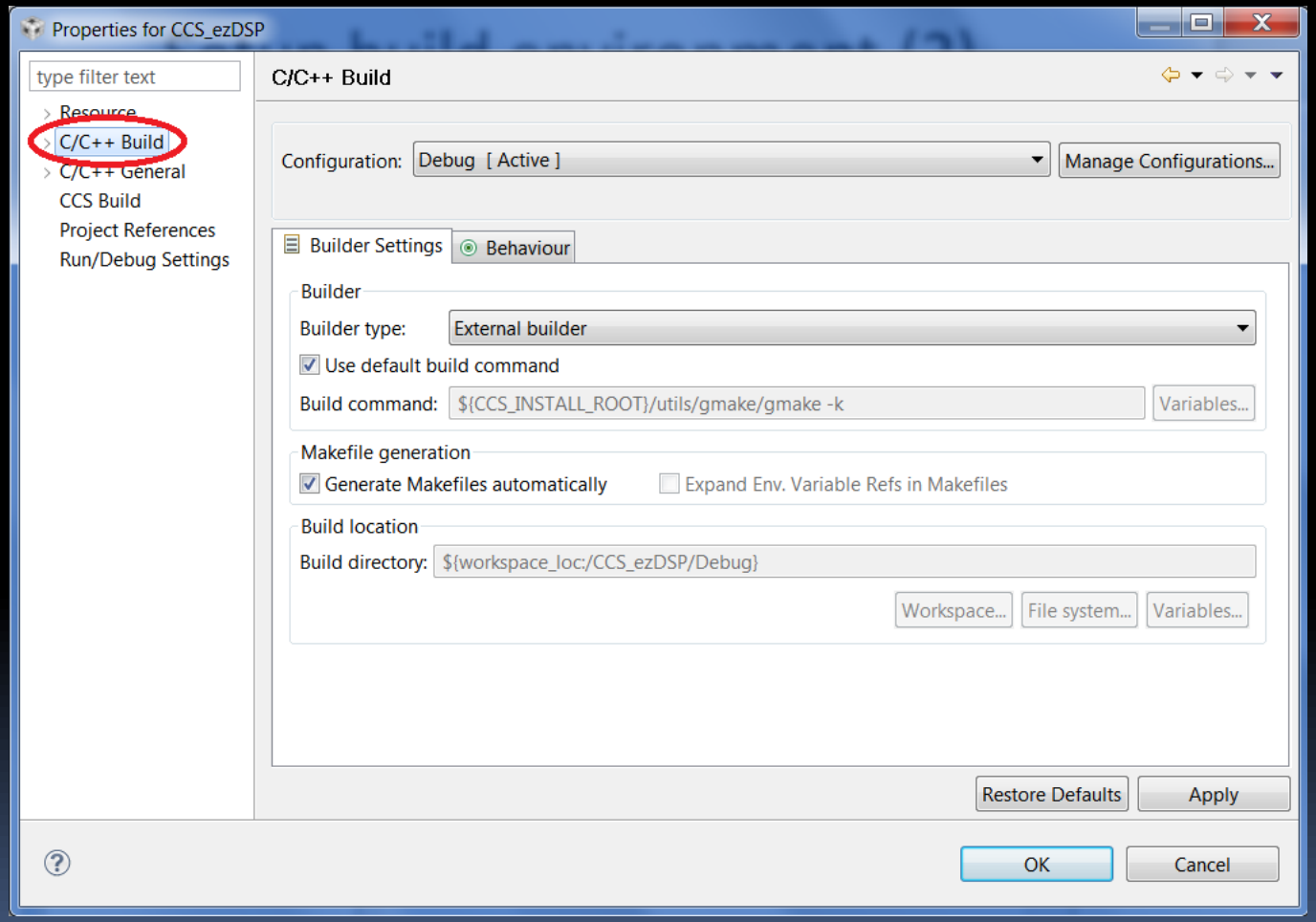
Setup build environment (1)

(Right click on CCS_eZdsp then select Property)



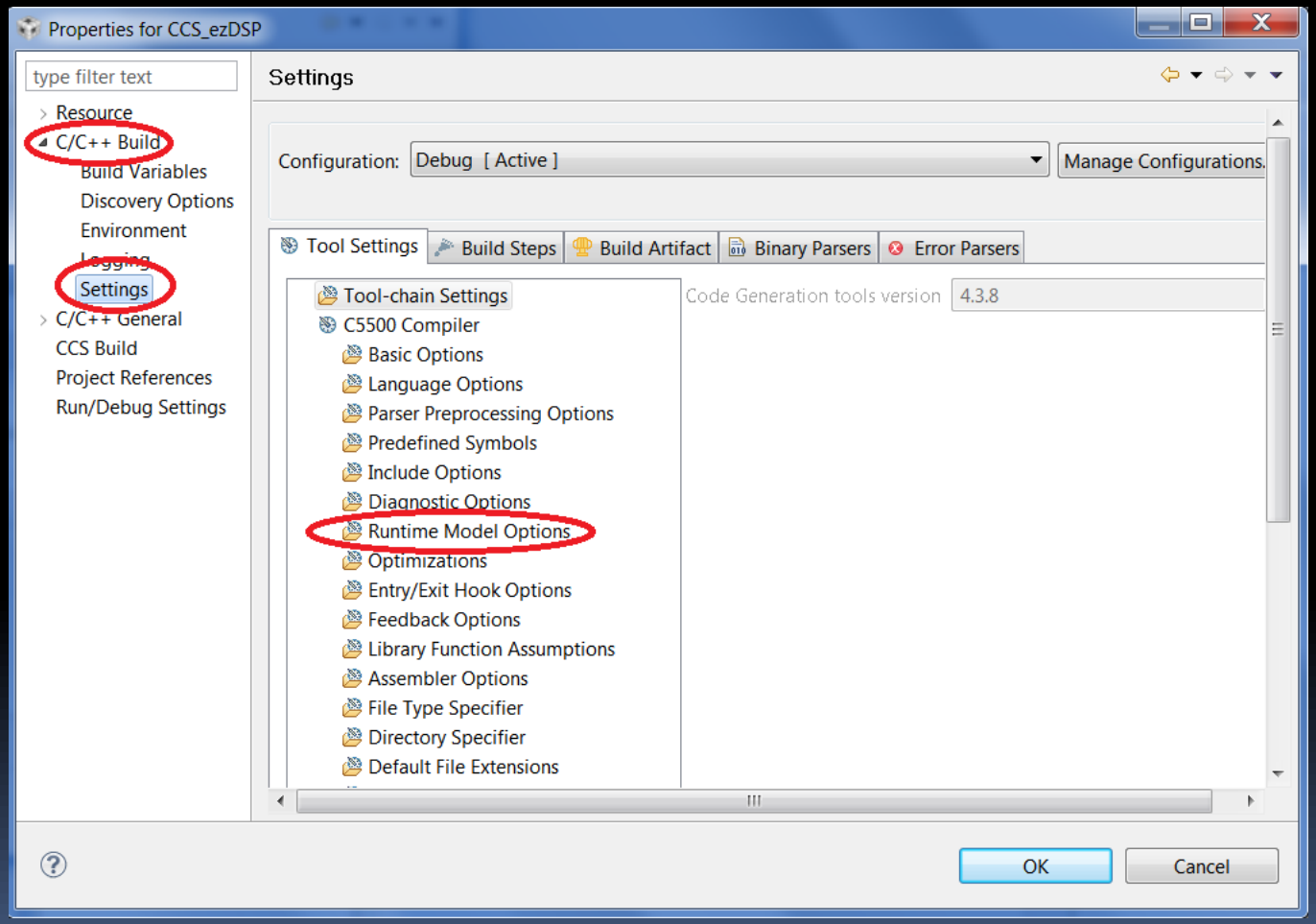
Setup build environment (2)

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



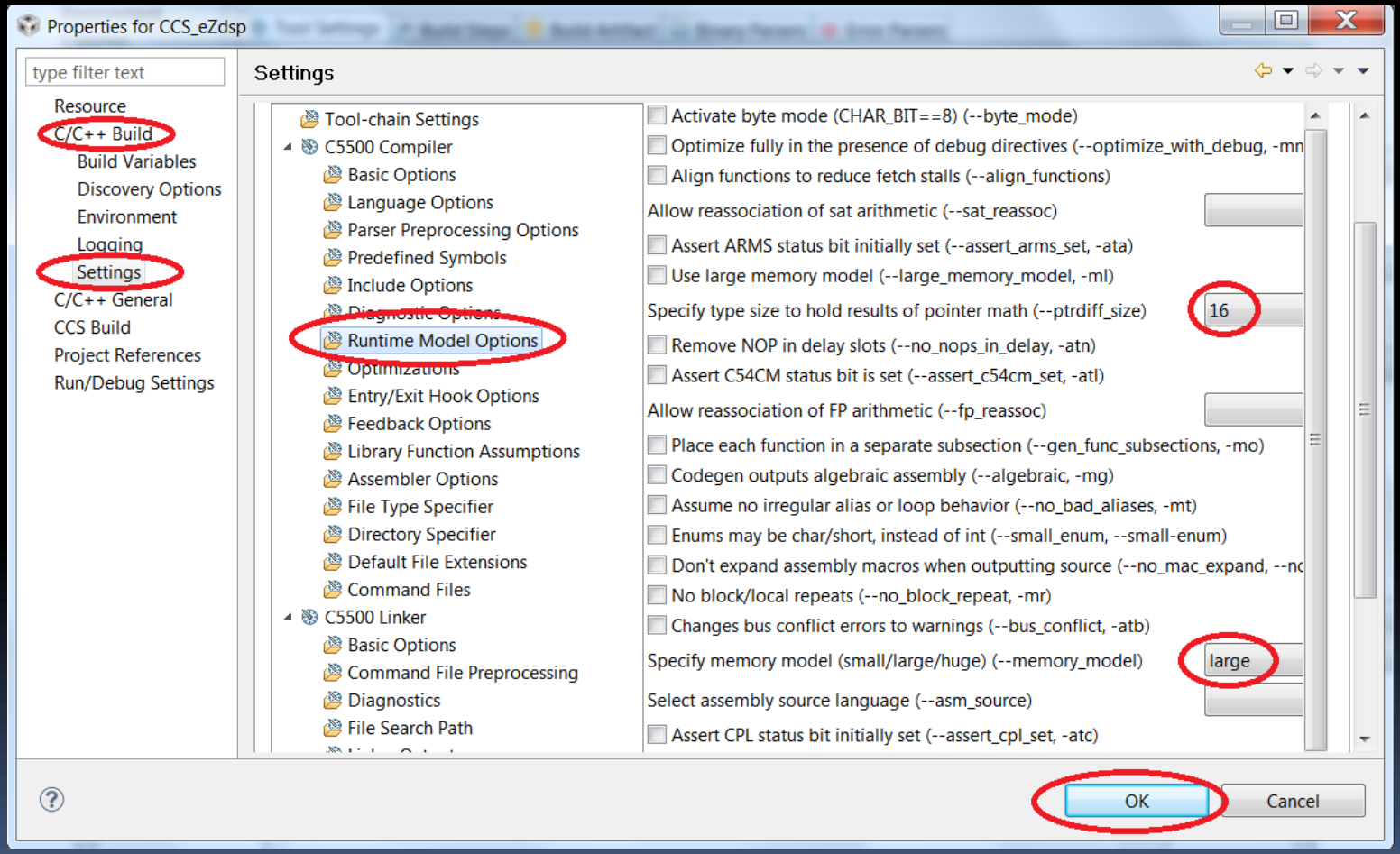
Setup build environment (3)

(Select Settings, then Runtime Options)



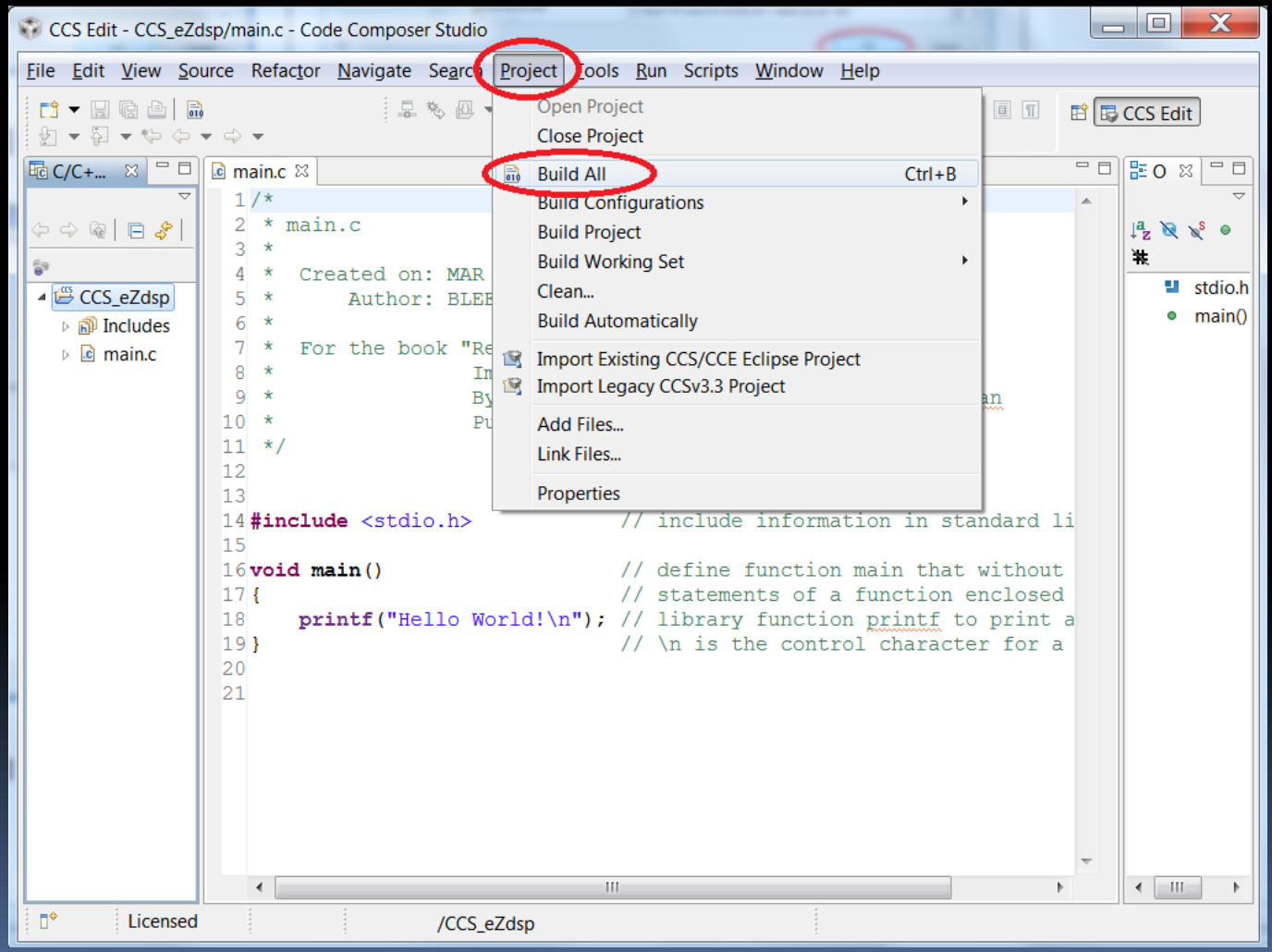
Setup build environment (4)

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



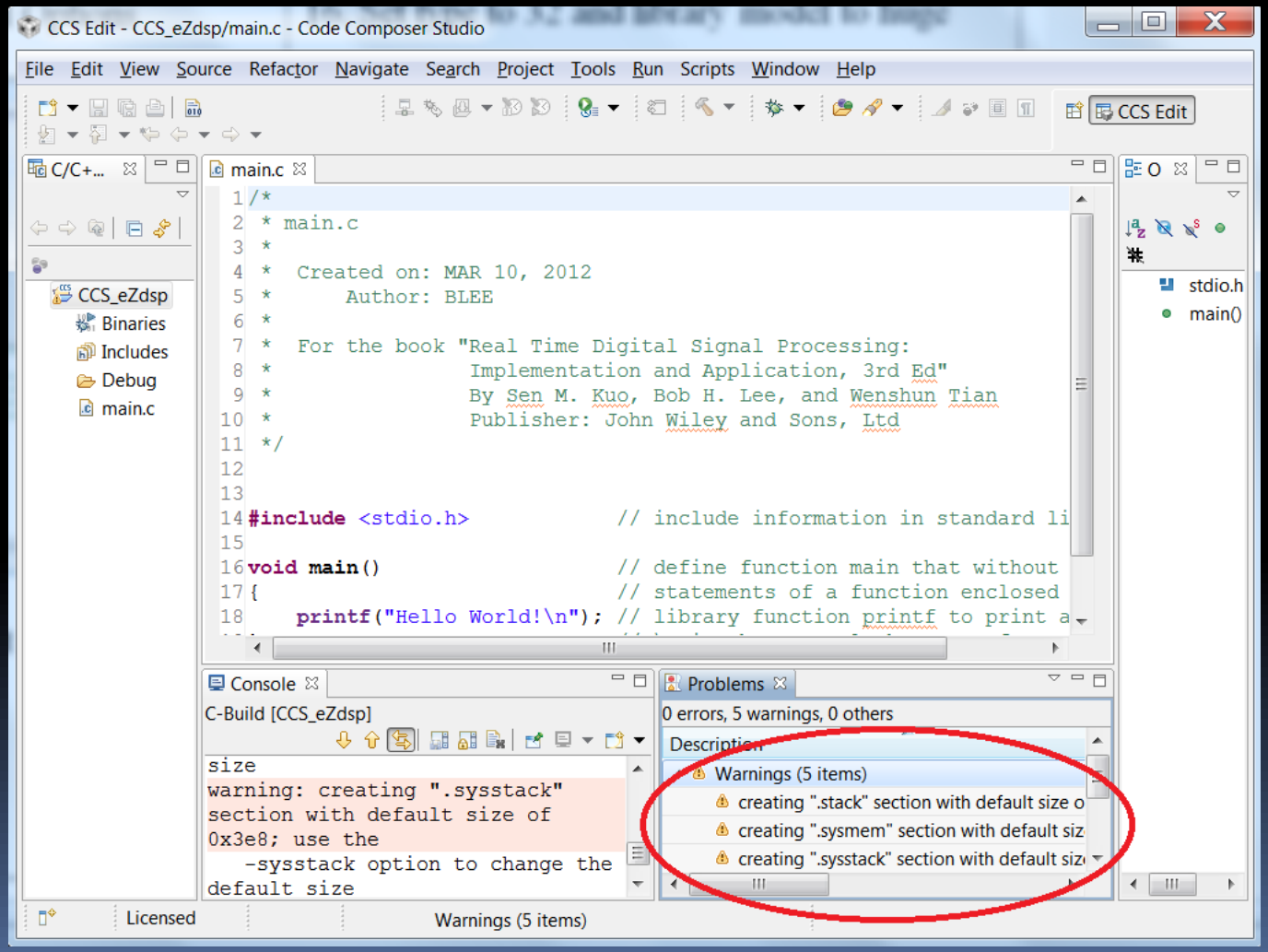
Build the project

(Project->Build All)



Build result

(No errors, but there are 5 warnings)



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, lnk.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 lnk.cmd file from the software came with the book.

Create linker command file

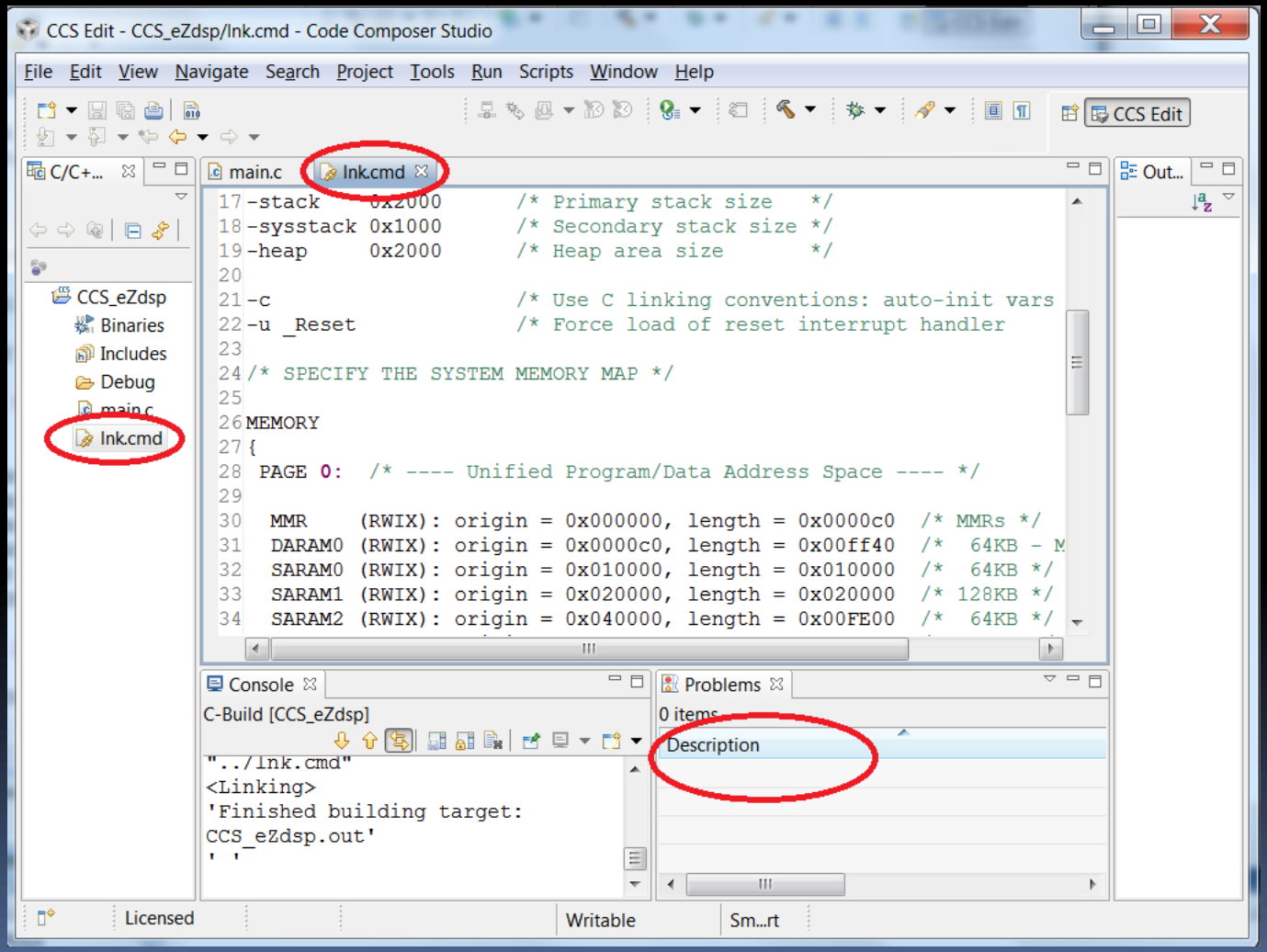
(Using a text editor create lnk.com as below)

```
/*************************************************************************/

/* 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 = 00000c0h length = 0off40h  /* 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
   .sysmem : > 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
}
```

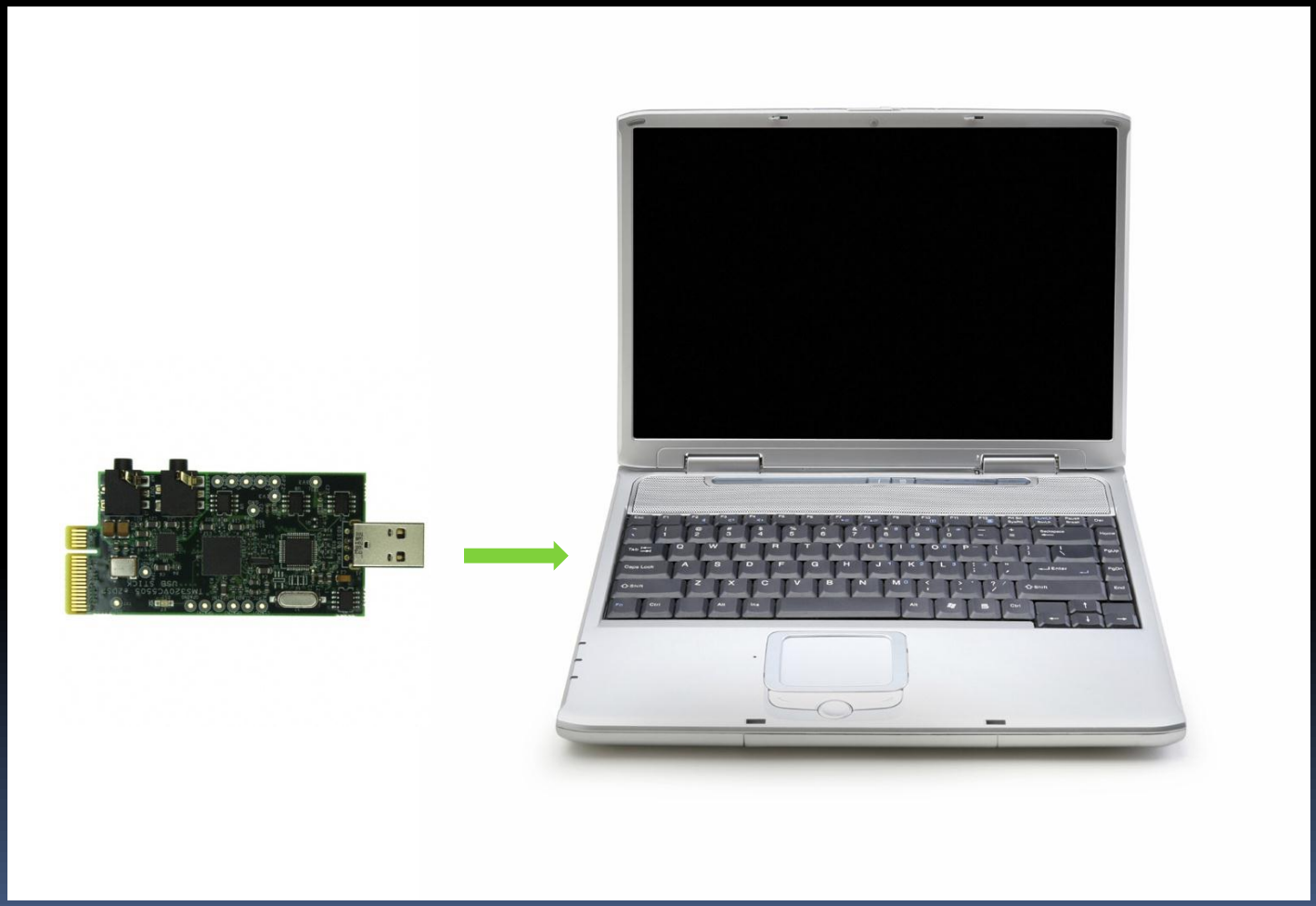
Once lnk.cmd file is copied

(Rebuild the project and there are no warnings)



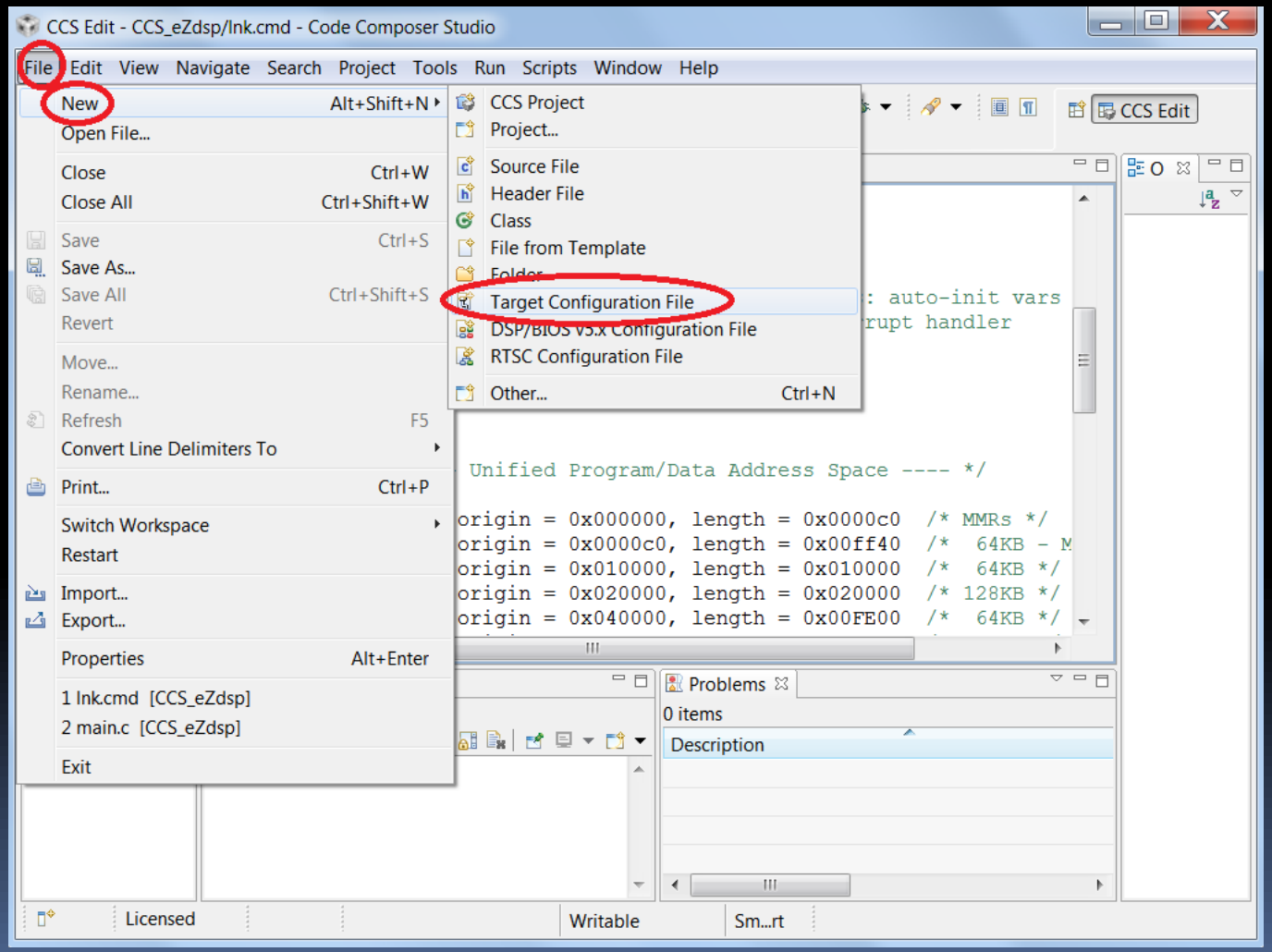
Connect eZdsp to PC

(Refer to eZdsp Start Guide for settings)



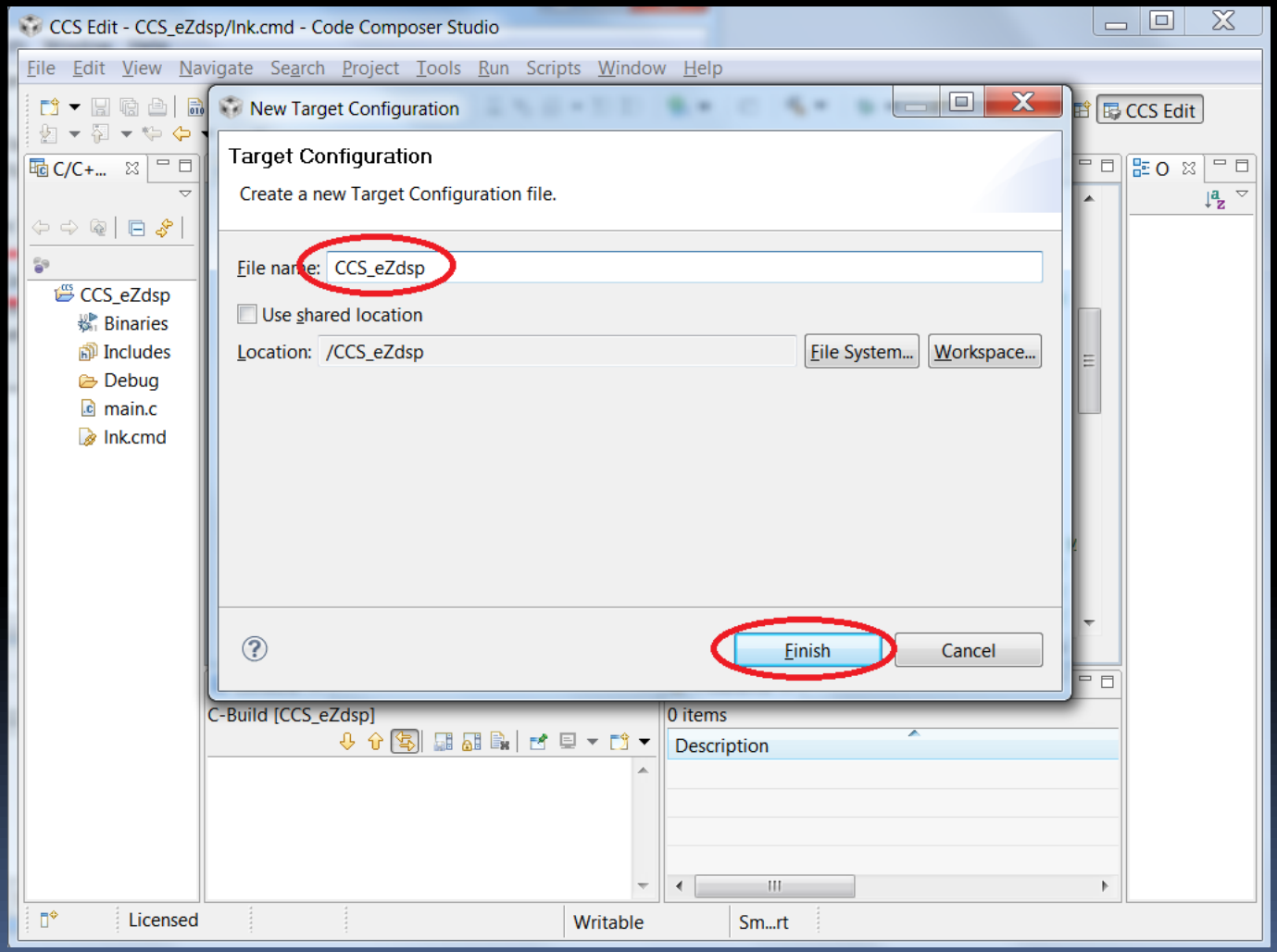
Set target configuration (1)

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



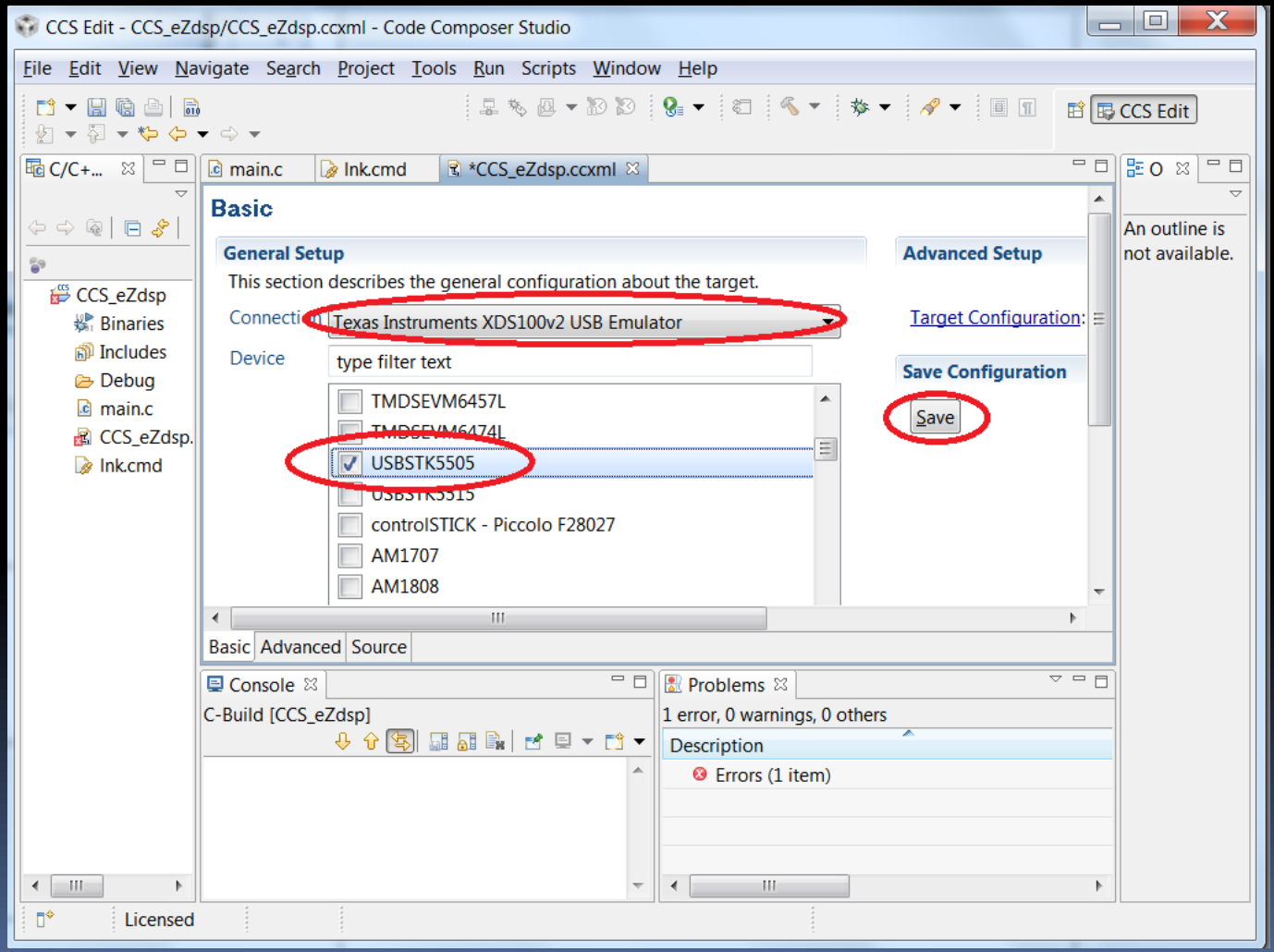
Set target configuration (2)

(Create a target configuration file name)



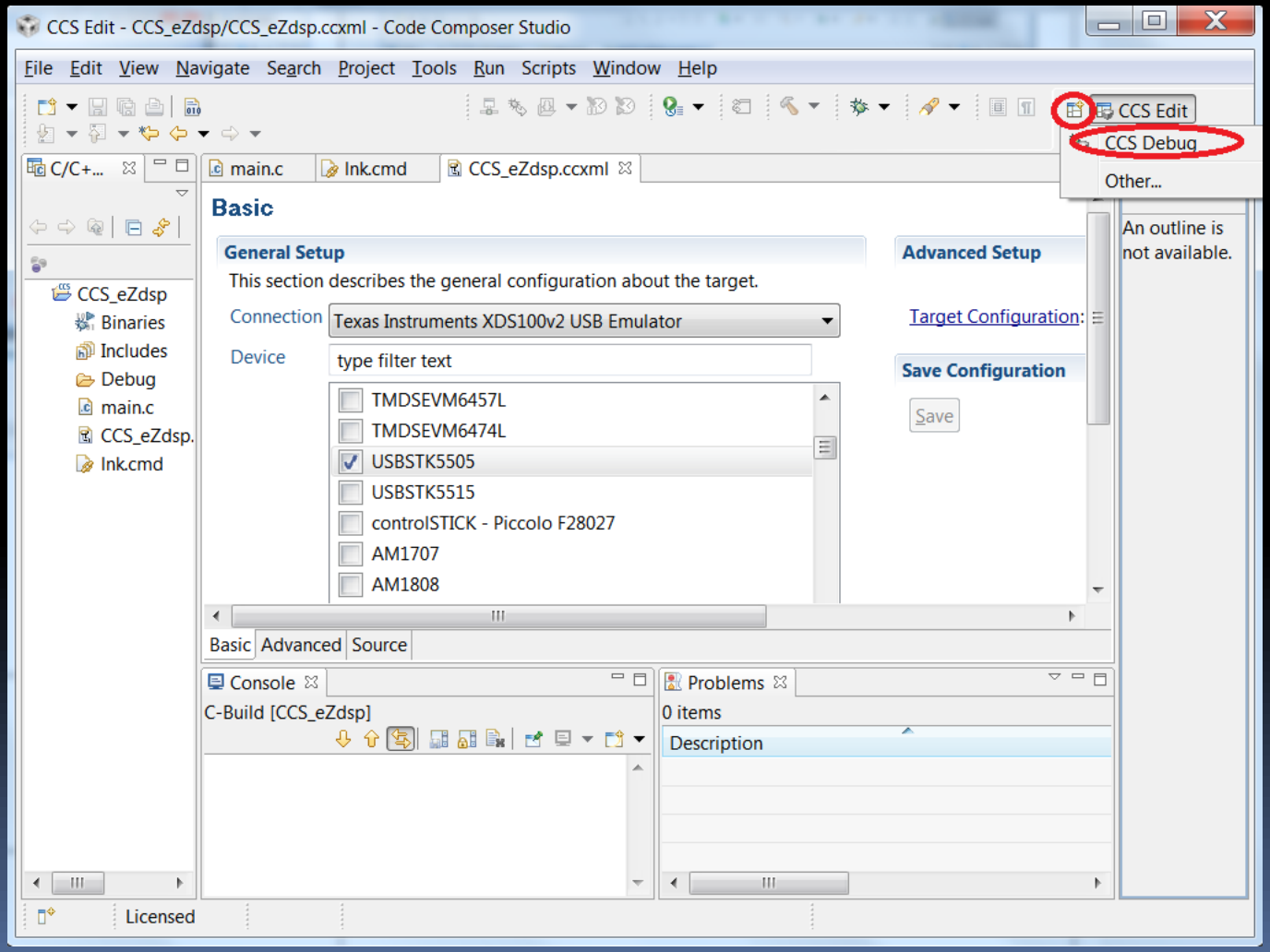
Set target configuration (3)

(Select XDS100v2 USB Emulator, USBSTK5505, & Save)



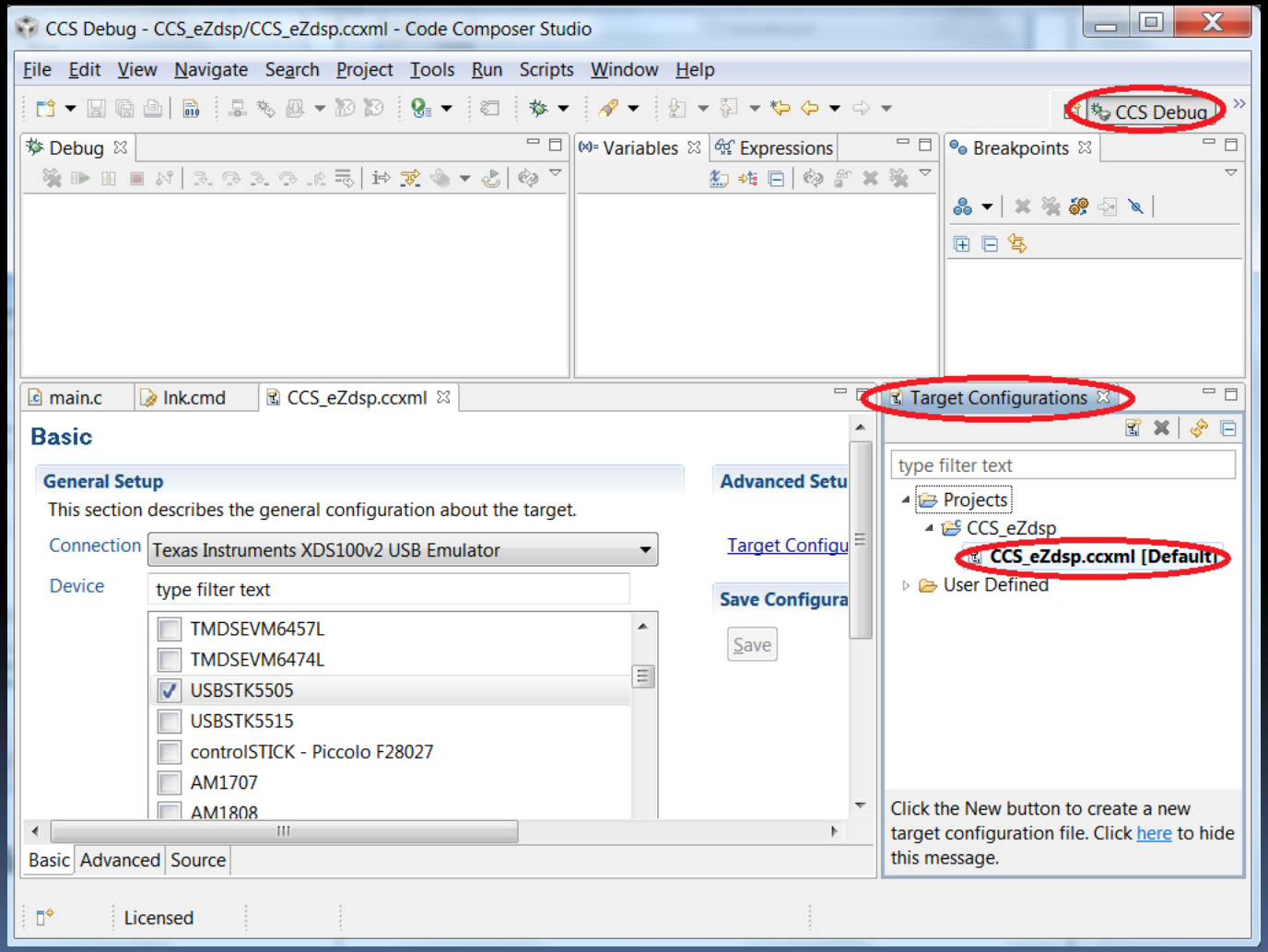
Change edit mode to debug mode

(Change from CCS Edit to CCS Debug Mode)



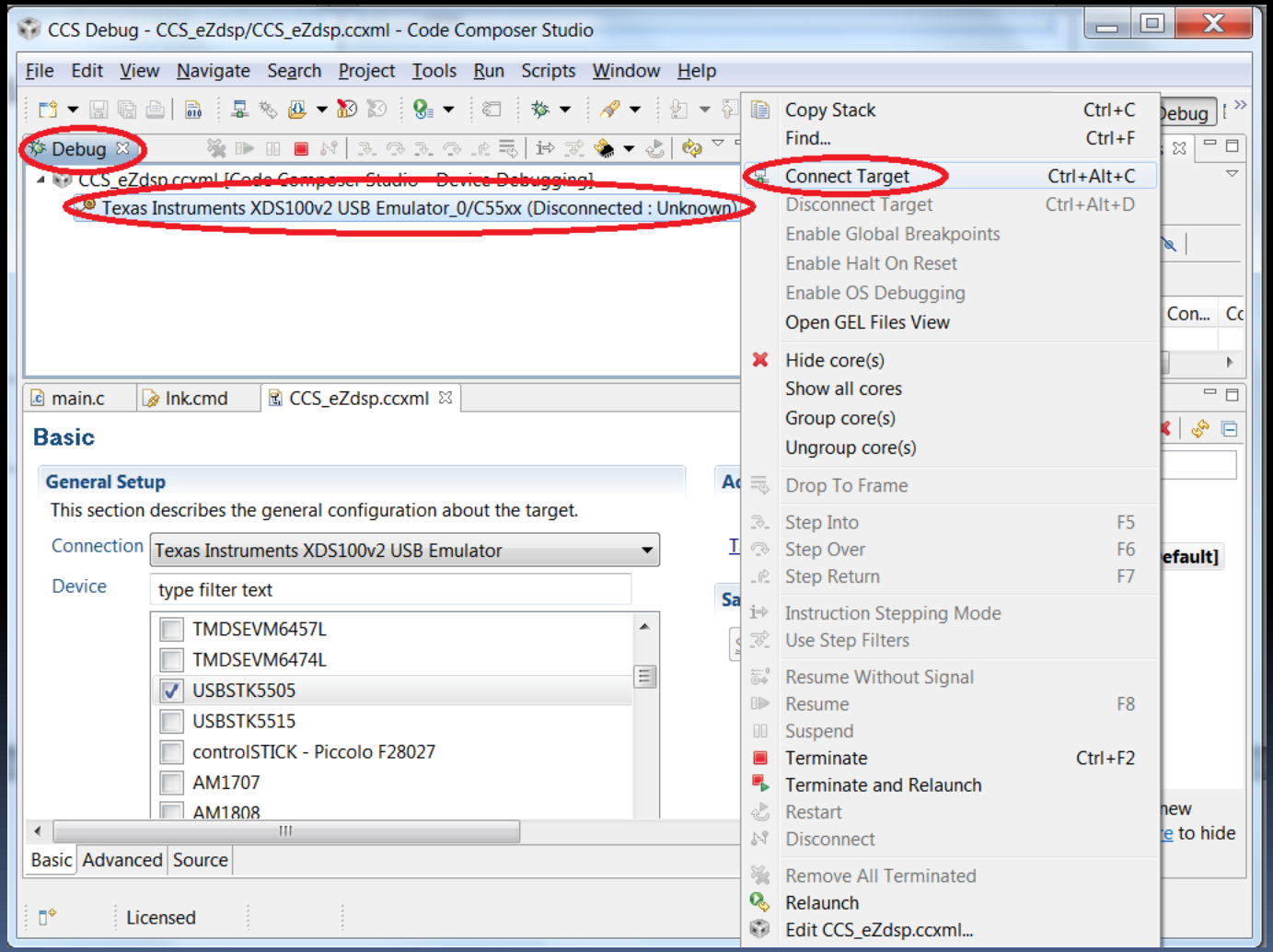
Launch selected configuration

(Target Configuration->Project->right click on CCS_eZdsp.ccxml)



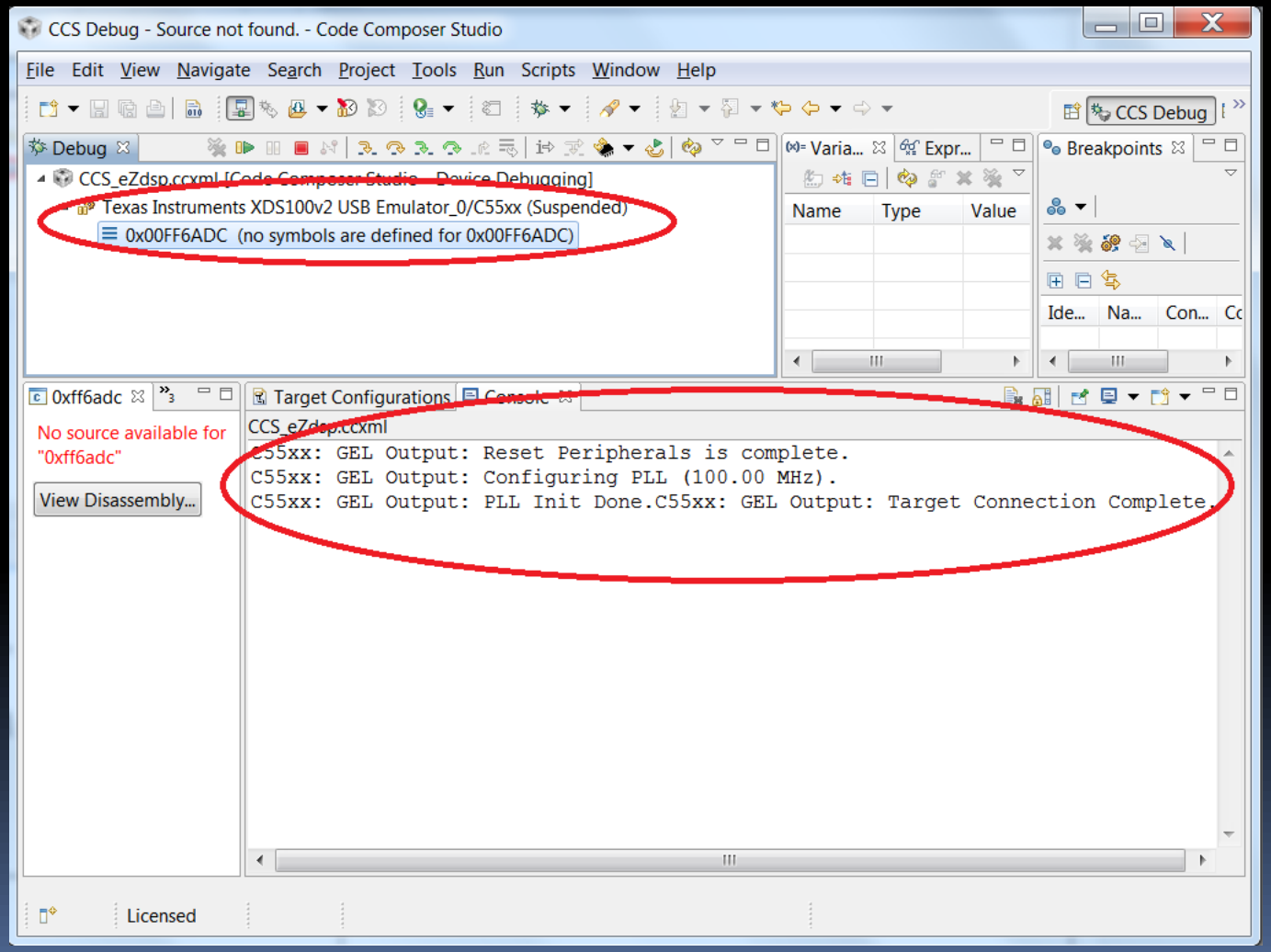
Connect eZdsp

(Right click on USB_Emulator_0, then Connect Tart)



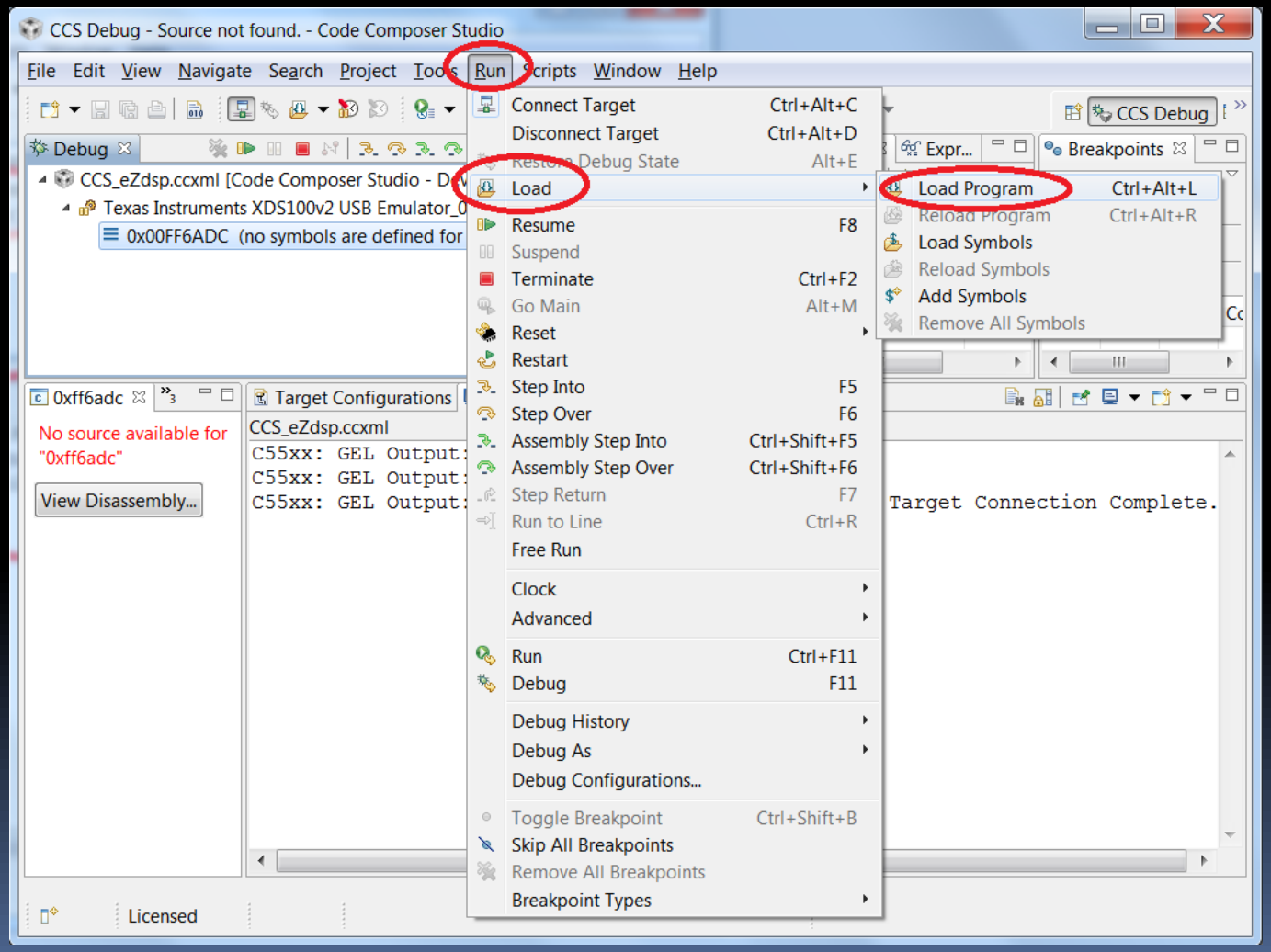
eZdsp connected

(Target reset and configured automatically)



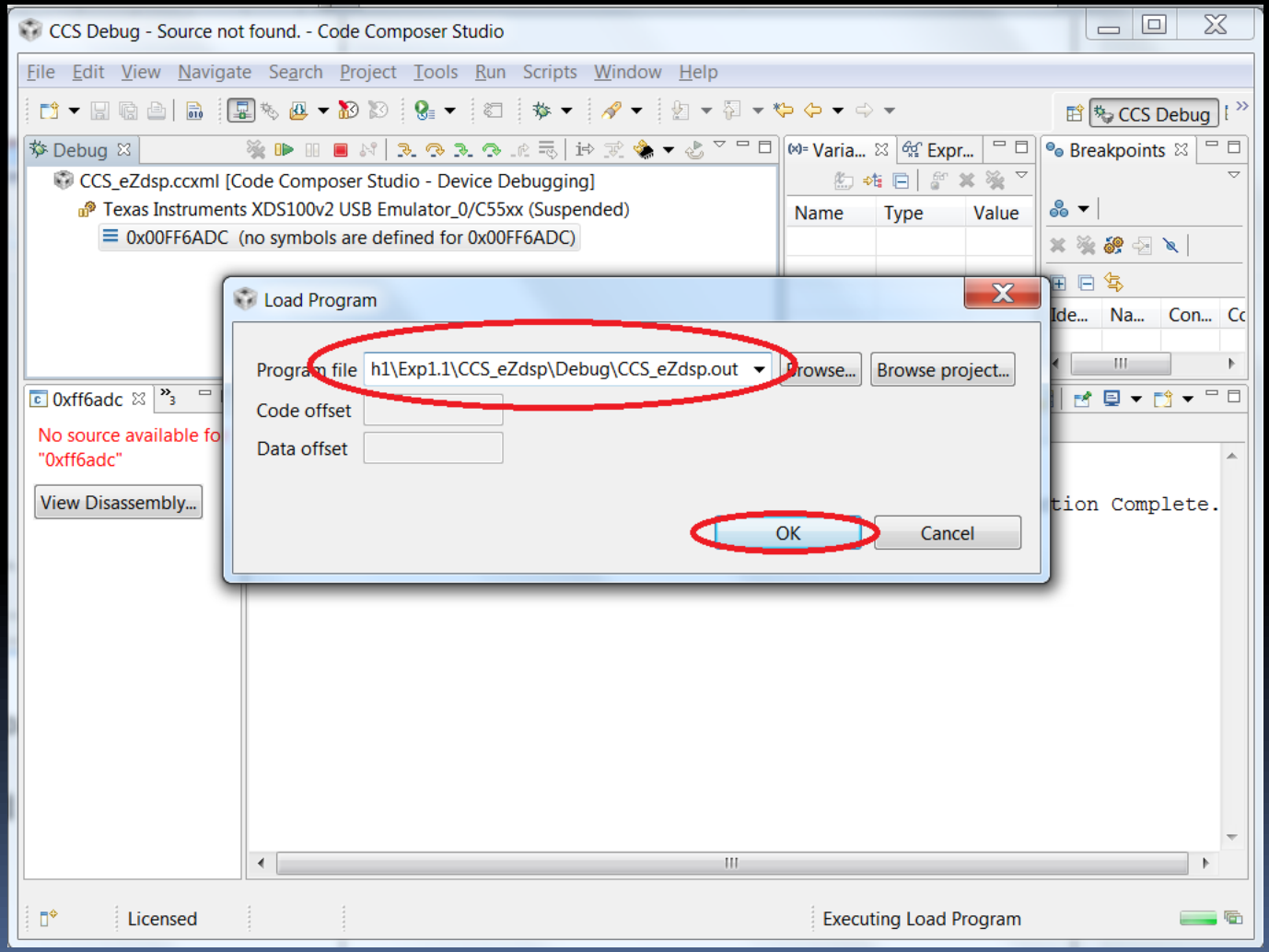
Load program CCS_eZdsp.out (1)

(From Run->Load->Load Program)



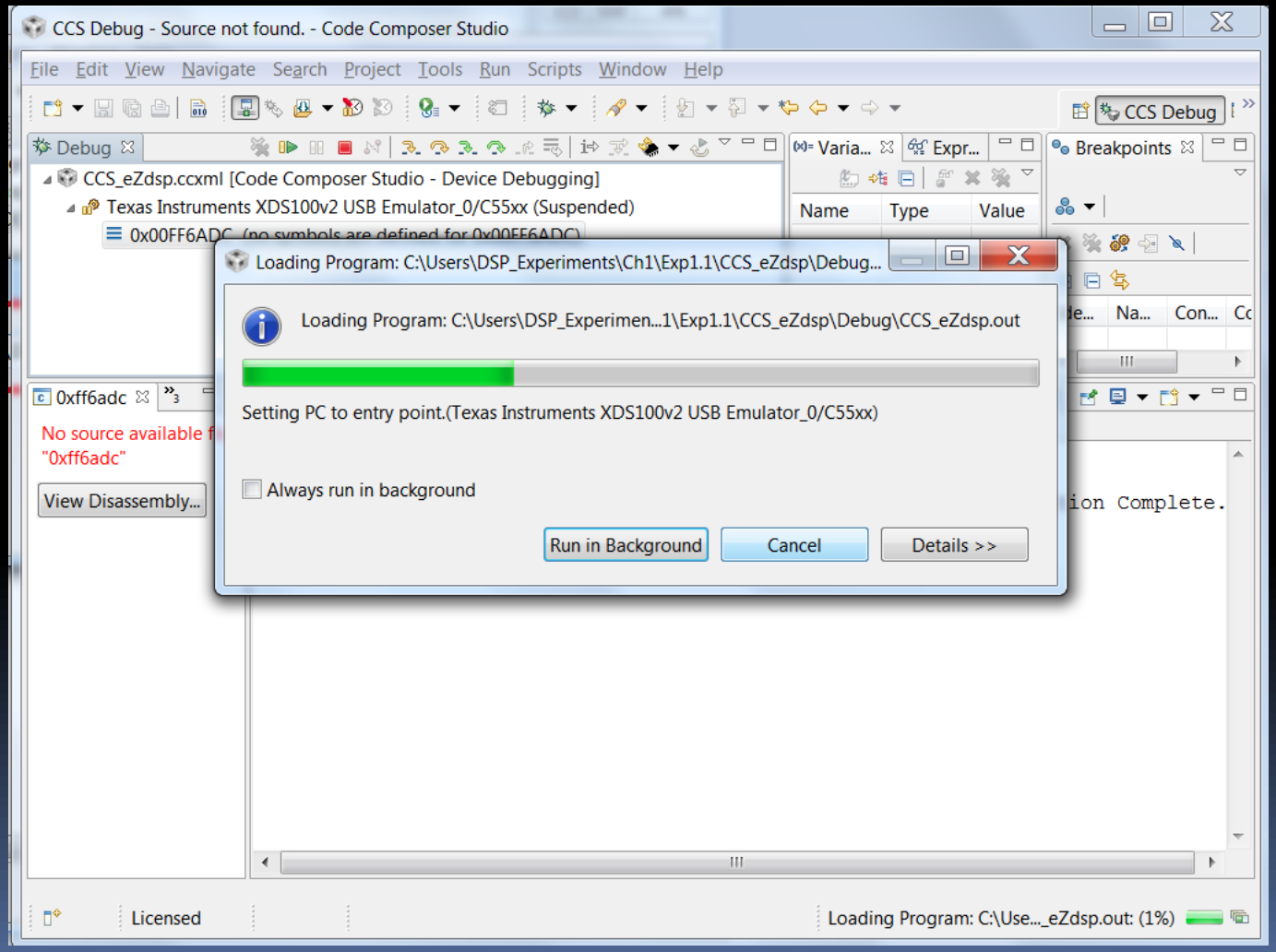
Load program CCS_eZdsp.out (2)

(…\DSP_Experiments\Ch1\Exp1.1\CS_eZdsp\DebugCCS_eZdsp.out)



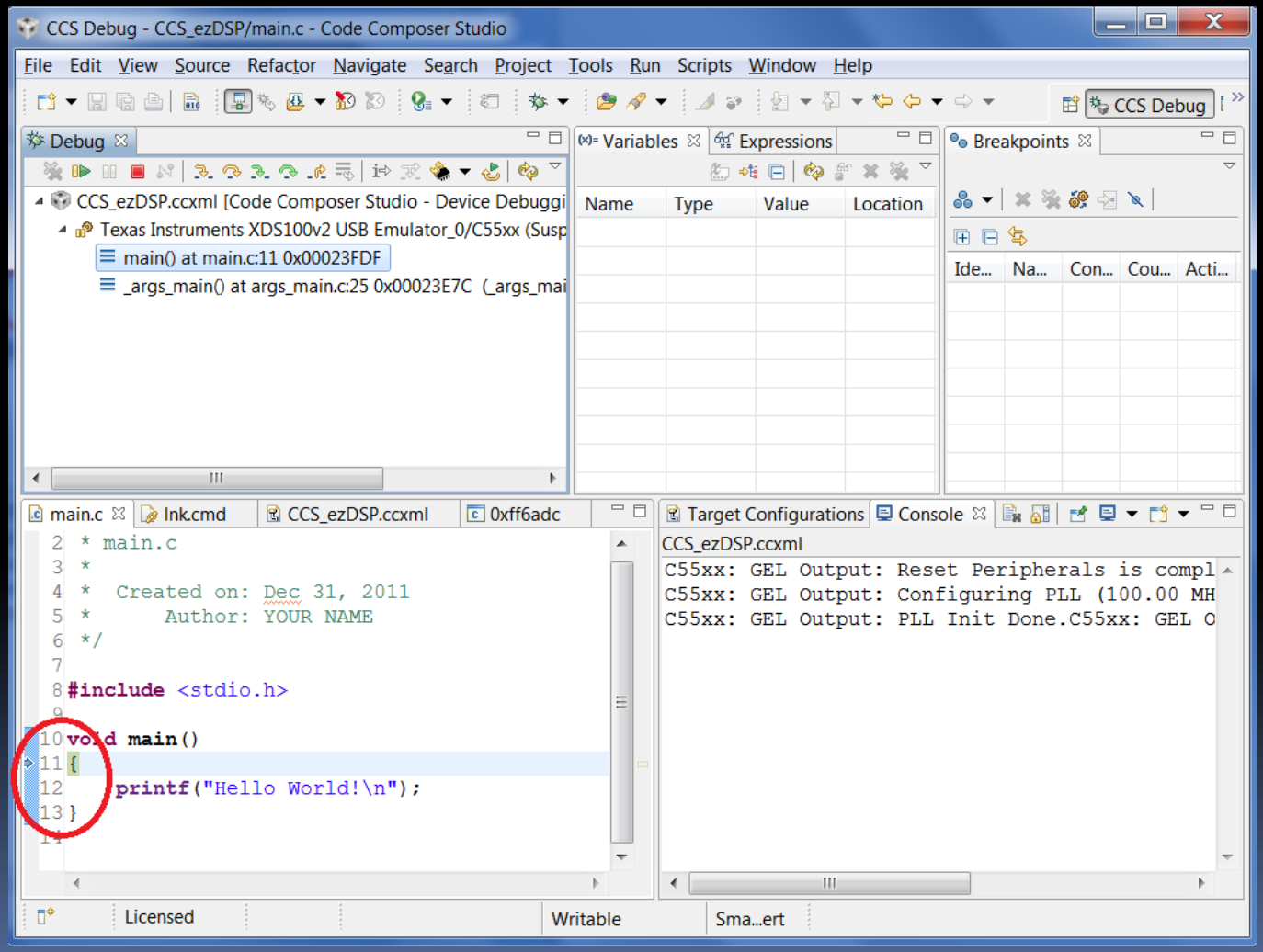
Load program CCS_eZdsp.out (3)

(CCS_eZdsp program is loading)



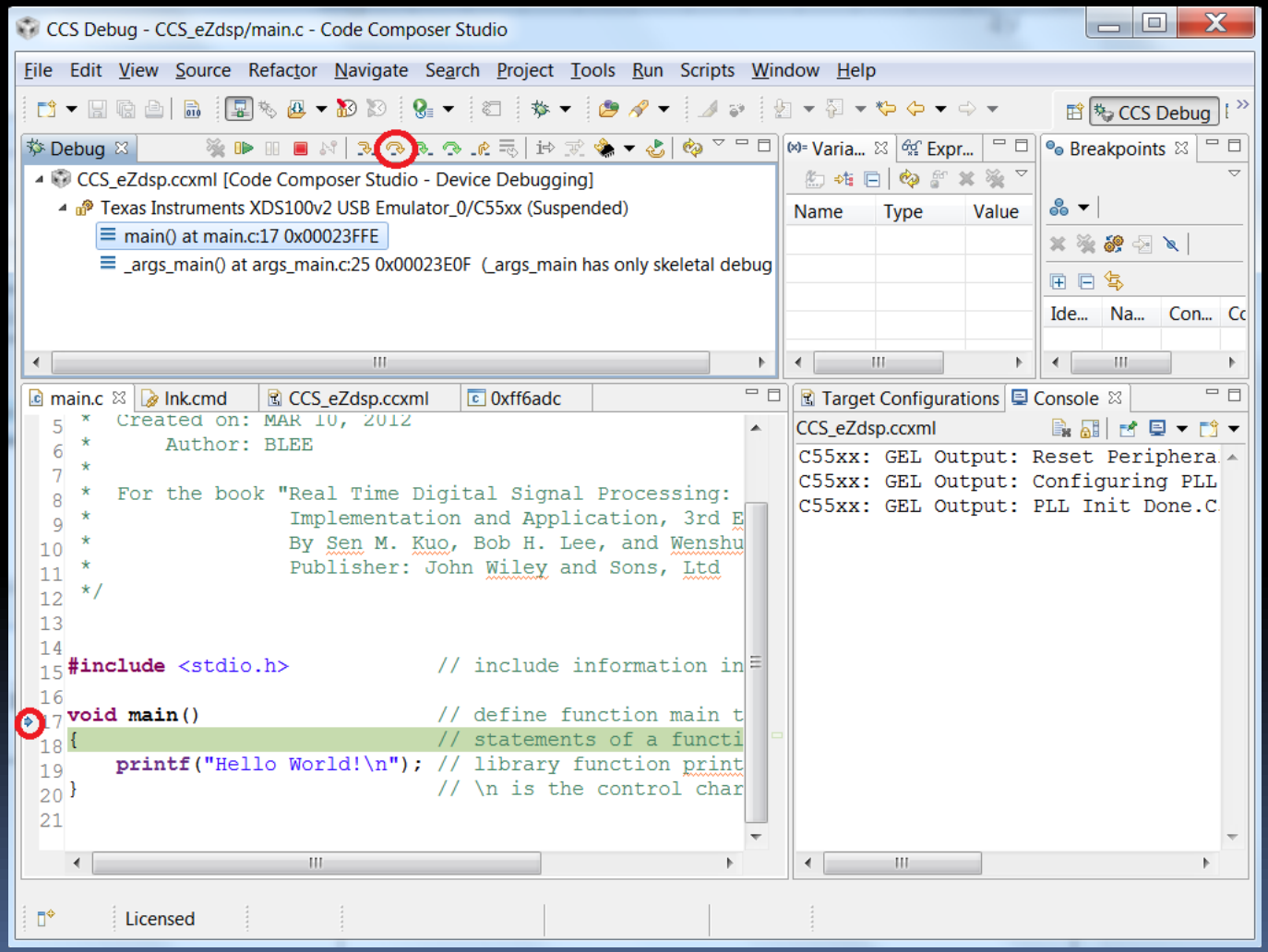
Once the program is loaded

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



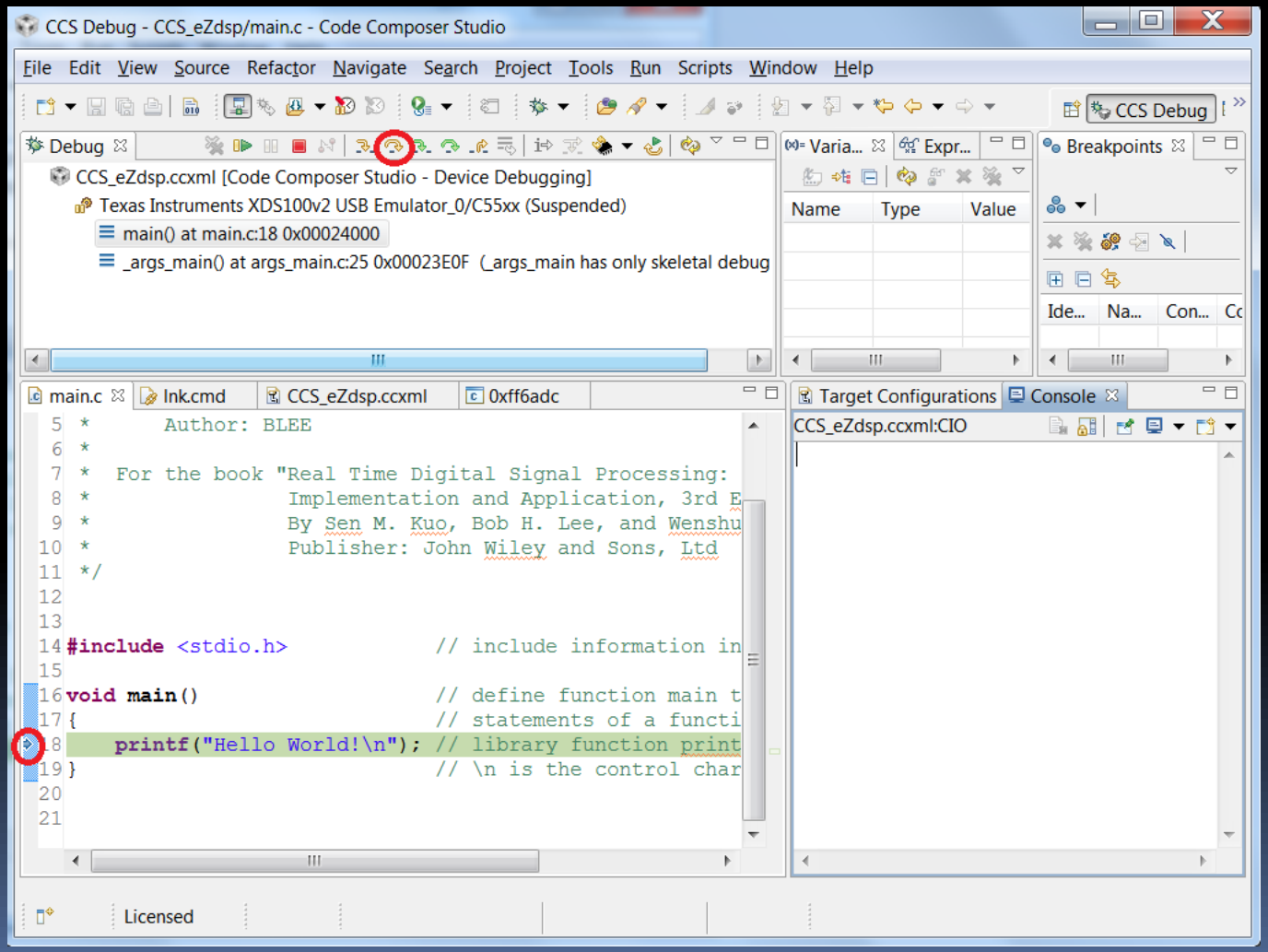
Once the program is loaded

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



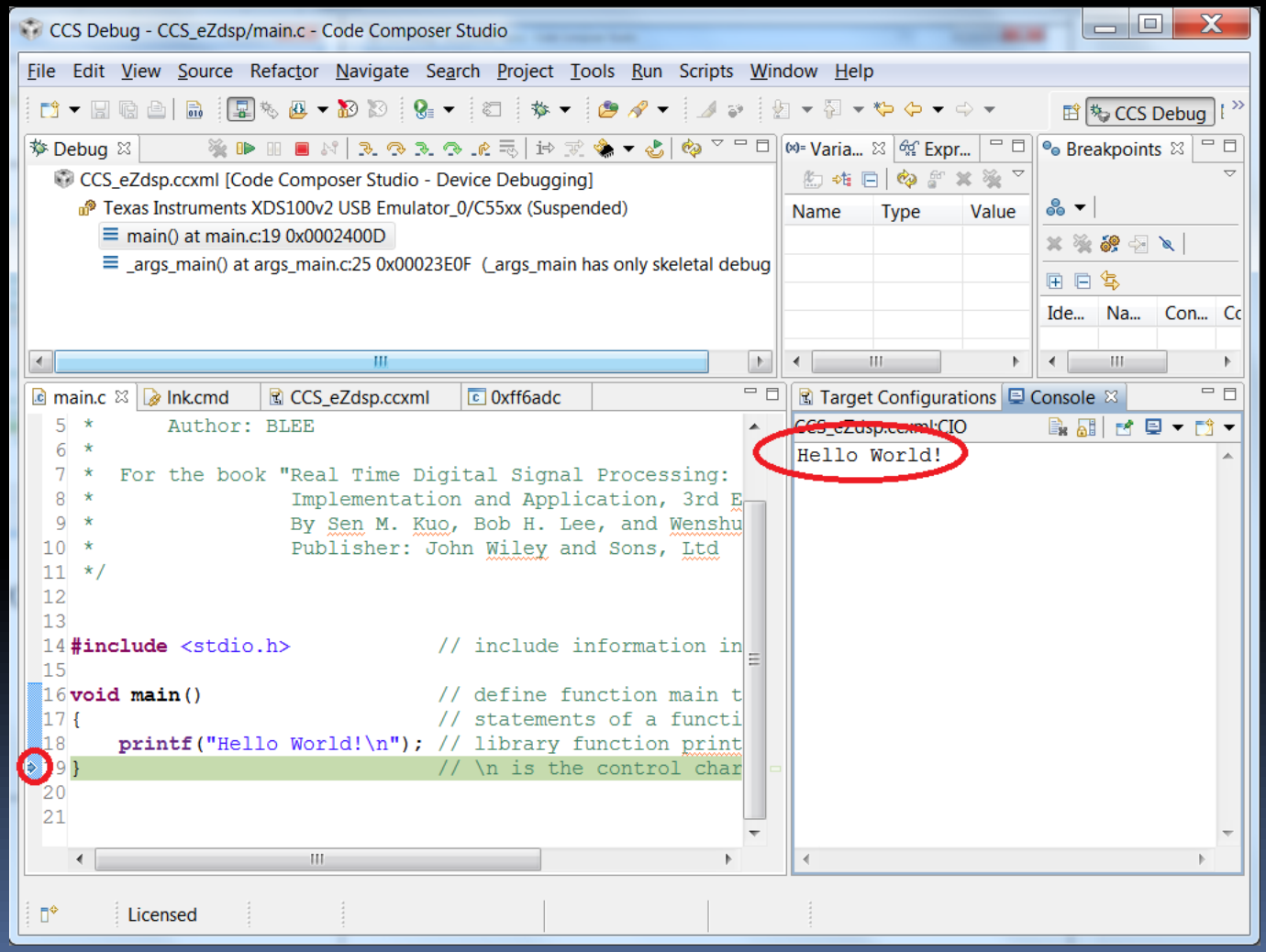
Step Over (F6) the program

(Watch program counter move through the program)



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 complier 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
- C Programming Language (2nd Edition), by Brian Kernighan and Dennis Ritchie, Prentice Hall