TI DSP step-by-step

C2000, C5000, and C6000

First Edition
Kyanoosh Shafaei

DSPgig.com

Notices

Knowledge and best practice in this field are constantly changing. As new research and experience broaden our understanding, changes in research methods, professional practice, or medical treatment may become necessary.

Practitioners and researchers must always rely on their own experience and knowledge in evaluating and using any information, methods, components, or experiments described herein. In using such information or methods, they should be mindful of their own safety and the safety of others, including parties for whom they have a professional responsibility.

To the fullest extent of the law, neither the Publisher nor the authors, contributors, or editors assume any liability for any injury and/or damage to persons or property as a matter of products liability, negligence or otherwise, or form any use or operation of any methods, products, instructions, or ideas contained in the material herein.

Registered at Library of Congress.

ISBN: 9798529159101

First printing edition 2021.

kshafaei@dspgig.com

www.dspgig.com

IMPORTANT NOTICE

DSPgig, Inc. reserves the right to make changes to its products or to discontinue any product or service without notice. Customers are advised to obtain the latest version of relevant information to verify that the data being relied on is current before placing orders. DSPgig, Inc. provides no warrants for this book and related software. Testing and other quality control techniques are utilized to the extent deemed necessary but provide no warranty. Please be aware that the products described herein are not intended for use in life-support appliances, devices, or systems. DSPgig does not warrant nor is DSPgig liable for the product and information described herein to be used in other than a development environment. DSPgig, Inc. assumes no liability for applications bugs, customer product design flaws, software performance, or infringement of patents or services described herein. All the information provided here is just for the reader's information, and it is the responsibility of the reader to verify them before using them in any means. Nor does DSPgig warrant or represent any license, either express or implied, is granted under any patent right, copyright, or other intellectual property rights of DSPgig, Inc. covering or relating to any combination, machine, or process in which such Digital Signal Processing development products or services might be or are used.

Contents

Preface

1) Introduction

This book is designed for college students or engineers unfamiliar with Texas Instrument (TI) Digital Signal Processors (DSP). The book focuses mainly on practical TI DSP programming in a fast and simple way. The original idea of the book was formed after many years of teaching various DSP courses and gathering all the keynotes from my courses to form a step-by-step DSP tutorial.

TI DSPs are among the best signal processors in the market but learning them is a real challenge. Having multiple DSP families, TI has published hundreds of documents for different families, which are too much to look through. This book provides what a beginner needs in one package with references and examples to build the foundation on TI DSPs. Moreover, in most chapters, the material is covered thoroughly, and there is no need to refer to additional references.

Most TI DSPs are part of 2000, 5000, or 6000 families. Although TI DSP families have many similarities, at the same time, they are very different. Including hundreds of TI documents in a single book is near impossible. However, by the end of this book, the reader should have a good idea of what he or she needs and where more detailed information can be found. The reader should be able to handle simple to moderate DSP projects for most TI DSP families by following the steps provided in each chapter.

Although the whole book can be read in less than 18 hours, it is highly recommended to install CCS (latest version and version 5.5) and do the examples and exercises. This is the only way to learn the topics entirely.

Although each chapter focuses on one particular subject, the first eight chapters cover the basics and need to be studied in order.

2) What else is needed?

The book contains multiple examples and exercises which can be tested by Code Composer Studio (CCS). One way to practice these exercises is using the DSP hardware. Another option is using an older version of CCS (version 5.5) which has a useful feature to simulate the DSP hardware. So if you don't have access to hardware such as DSPLab, use 'CCSv5.5'. The book is based on 'CCSv10.2', however, some examples can be tested with 'CCSv5.5'. All versions of CCS are free and can be downloaded from the TI website.

Also, the example codes used in this book are available at 'wiki.dspgig.com/dsp-step-by-step'.

3) Acknowledgment

First, I must start by thanking my awesome wife, Hanieh. From supporting me during the last four years while I was busy with writing 5 different books, to giving me advice on the covers. In fact, she was as important in getting this book done as I was. Special thanks to Kamran Ataran Rezai and Faraz Momahedi Heravi for proofreading the book and suggesting valuable corrections. Also, I like to thank Drew Vigen for giving me detailed and constructive comments on some of the chapters.

Please don't hesitate to send your suggestions or any corrections directly to me at 'kshafaei@dspgig.com'.

Chapter 1

CCS INTRODUCTION

Kyanoosh Shafaei
www.DSPgig.com

A computer is needed to study this chapter.

1) Introduction

Digital Signal Processors (DSP) from Texas Instrument (TI) company can be programmed by Code Composer Studio (CCS). There have been many versions of CCS. Most[1] of the TI processors can be programmed using the latest CCS version (which is version 10 at the time of publication). From the CCS version 3.3, the CCS environment is changed to a new Eclipse-based IDE[2]. The Eclipse-based IDE was introduced in 2009 with CCS version 4.0, and currently (in 2021), the latest version is CCS version '10.2'. Because the software's overall appearance has not changed over the past 12 years, it is very easy to switch from one version to another.

1.1) CCS and Simulators

For years, CCS has been one of the few embedded C software which supports 'simulators'. The DSP simulator runs the instructions inside the computer, so C or assembly projects can be tested inside the PC without using any DSP hardware. Therefore simulators are very useful to check the code (without the need to use DSP boards). Unfortunately, while many other companies add simulators to their software, TI decided to remove the simulator from 'CCS version 6.0' onward!

[1] - Some older DSPs are only accessible using very old versions of CCS (such as version 'CCSv2.2'). Even some very old DSPs are not supported under any CCS version and have special compilers.

[2] - One of the problems for software developers was the difference between IDE environments when switching between software. For example, if a developer works on JAVA, C, and .NET, then the location of settings, builds, and etc., was different for every environment. Eclipse IDE tries to solve this problem. If all IDEs use the same user interface type, it is very easy to switch to other environments after a developer learns one environment. Therefore TI decided to switch to Eclipse IDE as a standard IDE used by many companies.

One of the huge benefits of the simulator is for learning the CCS. Therefore in some chapters of this book, the 'CCS version 5.5' is used instead of the latest CCS. The 'CCSv5.5' is the latest CCS that supports the simulator. If you have access to TI DSP hardware, you may choose to use the hardware and the latest CCS version. However, make sure to use the same DSP family. For example, in this chapter, the C55xx simulator is used. So it is highly recommended to use any C55xx hardware (such as DSPLAB) for this chapter.

In the following chapters, we will switch between different DSP families, and if you don't have access to all DSP family hardware (such as C66xx, C64xx, C54xx, and C28xx), it is recommended to install both 'CCSv5.5' and the latest version.

There are some bugs in 'CCSv5.5' that do not exist anymore in the newer releases. However, we use these bugs to learn more about the core of the software and how it works.

1.2) Old JTAGs compatible with 'CCSv3.3'

CCSv3.3 was one of the stable CCS versions which supports many DSPs. After the Eclipse-based version of CCS was introduced (from 'CCSv4'), the support for some of the old JTAG debuggers was removed from the software. These JTAG debuggers[3] would cost more than $3000 and still can work very well for many DSPs. So, if there is an old JTAG debugger in the DSP laboratory and its name is not listed under the supported JTAG for the new version of CCS, try 'CCSv3.3'. Almost many concepts explained in this book are also applicable to version 3.3. The main difference between version 3.3 and the more recent versions is the 'workspace'. The 'workspace' is later explained in this chapter.

1.3) Installing 'CCSv5.5'

Check the following steps to begin:

• Download the CCS software version 5.5 from the TI's website at:

https://software-dl.ti.com/ccs/esd/documents/ccs_downloads.html

[3] - The JTAG debuggers are explained in the following pages.

Also, it is highly recommended to download the latest version of CCS and install it. If you have access to DSP hardware, instead of 'CCSv5.5', the latest version can be used.

• Then install both versions of CCS:

a) CCSv5.5: Code composer studio version 5.5
b) The latest version of CCS: At the time of this publication, the latest CCS version is version '10.2'. One might choose to install a more recent version if available.

Both versions are used throughout this book. The 'CCSv5.5' is needed because it is the latest version that supports simulators.

Make sure to follow the default settings for installation and not remove any default packages from the installation.

• Run Code Composer Studio version 5.5 by double-clicking on the following icon on the desktop.

• Next, the 'Workspace Launcher' window opens. Enter an address for the project in the 'Workspace' area. The projects by default are created in the workspace directory.

1.4) Workspace

Workspace is the location for all the projects and their settings and dependencies.

In the eclipse-based IDE, a directory called 'workspace' is defined for the projects. This directory keeps all projects in one folder and displays them inside the IDE. By default, any new project is located inside the workspace folder.

So workspace is the parent folder of the projects' folders.

The workspace is like a container for everything related to the project. For example, project 'A' may use a library 'B'. Library 'B' has a source file 'C'. Also, project 'A' is part of a multi-core DSP system with two cores. Project 'A' is for 'core 1', and project 'AA' is for 'core 2'. The CCS puts the settings to load project 'A' to 'core 1' and project 'AA' for 'core 2' in 'Config1'. The workspace

puts all these projects and related settings under one umbrella and keeps track of them. In this example, if file 'C' (which is part of library 'B') is changed, the workspace automatically compiles library 'B' before compiling project 'A'.

A project folder does not necessarily need to be in the workspace folder. It can be in any folder outside the workspace folder, but the workspace can track the location of the project and its dependency.

After entering the address for the workspace folder, click on the 'OK' button.

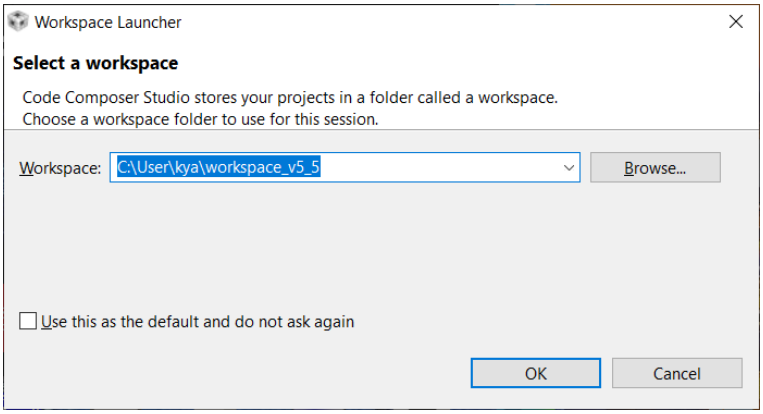


Figure 1: The first window opens after running 'CCSv5.5'

1.5) Licensing[4]

Now, the main window opens. After a while, a window similar to Figure 2 is opened and asks for a software license type. There are four options:

a- The **third** option is for 'FREE LICENSE', and works for 'simulators'. Select option 3 for now.

b- The first option activates the software with no restriction. From 'CCSv7', TI removed the license from the software and made it practically free. A license file for the previous versions ('CCSv4', 'CCSv5', and 'CCSv6') is located under the CCS download page:

[4] - The CCS was an expensive software and worked with valid licenses. Some versions of CCS were only restricted to limited DSP hardware. From 'CCSv7', TI decided to make the CCS free with no fee. Also, a permanent valid license is provided for all older versions.

Free license for older versions

With the release of CCSv7 all previous CCSv4, CCSv5 and CCSv6 releases are free of charge.

- Unzip and copy the **License File for Older versions** into **<install dir>/ccs_base/DebugServer/license**
- Go to menu **Help → Code Composer Studio Licensing information**
- On the tab **Manage**, click on the button **Add...**
- Type or Browse to the path to this license file in the field **Specify a license file**
- Restart CCS.

Activate the 'CCSv5.5' using either option one or option three.

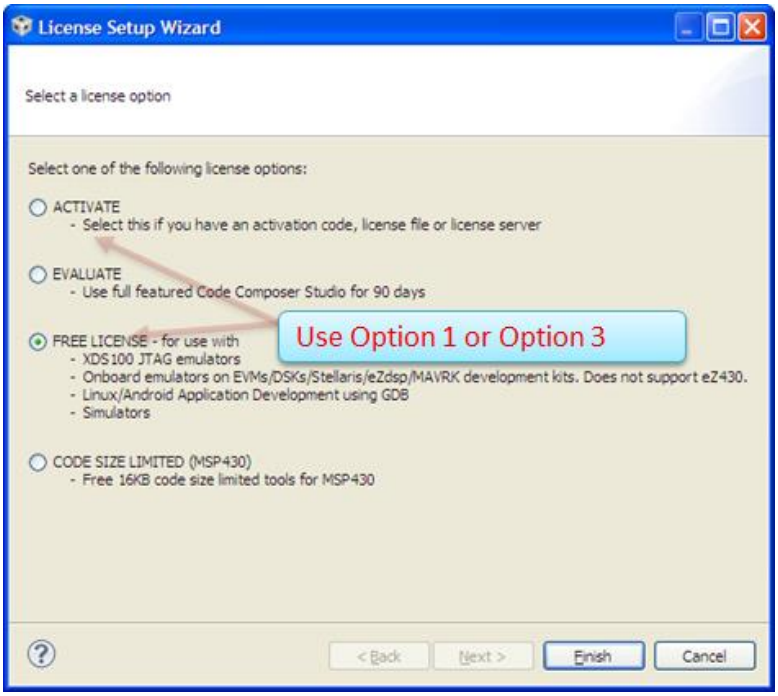


Figure 2: Activating 'CCSv5.5' (CCSv10.2 is free)

1.6) CCSv10.2

In case you have access to TMS320VC5509A hardware such as DSPLAB, you may choose to open 'CCSv10.2' instead of 'CCSv5.5'. From 'CCSv7' onward, some of the compilers are not installed by default by the installer. The CCS has introduced the 'APP Center' and placed some packages there for additional installation. Inside 'CCSv10.2', from the 'Help' menu, select 'CCS APP Center'.

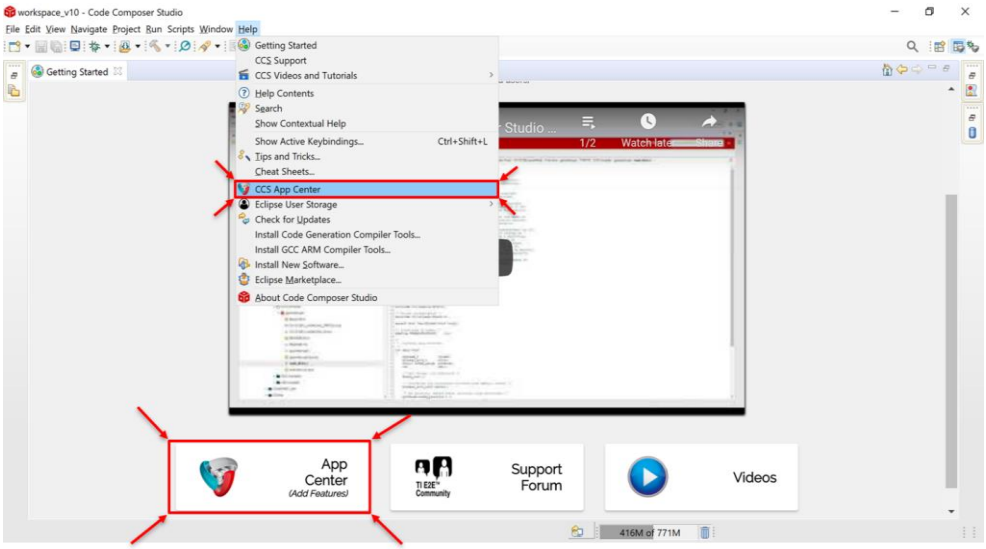


Figure 3: Opening the 'App Center' for installing new packages

Inside the ‘APP Center’, find ‘C5500 Compiler’ and select it. Then use the ’Install Software’ button to install the C55xx compiler as shown in figure 4.

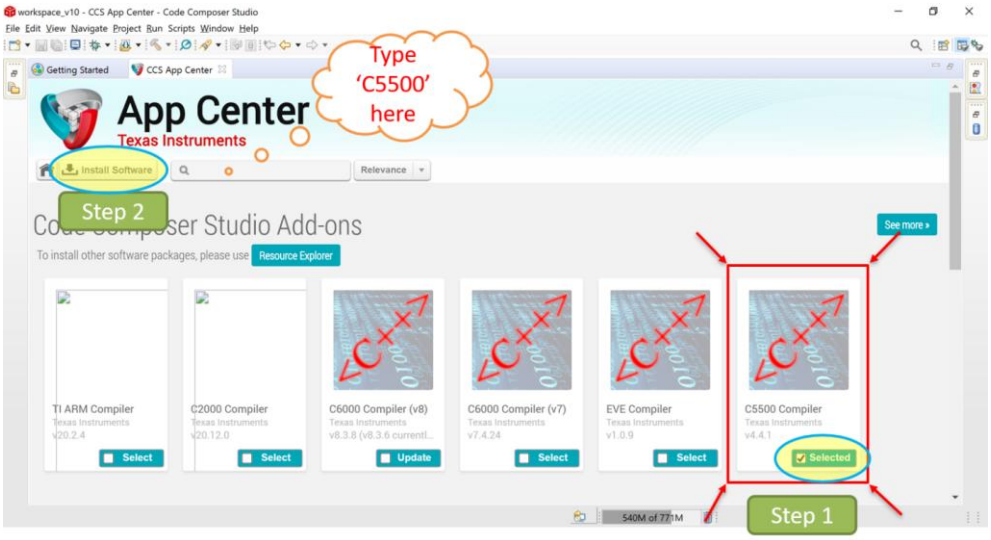


Figure 4: Steps required to install the C55xx compiler

In this book, the C55xx compiler is used in multiple projects. The main window of the CCS is shown in Figure 5. In the process of embedded system[5] programming, there are multiple phases:

[5] - Hardware with a dedicated processor and limited memory is often called an ‘embedded system’.

1- Edit: creating a project and typing the C or assembly code.
2- Debug: Loading the code to the processor memory and then run and test the code.

In each of these modes, the software menus need to be different. For example, in 'edit' mode, there is no need to reset the processor, so the 'reset' processor does not exist. Or in 'debug' mode, a new project is *not* created, thus, there is no 'New project' menu. Overall, the Eclipse-based IDEs support multiple modes for different applications.

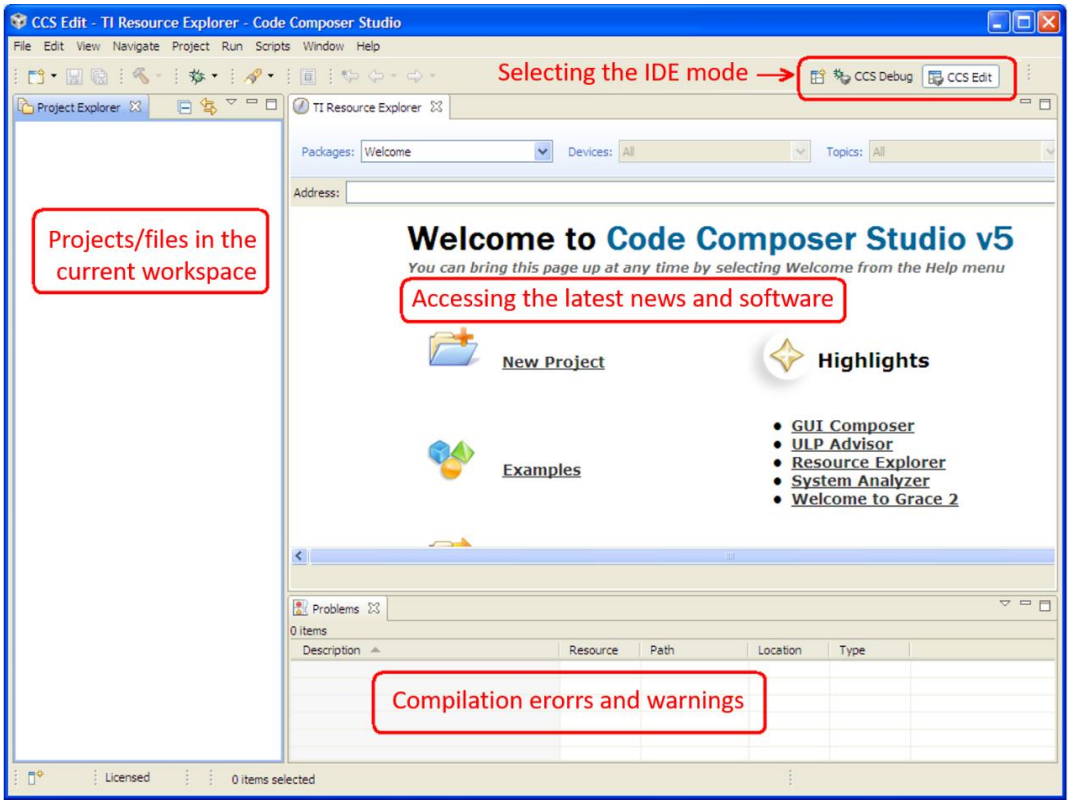
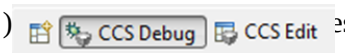


Figure 5: The home page of 'CCSv5.5' software

In CCS, the 'edit' and 'debug' modes can be selected from the upper right corner of the IDE (

ese buttons, the appearance of the software is changed, and different menus and windows are displayed.

In case of looking for a specific menu which does not exist, check the 'edit/debug' mode. For example, 'new project' is only available in 'edit' mode.

Before creating a new project, let's discuss how to connect a DSP board to the computer.

2) Types of JTAGs

JTAG emulator is a tool that connects the DSP processor to the computer. The TI DSP JTAG uses the 'IEEE 1149.1' standard to communicate to the processor. This standard is widely used by many other companies to develop JTAGs. Although JTAGs use the same standard, but only[6] TI JTAGs can be used for TI DSP[7]. The connection between JTAG and the DSP processors is usually a 14-pin connector.

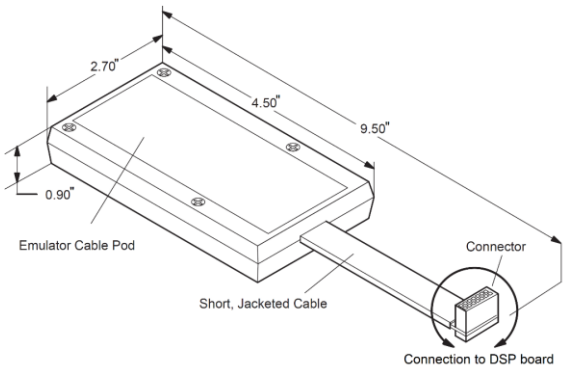


Figure 6: A typical 14-pin JTAG emulator

TI has developed some of the best JTAG debuggers for DSPs. The price for these JTAGs has been reduced in recent years, but TI JTAGs are still among the best JTAGs in the market. The following figure presents the JTAG pins.

TMS	1	2	$\overline{\text{TRST}}$
TDI	3	4	GND
PD (V_{CC})	5	6	no pin (key)†
TDO	7	8	GND
TCK_RET	9	10	GND
TCK	11	12	GND
EMU0	13	14	EMU1

Figure 7: The connector for the TI JTAG emulator in the DSP board

There are three types of DSP JTAGs: The '560 series', '510 series', and '100 series'. The '560 series' is the most advanced JTAG, and the '100 series' (like XDS100) is the most affordable.

---

[6] - There are a few other manufactures with compatible JTAGs which can be used for TI DSPs.
[7] - This standard defines the physical commands and the internal structure of the debug port. But it does not define the details of all internal debug port registers. So even though all JTAGs use the same standards, because the internal structure of processors is different, the JTAG of one company can not be used for another company.

Figure 8: XDS100v2 is a small JTAG that works with many DSPs

Figure 9: 510 series manufactured by different companies

2.1) Connecting JTAGs to DSP boards

The JTAG emulator is responsible for loading the code into DSP memory and real-time debugging. Most DSP processors have dedicated pins for the JTAG emulator. These pins can be connected to JTAGs in different ways. Most JTAGs (especially '510 and 100 series') are connected to DSPs by a 14-pin connector.

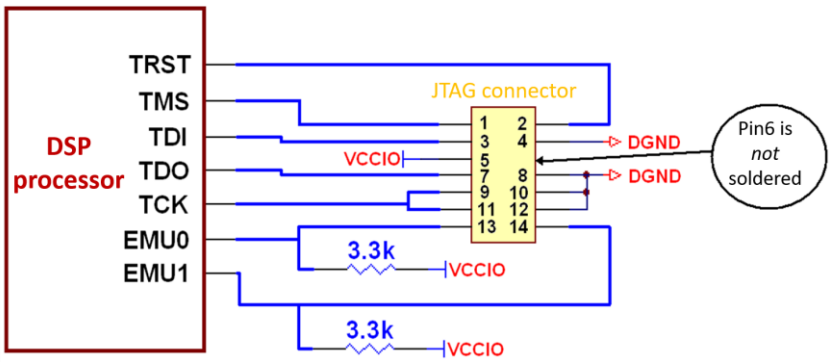


Figure 10: JTAG connection circuit inside DSP[8] board

[8] - On most boards, VCCIO is 3.3V or less. The recommended value for two pull-up resistors is between 2.2k to 10k ohm for different DSPs.

If you have a C55xx DSP board, connect it to the computer using the emulator. Remember to always connect the JTAG cable to the DSP board *while the DSP board is off*. Also, the JTAG emulator is not very stable, and any mechanical shock can cause JTAG disconnection.

Most TI DSPs have a document for the emulator connection. Always refer to that document when designing new hardware.

3) Creating a new project

To create a new project, from the 'File' menu, select 'New' and then click on the 'CCS Project'.

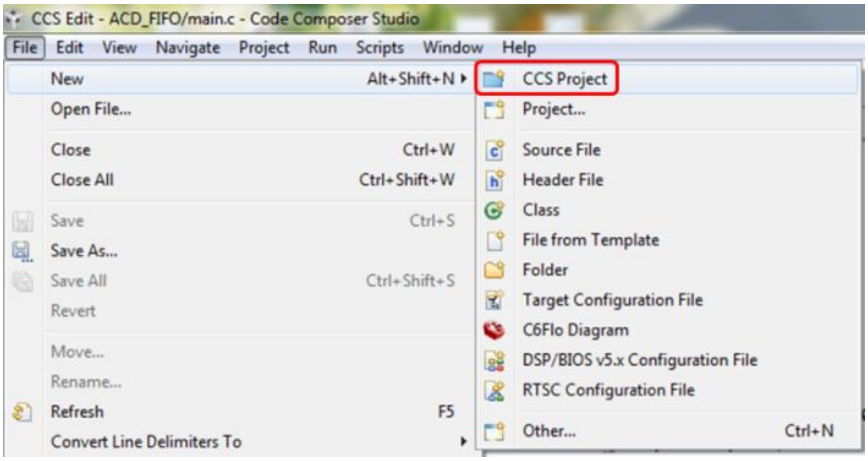


Figure 11: Creating a new project

Adjust the 'New CCS Project' window based on the following picture:

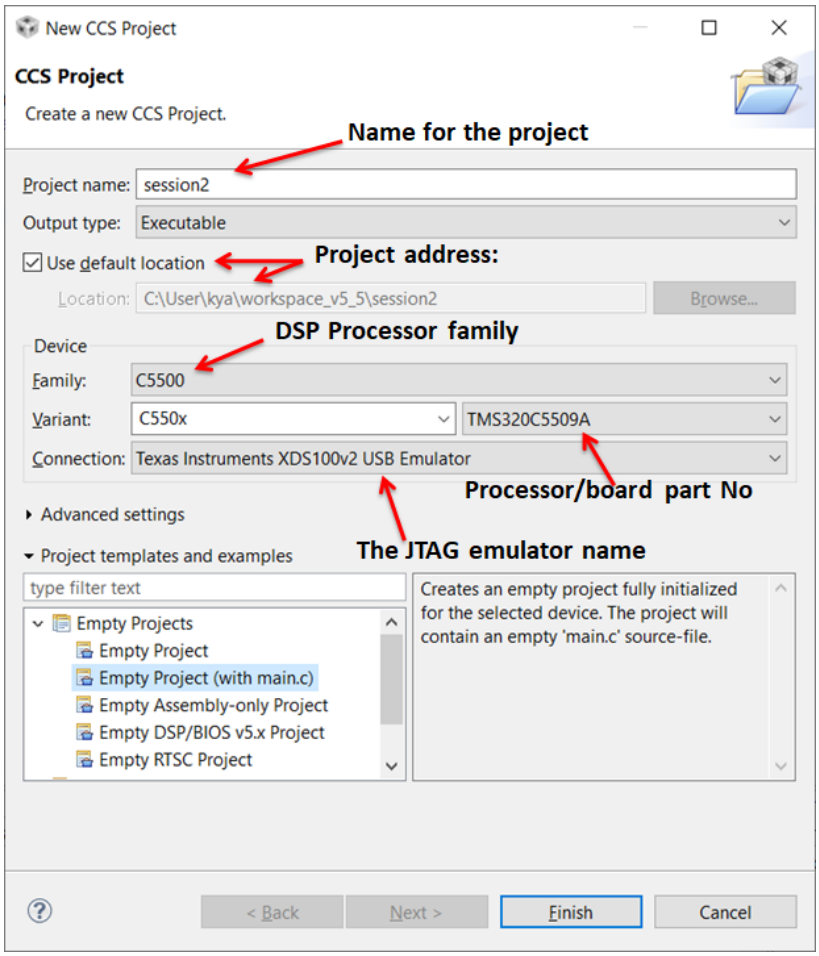


Figure 12: Project settings in 'CCSv5.5'. In the later version of CCS, this window is a little different.

In this window, the project address can be modified to any location, but for now, check the 'Use Default Location' to create the project inside the workspace.

Two options can be selected in the 'output type' section: The 'Executable' and 'Static Library'. The 'Executable' is the default and creates an executable file (with the '.out' extension) that can be loaded to the memory later by the JTAG. The 'static library' is for creating standard libraries (with a '.lib' extension) that can be added to other projects[9].

[9] - There are two methods to use a C function inside a project:

A- Adding the source code (C file) to the project.

B- Placing the function inside a library and then adding the library to the project: In this approach, the C function is added to a library project, and then the output of the library project ('*.lib') is added to any project. The library can hide the source of c functions from the end-user.

In both methods, the function name should be defined with the keyword 'extern' in the end-user code. For assembly functions, the same rules are applied. For more details, refer to future chapters.

For 'Processor Type', select the 'C5500' Series. The C55xx is a signal processor used for mid-side processing (up to 400 million calculations). Selecting 'TMS320C5509A' teaches CCS the exact processor part number, and then CCS adds the 'RTS library' and 'memory map file'[10] for that processor to the project.

Because the assumption is that there is no actual hardware connected to the computer, the 'Connection' should be the Simulator. But unfortunately, the CCS does not accept a simulator for the connection type in this window. Later, the 'Connection' is changed to Simulator. So for now, select one of the JTAGs such as XDS100ver2 or XDS100ver3[11]. Finally, click on 'Finish' to create the project.

4) What is a '.gel' file?

While creating a new project, the processor name (i.e. 'TMS320VC5509A') was given to the CCS. Thus, CCS assumes the hardware has only one chip, which is 'TMS320VC5509A'. What if the hardware also has memory? How should the CCS know to use this external memory and increase the available memory for the processor? The '.gel' file is a script file that helps CCS to configure the hardware correctly. The '.gel' file teaches CCS how and when to configure the DSP internal registers. In this chapter, the code is loaded into the simulator[12]. And for simulators, there is no need to configure any registers (therefore, no '.gel' file is needed). But for those readers who are using real-hardware (such as DSPLAB), the proper '.gel' file is needed.

In most cases, it is necessary to add a gel file when using DSP boards.

There is a default '.gel' file for each processor in the software(which is already added to the current project). But, when using hardware, it is sometimes necessary to replace the standard '.gel' file with the '.gel' file provided by the hardware manufacturer. This action should be taken on most external boards (such as DSK or EVM[13]).

Usually, a few processor registers need to be initialized when launching each hardware. The '.gel' file is a simple script that tells CCS when to program registers through the JTAG emulator. For example, there are functions in the '.gel' file that are called when CCS is connected to the

[10] - The RTS and Memory Map files are explained in future chapters.
[11] - These two JTAGs are affordable TI JTAGs that work with many of the famous processors.
[12] - There are multiple simulators in 'CCSv5.5' that can be used to examine DSP code without a real board. The simulators are so accurate that if a code works properly in the simulator, it will most likely work similarly on the real board. In practice, the difference between a simulator and a real board is the peripherals such as USB, serial, etc., which are not implemented in most simulators.
[13] - DSK or EVM is the general name for DSP boards made for DSPs and other processors.

processor through the JTAG[14]. And as the CCS programs the debug port inside the processor, it also copies a few numbers into specific addresses in memory (as instructed by the '.gel' file).

This may look complicated, but in most cases, there are only a limited number of registers that need to be programmed. Therefore the '.gel' file is not very complex.

It is important to note, depending on the processor's type[15], the JTAG emulator can successfully program the debug port without the '.gel' file.

The '.gel' file can have functions for every step of debugging. For example, for the DSPs with internal flash, the CCS uses the '.gel' file to unlock the flash before loading the program to the flash. So, there is a function in the '.gel' file that CCS calls before programming the flash.

The good news is, there is no need to write any '.gel' file while using DSP processors. It is only important to know the name and location of the '.gel' file needed for each hardware.

5) What is 'Target Configuration' file ('*.ccxml')?

In embedded software programming, after compiling the project, it should be loaded into the hardware. The hardware is usually connected to the PC with a JTAG emulator. The software IDE needs to know the JTAG information to load the project into embedded hardware memory. Assume there are multiple JTAGs connected to the same PC, so the software IDE needs to know which one of the JTAGs is for the current project. In the CCS, all these settings for the JTAG connection are stored in the 'ccxml' file. In some IDEs, the JTAG settings are stored in the project setting. But in the CCS, the JTAG settings are stored in a 'ccxml' file, therefore it is easy to copy the JTAG connection setting from one project to another.

---

[14] - OnTargetConnect

[13] - It depends on the processor type. For example, some DSPs have two cores: an ARM core and a DSP core. After power-up, the ARM processor starts executing the instructions, but the DSP processor is reset by default. Only the ARM processor can take DSP out of reset by changing a bit inside an internal register. However, the DSP kernel needs to get out of reset in order to connect to the JTAG emulator. Here without the '.gel' file, the DSP can never get out of reset, and the JTAG can not connect to it.

The JTAG connection for the current project was set to ‘XDS100v2’. The ‘ccxml’ file contains the JTAG name, processor’s name, ‘.gel’ file name and address, and JTAG ID (in case of multiple JTAG emulators).

Double-click on the file with the ‘*.ccxml’ extension in the ‘Project Explorer’ window to open it. In ‘CCSv10.2’, the ‘ccxml’ file is in the ‘tagetConfigs’ folder, but in ‘CCSv5.5’, it is in the root project directory. If there is no ‘ccxml’ file, add a new ‘Target Configuration File’ by right-clicking on the project’s name and selecting ‘New’.

The target configuration file (‘ccxml’) defines the hardware and can be shared by multiple projects. Therefore while creating a new ‘ccxml’ file, there is an option to select a shared location for the ‘ccxml’ file, which later can be used by multiple projects. In this case, the file can be outside the current project folder.

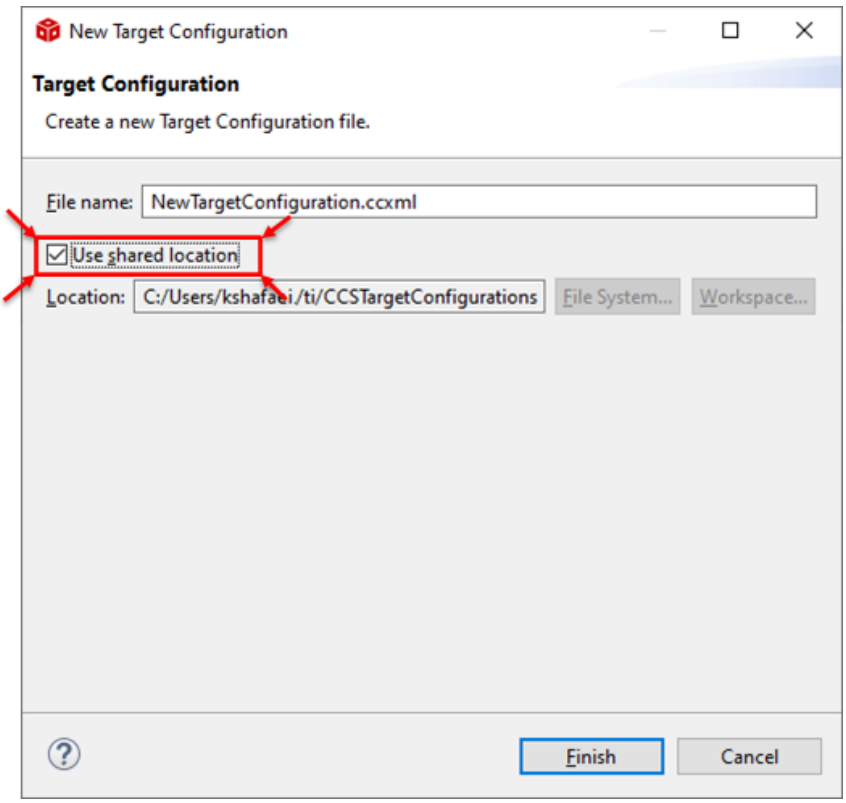


Figure 13: Using a shared directory for all projects when creating a new ‘Target Configuration File’

In ‘CCSv3.3’, the JTAG connection setting was done by another software outside the program. These settings cannot be moved from one computer to another computer. However, in version 4 and above, these settings are in the ‘ccxml’ file and can be transferred from one computer to the new computer along with the project.

5.1) ‘Target Configuration’ file settings

Now double-click the ‘*.ccxml’ file to open it.

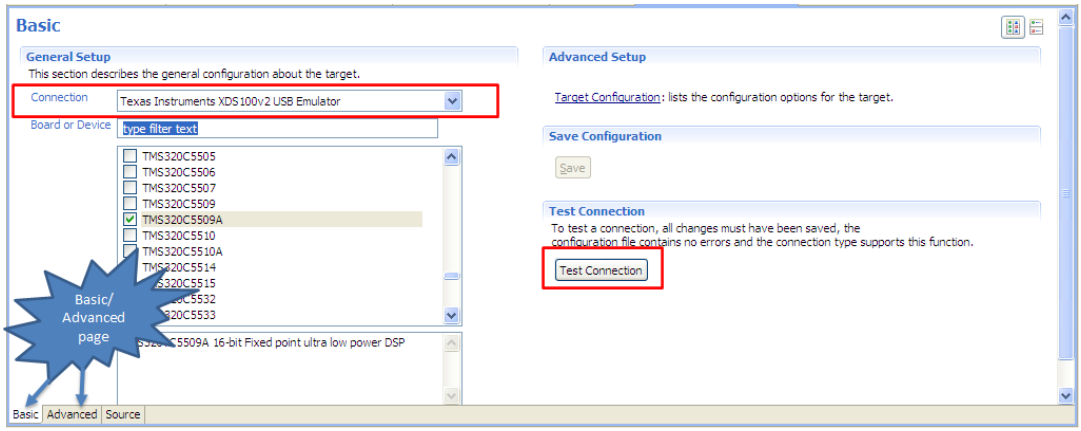


Figure 14: ‘Basic’ page inside ‘Target Configuration’

The ‘ccxml’ file has two windows:

A- The ‘Basic’ window is for the JTAG type (for example, ‘XDS100v2’) and the processor name. There is a button called ‘Test Connection’. If the JTAG is already connected to the computer, pressing ‘Test Connection’ checks the connection between the computer and the processor in the hardware. *Before this test is passed, any attempt to load the code to the hardware will fail*.

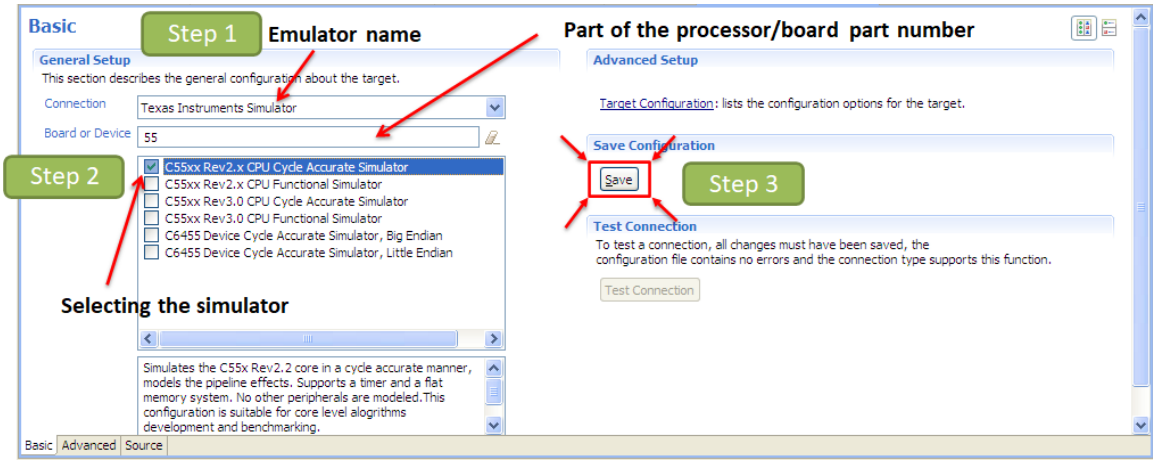


Figure 15: Using the simulator instead of JTAG emulator (only in 'CCSv5.5')

B- The 'advanced' window specifies the '.gel' file. To change the '.gel' file, select the processor's name and use the 'Browse' button to select a new '.gel' file. Also, when there are multiple JTAG emulators, the *ID of the JTAGs* can be used here to select the JTAG emulator.

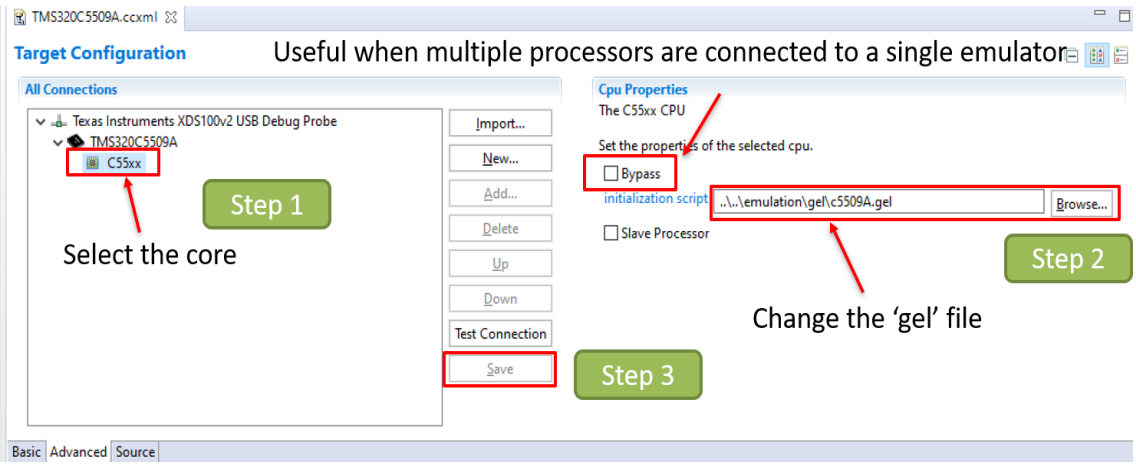


Figure 16: Changing the '.gel' file in 'ccxml' file

The simulator does not need the '.gel' file. Depending on the manufacturer's advice, the '.gel' file may be needed only when using the hardware.

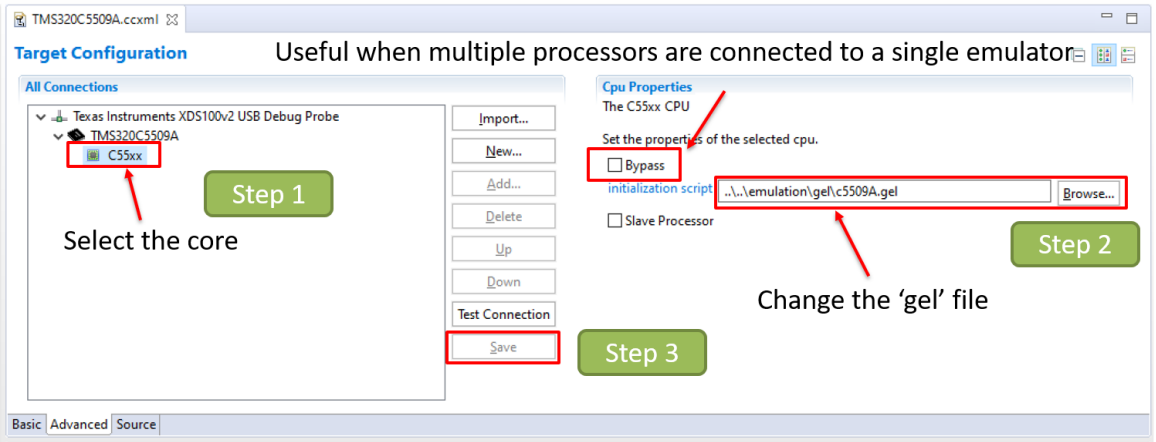


Figure 17: Changing the debugger setting in the 'ccxml' file

Inside the 'Basic' window, select the 'Texas Instrument Simulator' for the 'Connection' section (only available for 'CCSv5.5')[16]. Now, type '55' in the 'Board or Device' field to reduce the list, and then select the 'C55xx Rev2.x CPU Cycle Accurate Simulator'. Finally, click on 'Save' to save the settings (check Figure 17).

Now everything is ready to write the first program.

6) First C program in CCS

Every C project needs to have a `'main()'` function. Right-click on the project's name, and from the 'New' menu, select the 'Source File'. Name the new file as 'main.c' and save it in the project directory. Now double-click on the 'main.c' file in the 'Project Explorer' to open it. Modify the 'main.c' file as follows:

```
/*
 * main.c
 */
#include <stdio.h>
int main(void) {
    printf("Hello ...");
    return 0;
}
```

[16] - After selecting the emulator in the 'Connection' field, the list of supported processors for that emulator is displayed. So if you decide to buy a new JTAG, always check here if the JTAG supports the target processor. There is only one exception which is 'XDS510'. This JTAG has been on the market for many years, and many unauthorized versions of 'XDS510' have been made that cannot even work with 'CCSv5'.

7) The command file ('*.cmd')

In some embedded processors, complicated linker scripts exist for memory management. However, TI has a simple and very effective solution for memory management. In the CCS, the command file (with '.cmd' extension) is used to manage memory. The command file directs the linker on how to use the processor's memory for code or data. Chapters 3 to 5 are about the command file.

In CCS, there is a default command file, but only simple projects work with this default command file. However, to make sure there is no problem with the code in this chapter, add the following command file to the project. The command file is *case-sensitive*, therefore exactly follow the lower-case and upper-case text.

Code 1: The 'simple.cmd' command file in the project directory

```
MEMORY
{
PAGE  0:

      DRAM1 : origin =00400h , length=10000h
}
SECTIONS
{
     .stack: {}  >  DRAM1   PAGE    0
}
```

The 'printf()' function uses a huge part of the stack. This project has a stack size problem. The above command file places the 'stack' at the beginning of the memory area to hide the StackOverflow bug. In future chapters, this problem and a proper solution are explained. For now, *do not change any part of the command file*.

To add a new file (with '.cmd' extension), from the 'File' menu, select 'New' and then 'Others'. In the window that opens, from 'General', select 'File' (next figure). Then follow the wizard, and at the end, make sure to select a name with '.cmd' extension for the file (such as '5509.cmd').

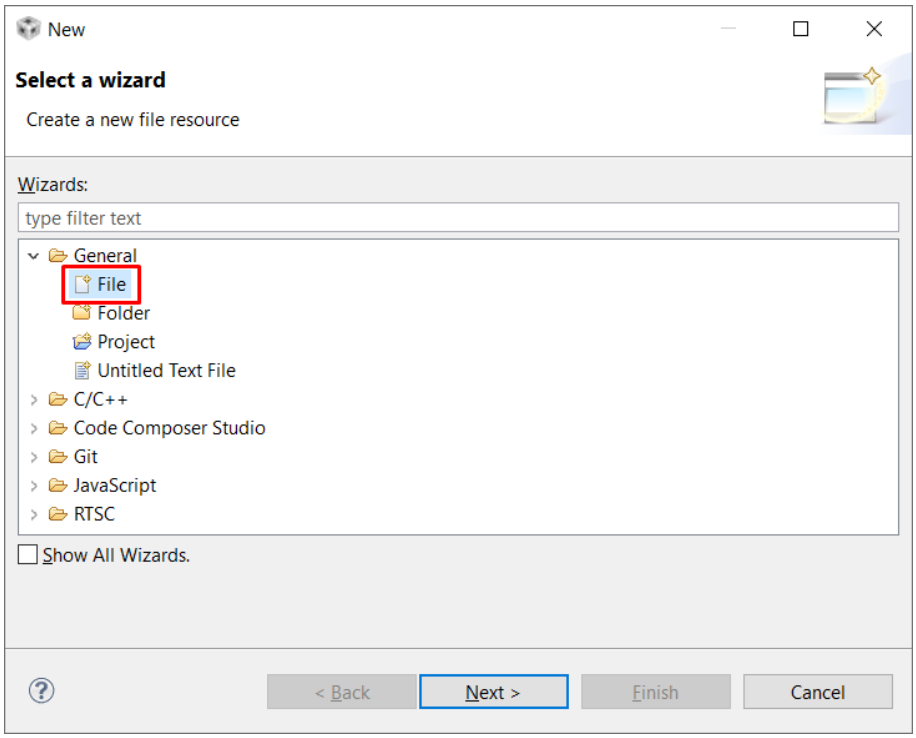


Figure 18: New file wizard

8) Build and Run:

Select the icon  to build[17] the project. Make sure no errors are generated. If there is an error in the project, try to fix it. Errors in CCS are displayed in the bottom right window (the following figure).

Figure 19: The error window

[17] - Build means compile and then link. The build process is explained in more detail in future chapters.

If the project has no error, enter the 'debug' mode by clicking on the icon. The debug icon ( ) is a shortcut to do all the steps of compiling, linking, and running the program. Pressing the debug icon switches the CCS mode from 'edit' mode to 'debug' mode. There are two buttons at the top right of the IDE (next figure), which can be used to change from 'edit' to 'debug' mode.

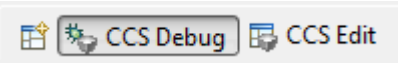


Figure 20: CCS 'edit' mode or 'debug' mode

If everything is performed successfully, clicking on changes the IDE mode to 'debug' mode with no error message. Then the debug window (figure 21) shows the code execution stopped at the 'main()' function. Use the icon to execute the code.

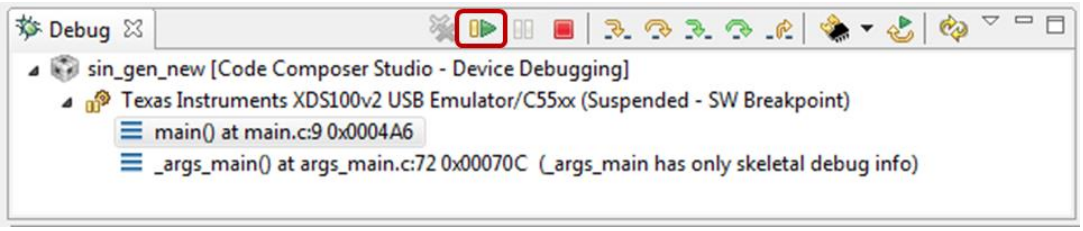


Figure 21: Running the program

After running the program, it should print a text in the 'Console' window on the bottom right corner of IDE.

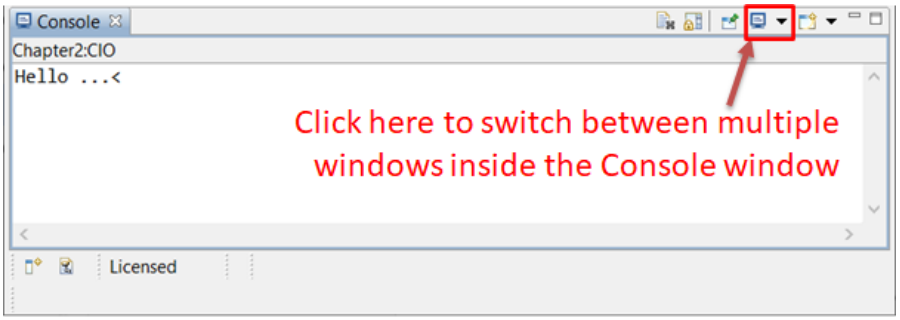


Figure 22: Console window displays the compilation output and the 'printf()' output(CIO)

***Exercise 1**: Use the other icons (such as) next to the icon. Restart the debug session by pressing before trying these icons. Pay attention to the difference between assembly single step and C single step.*

'Printf()' function is one of the useful functions that TI supports for its processors. The 'printf()' sends the text string through the debug port to the computer, and then CCS displays the text in the 'Console' window. TI also supports functions such as 'fopen()' or 'fwrite()' and 'fread()'.

Exercise 2: *Try a very simple program to read from a file and display the contents with 'printf()'. Note when using the 'fopen()' to open the file, the current folder that 'fopen()' looks for the file is the 'debug' folder inside the project folder. So put the input file in the 'debug' folder. The 'debug' folder is created inside the project folder after compiling the project for the first time.*

9) Some software features for debugging

9.1) Break point

Change the program as shown next:

```
#include <stdio.h>
int i= 10;
int  Array[100];
main()
{
   printf("Hello .... " );
   for( i = 0 ; i< 100 ; i++ )
   {
    Array[i] = i;
   }
}
```

Rebuild the project () and if there was no error, load it onto the simulator ().

The  icon is a simple solution to load the code into memory, but it takes longer because it disconnects the debug port and reconnects again, and then loads the project. After changing the code, if the program is already in the 'debug' mode, select the 'Build All' (CTRL + B) from the 'Project' menu. This builds the project and then loads it into memory (without disconnecting and then reconnecting). After the build, select the 

when asked to reload the code.

Inside the 'main.c' file, go to the line 'Array[i]= i;' and select 'Toggle Break Point' (CTRL+SHIFT+B) from the 'Run' menu (if there was no 'Run' menu in the software, change the software environment from 'edit' to 'debug' mode). This adds a breakpoint at the line. The

breakpoint is one of the tools used for troubleshooting. The breakpoint is marked with a blue circle ( ) that appears on the corresponding line and stops the execution on that line.

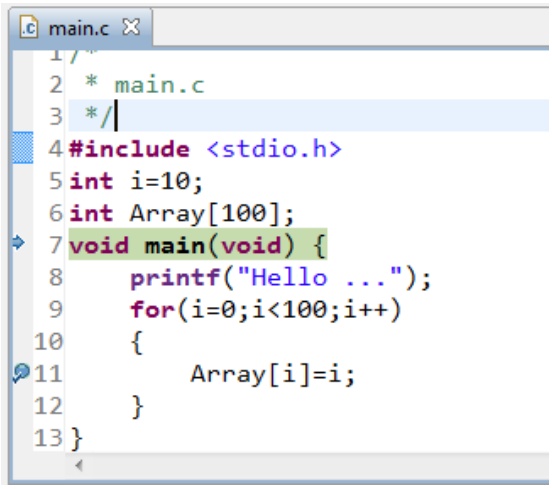


Figure 23: Creating a Breakpoint in software

If the code is already loaded into memory, use F8 ( ) to run the code. The program execution stops at the breakpoint location.

The breakpoint is in the `'Array[i]=i;'` line (which is after `'printf()'`), but why isn't the `'printf()'` function not displayed at the 'Console' window? When the `'printf()'` was designed, there was a huge cycle overhead to transfer the data from the board to the computer. To improve the execution speed (and reduce the overhead of the handshake[18] between the computer and the board), the `'printf()'` stores all the text inside an internal buffer. When `'printf()'` detects a 'next line character' (`'\n'`) in the text, it starts the transfer of all the previous text. Here, the printf text does not have any `'\n'`, and even though the function is already executed, the text has not been sent to the computer and is held inside an internal buffer. To see the text, either add a '\n' to the text or execute the `'main()'` function until the whole code executed:

```
printf("Hello … \n");
```

[18] - The CCS uses the JTAG debugger to read the data inside the DSP memory. For some memories, the JTAG debugger has to stop the core before reading from memory. That can slow down the code execution while debugging. There is a special synchronizing mechanism between the 'printf()' function and the JTAG debugger to reduce the number of memory accessed by the debugger.

10) Watch window

From the 'View' menu, select the 'Expression' window. This will open the watch Expression window. The 'Expression' window can display the variables' value during execution. Inside the 'Expression' window, click on the 'Add new expression' and type `'i'` (next figure). Next, add `'Array'` to the 'Expression' window.

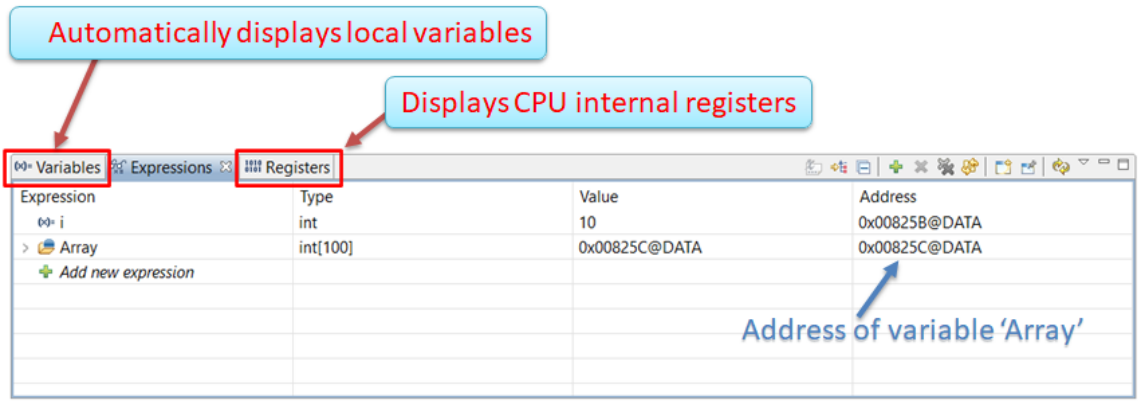


Figure 24: 'Expression' window

Use F10 (C single step) multiple times and check how `'i'` is changing. Click on ⊞ to display the contents of `'Array'`.

Exercise 3: *The 'Expression' window can display more complex variables. Try the following expressions inside the 'Expression' window:*

```
&i                    ← This is the address of 'i'
Array[i]
2*i+5
```

11) Memory view

From the 'View' menu, select 'Memory Browser'. The memory browser window opens. The memory browser shows the data in memory. Find *the address* of the variable `'Array'` from the 'Expression' window and type it in the 'Memory Browser' window. Here the address is 0x825C.

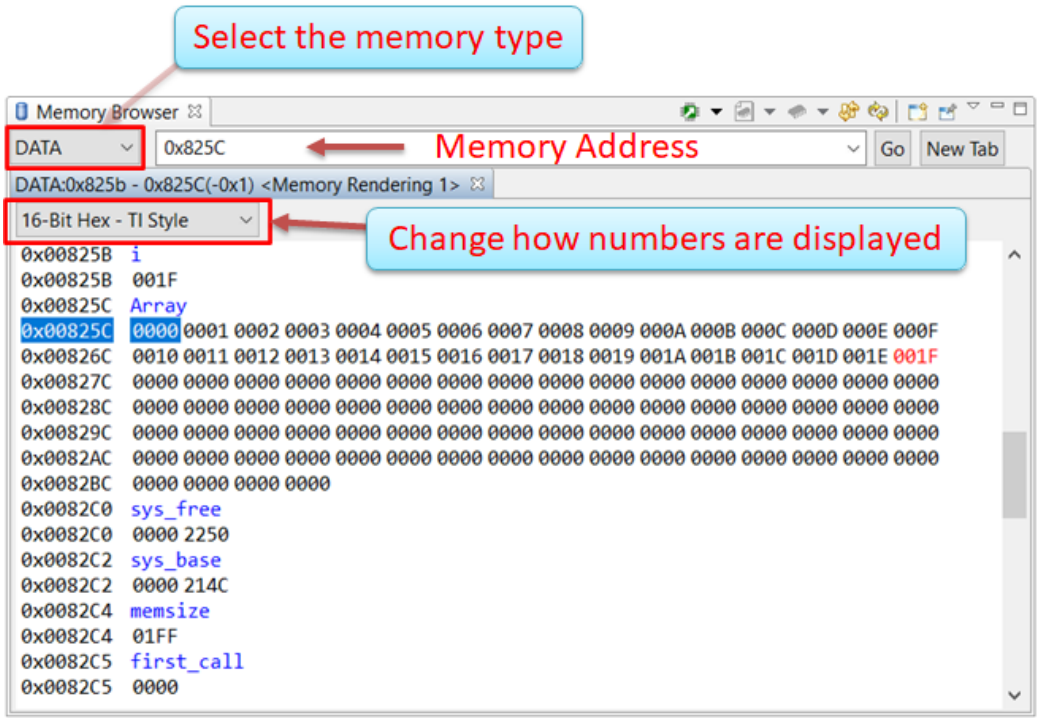


Figure 25: Memory view

Some processors have multiple addressable memory spaces. The memory space is explained in more detail in the future chapters. The C55xx family has three memory spaces: 'Program', 'Data', and 'IO'. When entering an address, the memory type for that address should also be specified. Because variables are in data space, the memory type is set to 'Data'.

12) Plot data in memory with Graph

Double click on breakpoint symbol ( ) to remove it. Then add a new breakpoint to the last line of the code (i.e., 'return 0;'). Run the code (F8) until it reaches the end of the 'main()' function. This will update the contents of the variable `'Array'`.

From the 'Tool' menu, select 'Graph' and then 'Single Time'. Inside the Graph properties window, change the setting as shown in the next figure. The goal is to plot the variable `'Array'`.

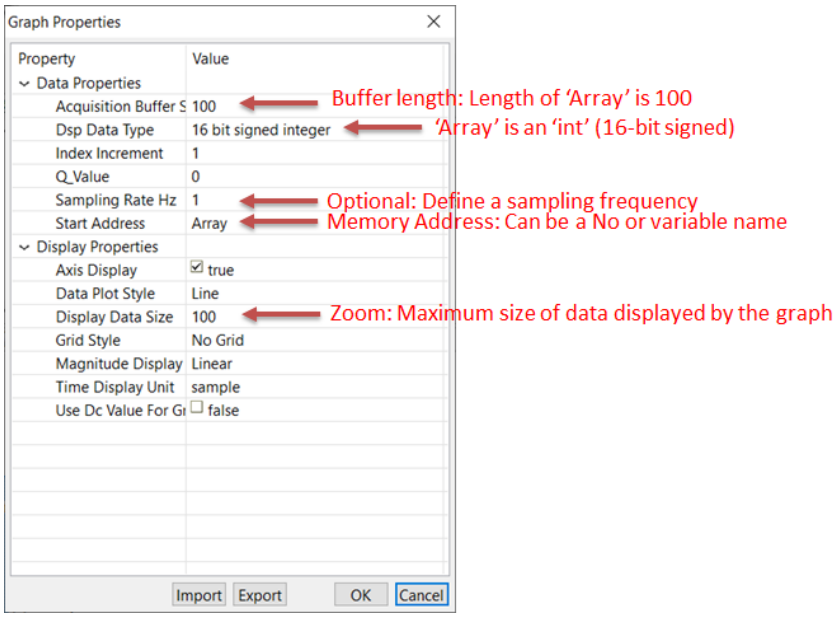


Figure 26: Graph properties window displays the content of variable 'Array'

After selecting OK, the next plot window is opened.

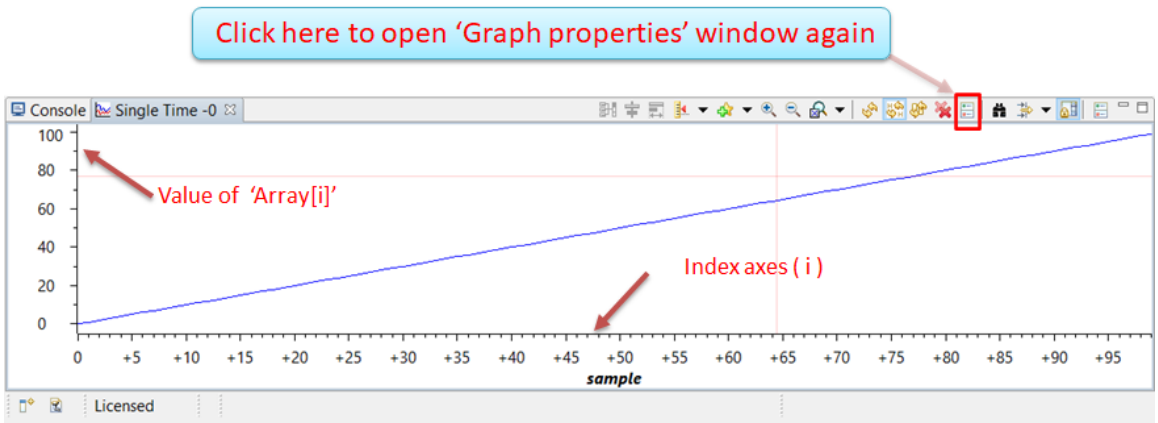


Figure 27: Plot data

Exercise 4: *The Graph window assumes that the data is coming from an Analog to Digital Converter (ADC) and then stored in an acquisition buffer. So it has more settings for sampling and frequency analysis. Use the 'Tool' menu→ 'Graph' → 'FFT Magnitude' to plot the FFT of 'Array[]'.*

13) Display an image

Add the code shown here to the current code in the 'main.c' file.

```
#include <stdio.h>
int i=10;
int Array[100];
unsigned short   Image[128][128];
int j;
int main(void) {
    int h=9;
    printf("Hello ...");
    for(i=0;i<100;i++)
    {
       Array[i]=i;
    }
   for( i = 0 ; i < 128 ; i++ )
   {
      for( j = 0 ; j < 128 ; j++)
      {
         if ( i == j )
         {
            Image[ i ][ j ] = 255 ;
         }
         else
         {
            Image[ i ][ j ] = 2*i;
         }
      }
   }

    return 0;
}
```

Build, load, and run the code (+). The `'Image'` array is a 2-dimensional grayscale image. While in the 'debug' mode, from the 'Tool' menu, select 'Image Analyzer'. Then two windows are opened: 'Image' and 'Properties'.

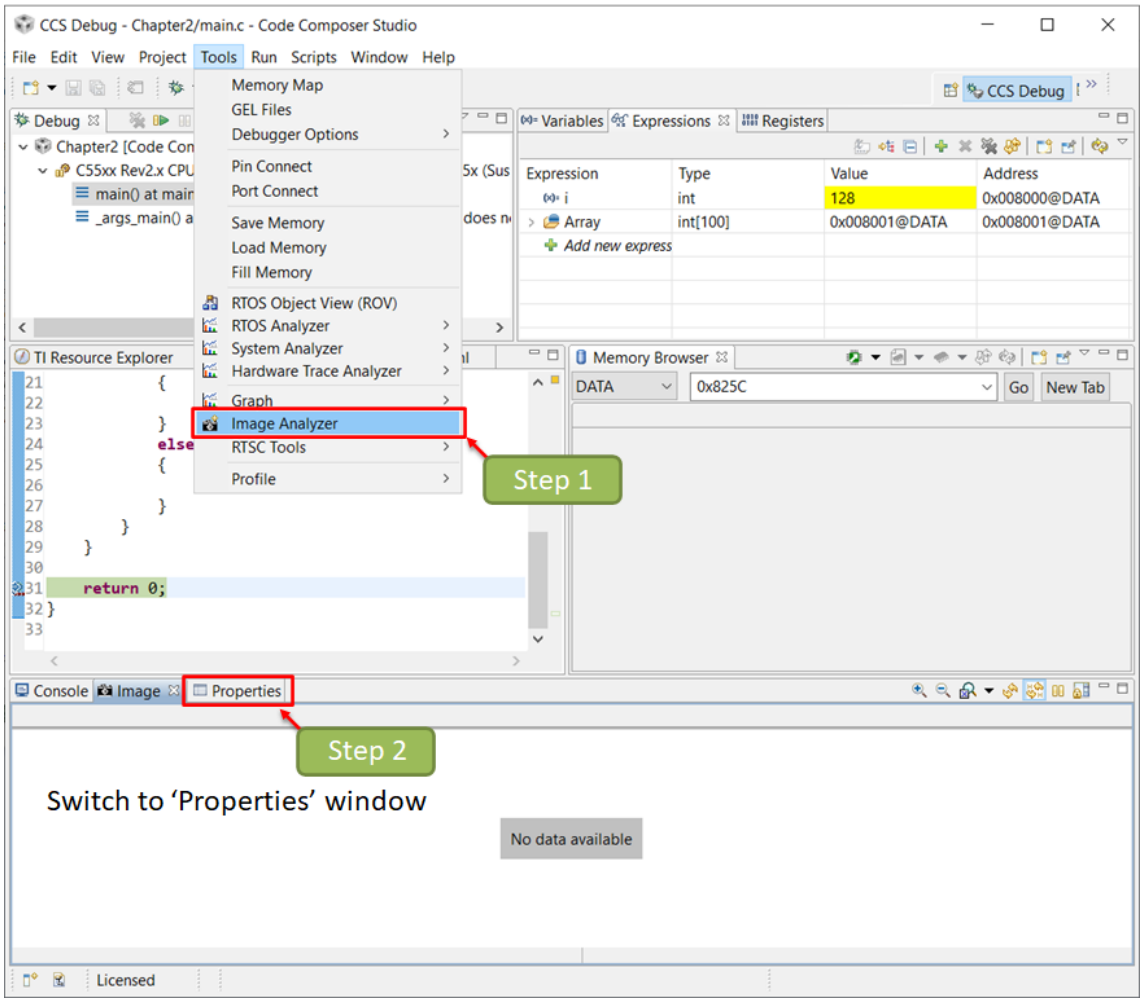


Figure 28: Switch to the 'Properties ' window to apply the image parameters

Switch to the 'Properties' window (if the 'Properties' window was not opened, right-click inside the 'Image' window and then select 'Properties'). The 'Properties' window always shows the properties of the selected item. For example, if a file is clicked, the 'Properties' window displays the file's properties. Therefore make sure before switching to the 'Properties' window, click inside the 'Image' window to select it.

Inside the Image properties, select the 'Image format', and from the drop-down menu, select 'RGB' for the image format. Now, the property for the RGB image is displayed. Change the settings as shown in the next picture.

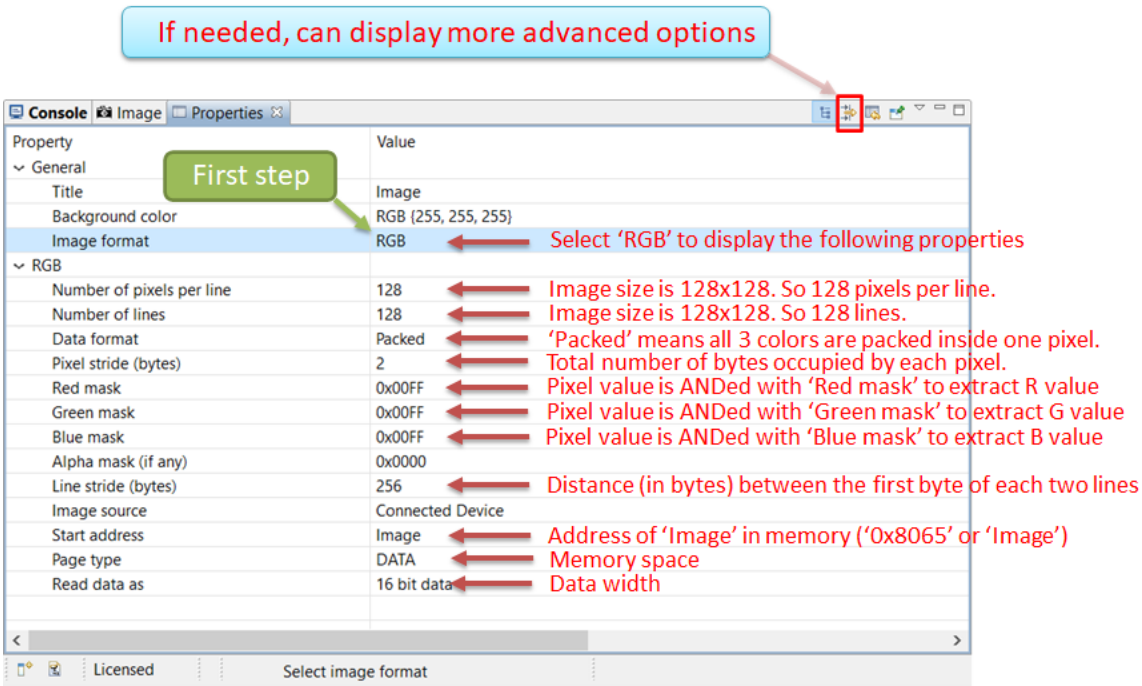


Figure 29: Image properties

Now switch back on the 'Image' window and then right-click on the page, and then select 'Refresh'. Now the 'Image' window should show the next image. *In older versions of CCS (such as 'CCSv5.5'), sometimes, the software fails to display the image. This is a bug and is fixed in newer versions* (Closing CCS and reopening the software may help).

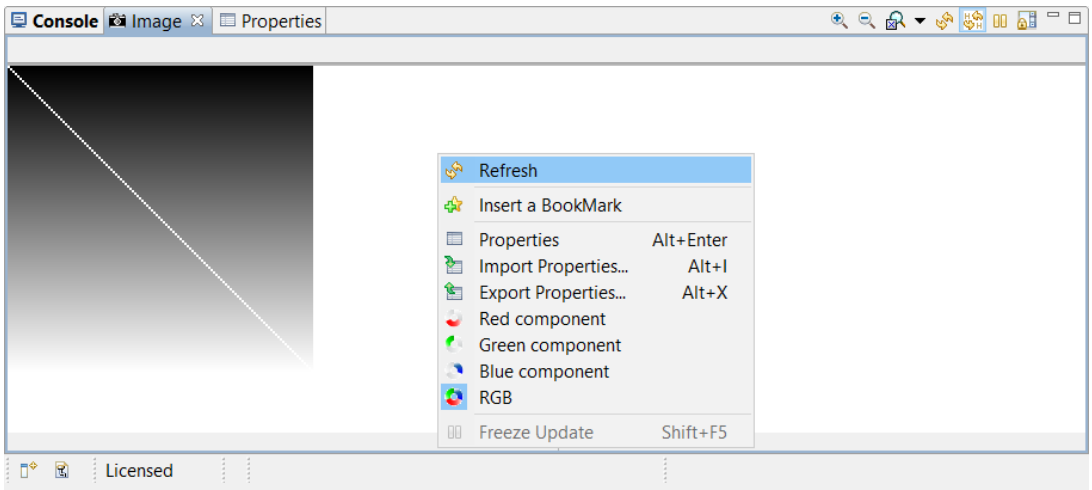


Figure 30: Grayscale 2-D 'Image[][]' array

14) CCS in more detail

Some of the important CCS features are reviewed in this section.

14.1) RTS file

The ‘RTS’ file is one of the files that exists in every C project. The ‘RTS’ has native C functions which the user code can call. For example, ‘fopen()’, ‘scanf()’, ‘printf()’, ‘memcpy()’, and all other standard C functions that are available as part of the standard C are in the ‘RTS’ library. The CCS has already added the RTS file to the project. Right-click on the project name and select properties. The ‘Properties’ window opens (next figure). From the ‘Resource’, select ‘General’ to see the ‘Runtime support library’. Do not change anything at this time. The RTS file is discussed later with more details in Chapter .

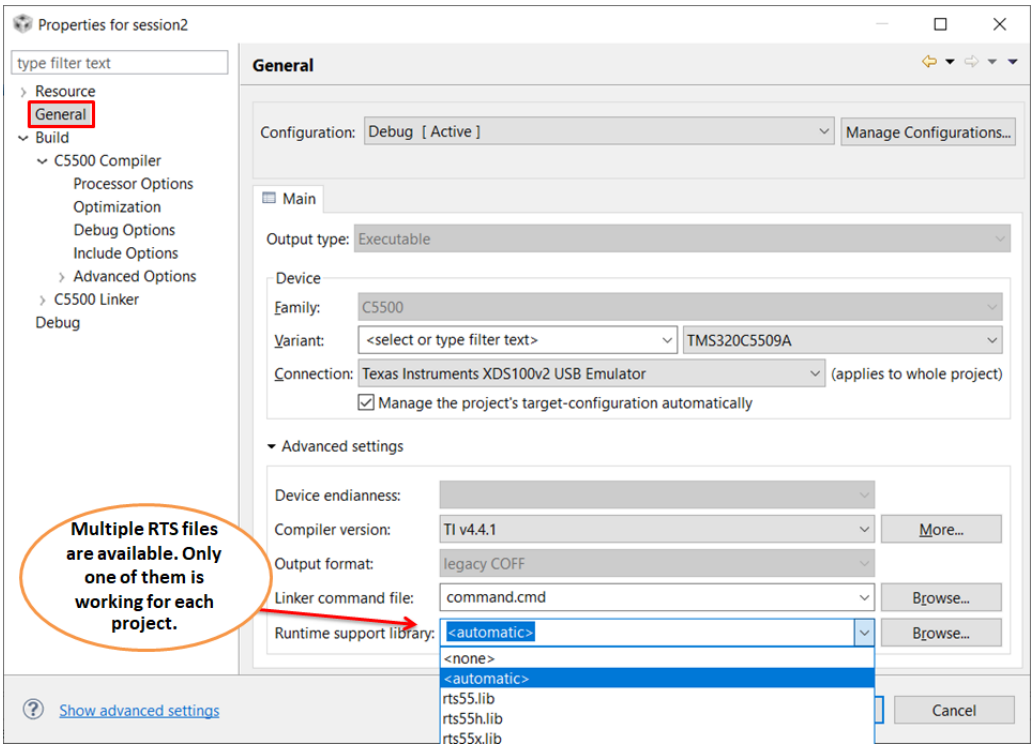


Figure 31: The ‘General’ page shows the default project settings

14.2) Basic of file management in CCS

In many IDEs, the files should be added to the project. In the older version of CCS (i.e., ‘CCSv3.3’), the ‘Add a file to the project’ menu is used to add files to the project. But in Eclipse-

based IDEs (such as 'CCSv5.5'), there is no need to add files. If a file is copied into the project folder in the current workspace, it is automatically considered part of the project and compiled along with other project files. This can cause a problem because CCS compiles all the files in the project directory by default. Thus if a file is not part of the project, it should be excluded from the compilation. To exclude a file in 'CCSv5.5', right-click on the file name, and then from the 'Resource Configuration', select 'Exclude from Build' (Do NOT exclude any file from the current project). In 'CCSv10.2', right-click on the file name and then select 'Exclude from Build'. After excluding a file, its icon is changed().

Figure 32: Excluding a file from the build

Another way to add a file to the project is to link it to the project. Right-click on the project name and select the 'Add Files'. After selecting the files, a window is opened (next figure). If the 'Link to files' option is selected, the file is added to the project without being copied to the project

folder. This has many benefits, especially when adding a file that is part of a bigger package. If a file is copied to the project folder, the compiler may not find the header files (‘*.h’) used in the file.

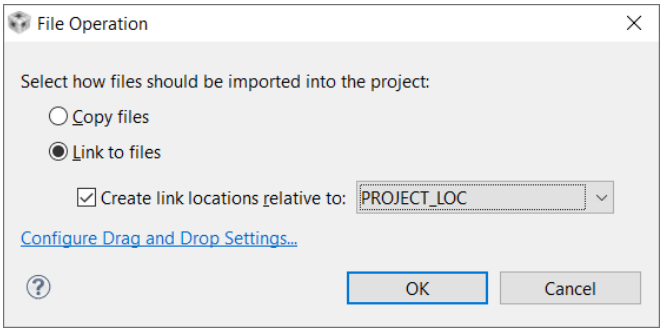


Figure 33: ‘Link to files’ instead of ‘Copy files’ to the project.

14.3) Shortcut keys

There are many shortcut keys for different CCS functions (such as F7 and F8). From the ‘Window’ menu, select ‘Preference’ to open the Preference window. The shortcut keys can be changed from ‘General’→ ‘Keys’ (next figure). Spend a few minutes in this window and go over different settings such as ‘C/C++’ or ‘Run/Debug’.

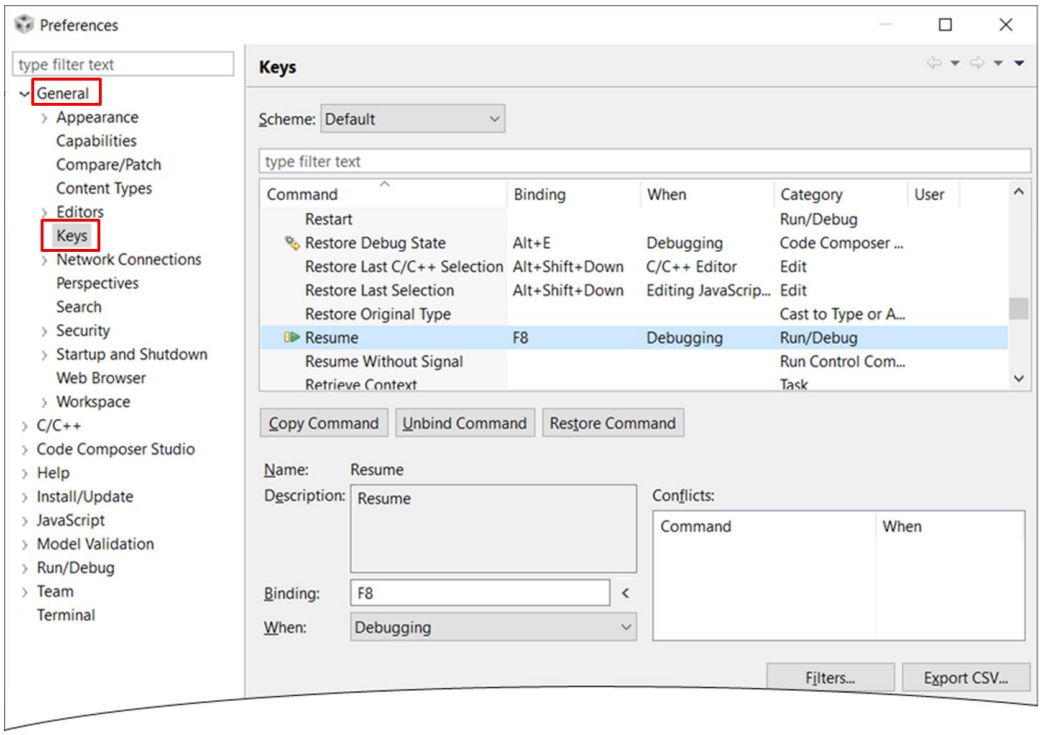


Figure 34: General settings for CCS can be changed in the ‘Preferences’ window

14.4) Review workspace again.

In many new processors, there are multiple cores. Also, modern hardware has multiple processors (instead of a single chip). When designing hardware, instead of having a separate JTAG connector for every processor, it is better to use a single JTAG connector for all the processors. As a result, the CCS software can load the code for each processor through the same JTAG emulator (by pressing ). Some debugging tools are only available when a single JTAG is used for multiple cores. For example, it is possible to create a global breakpoint so that when one of the cores reaches a specific line, the other cores stop executing at the same time. For each core, there is a project in the workspace.

Also, several other configurations depend on the workspace. For instance, the 'Preferences' window (inside the 'Window' menu) can change the shortcut keys' functions (for example, select F5 for RUN like old 'CCSv3.3'). All of these configurations are now transferred as the workspace is moved from one computer to another.

Accordingly, there is no option to open a project. Instead of opening a project, the project needs to be imported to the current workspace. There are two types of projects: the old (legacy) projects from 'CCSv3.3' and the new project based on the Eclipse IDE. Both of these projects can be imported from the 'Project' menu.

14.5) Relocation of the project using WORKSPACE_LOC or PROJECT_LOC

The easiest solution to move a project is to move the entire workspace directory. At startup, CCS asks for the workspace address, which can be used to switch to a new workspace. The workspace can be changed from the 'File' menu by selecting the 'Switch Workspace' option. It is always recommended to create the project in the workspace directory, otherwise, there may be problems after moving the workspace.

Another useful feature is 'Path Variables'. Path variables are defined by CCS for a specific directory in the workspace. For example, WORKSPACE_LOC is a fixed text that always points to the current workspace location. Whenever the workspace directory is moved from one computer to a new computer, the value of WORKSPACE_LOC is automatically updated. The list of these variables can be found in the project 'Properties' window, inside the 'Link Resource' page (the following figure). These variables (or Path Variables) can be used to address a file or include a search path.

These variables have been used in different parts of the software. For example, while adding a new file (figure 33 on page 43), the recommendation is using the ‘Link file to’ option (instead of ‘Copy to’). When linking a file to the project, it is possible to use PROJECT_LOC or WORKSPACE_LOC variables to address the file relative to the project's location. For instance, suppose the workspace directory is in the `‘C:\a\b\w’` folder, and the files used are in the `‘C:\a\b\f’` folder.

Then the file address can be added in two ways:

Method 1: `C:\a\b\k` ← (not recommended)
Method 2: `WORKSPACE_LOC\..\k` ← (recommended relative addressing) [19]

Here the WORKSPACE_LOC is `‘C:\a\b\w’`, thus, the file address would be `‘C:a\b\w\..\k’`.

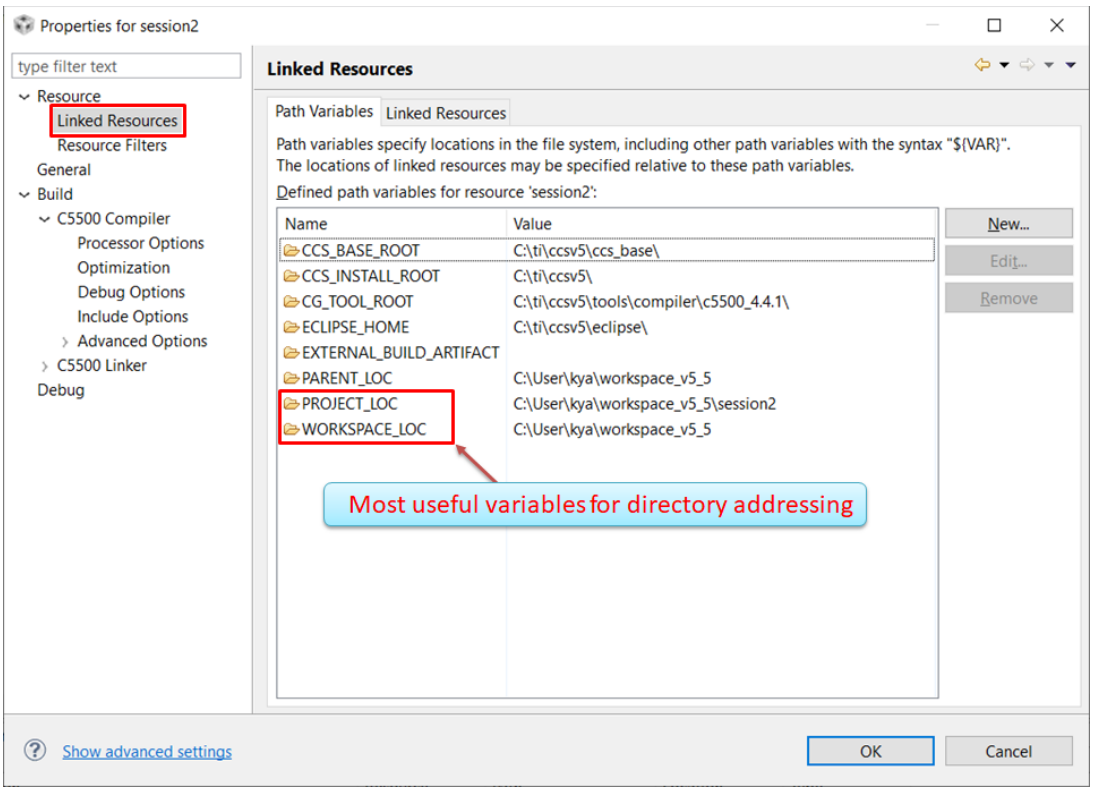


Figure 35: Variables for directory addressing

[19] - The two dots (..) used to go to the parent directory.

Chapter 2

A BRIEF OVERVIEW OF ASSEMBLY PROGRAMMING

Kyanoosh Shafaei
www.DSPgig.com

1) Introduction

The CCS compiler can compile C and assembly code simultaneously. The assembly code should always be in a file with a '.asm' extension (for example, 'init.asm') and then be added to the project. In many projects, at least a small part of the code is in assembly. In this chapter, the very basics of assembly coding are reviewed.

Since the assembly instructions differ between TI processors, after a general overview, a few examples for the C55xx family[1] are studied.

2) Introduction of internal registers

Internal DSP registers can be divided into two main categories:

1- Computational registers.
2- Addressing registers.

2.1) Computational registers

Accumulator ('ACC') register is the main register used for multiplication and addition. ACC is between 32 bits to 64 bits (depending on the processor type). When a processor has multiple accumulators, their names are AC0, AC1, and so on.

[1] - Check table 1 at the end of this chapter for the list of available families and their part numbers.

Another computational register is the 'T' register, which is used primarily for multiplications in some processors.

2.2) Addressing registers

There are at least two main addressing methods in most DSP processors:

1) Direct Addressing
2) Indirect Addressing

2.2.1) Direct addressing

With the direct addressing, the memory address is directly used in the instructions. Whenever a memory address is used in the instruction, then the addressing is called *direct*. Conversely, when registers are used for addressing, the addressing is called *indirect*.

In DSPs, memories are divided into multiple pages. For example, in the C54xx series, the total memory space (64KB) is divided into 2^9 pages (each page is 128 bytes). To access a memory, the page number (9 MSB bit of the address) is loaded to the Data Page pointer ('DP'), then the offset (7 LSB of the address) is used directly in the instruction. In the C55xx, the DP register width is increased from 9-bit to 23-bit, and its name is changed to XDP. The XDP points to any location in memory, and the 7-bit memory offset is added to the XDP register.

In the next picture, the address of a memory is entered in the XDP, and then the 7-bit offset is used to address up to 128 bytes of memory.[2]

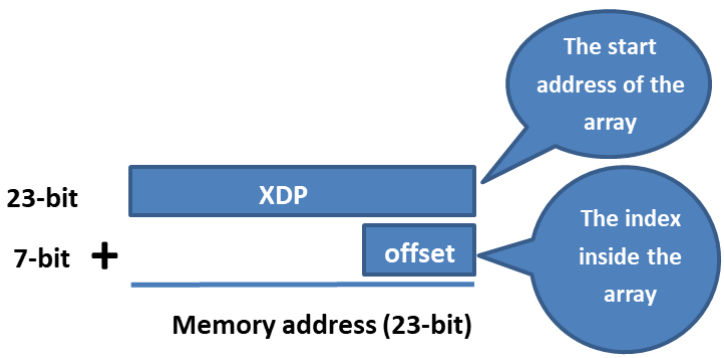


Figure 1: C55xx direct addressing using XDP register and offset from instruction

[2] - Understanding the DP addressing is important when reading chapter 12 'memory model'.

Chapter 3

COMMAND FILE

Kyanoosh Shafaei
www.DSPgig.com

This chapter includes one of the most important and basic concepts of embedded DSPs. The command file has an important role in working with all the DSPs. Throughout the book, the command file is used in many chapters, so it is important to understand it.

1) Introduction

If you have done assembly programming with other types of processors, you have probably come across the 'ORG'[1] directive. The 'ORG' is a method to specify the location of the data in the memory. The 'ORG' format is:

```
ORG         <address>;
```

*The 'ORG' can **not** be used for TI-DSPs[2] in CCS.*

This command[3] specifies the address of the code following it. Wherever the 'ORG' is used, all the assembly code after it is placed in the address specified by the 'ORG'. For example, for an arbitrary processor:

```
ORG    1000H
MOV    #2,AC;
  .
  .
  .                  → A
ORG    2000H
MOV    #5,AC;
  .
  .
  .                  → B
```

[1] - 'ORG' is an abbreviation for Origin and means the start address.
[2] - Texas Instrument (www.ti.com) Digital Signal Processor
[3] - 'ORG' is a linker directive and should not be mistaken with assembly instructions. It just guides the linker to place the codes followed it in the correct location in memory.

Here, the code is divided into two parts of A and B. Part A starts from address 1000H, and part B starts from 2000H. The 'ORG' is a simple method to place the code in the desired memory location. However TI-DSP, instead of 'ORG', uses another approach that is a little more complex but more useful.

2) What is the Section?

The TI-DSP uses '.sect'[4] instead of 'ORG' for placing the codes in the memory. While the 'ORG' method places the codes in the memory only in one step, the method used by TI has two steps. For example, 'ORG 100H' directs the linker to use the area starting from address 100H. But, the TI method is performed in two steps:

Step 1: In this step, inside the assembly file, a name is selected for the target code:

```
   .sect    "name1"
```

This line indicates that a name is assigned for the code section after '.sect'. Here, '*name1'* is an arbitrary name, and any other name can be used in its place. All the assembly codes after this line will be identified with the name of '*name1'*. The '*name1'* is referred to as the *section name*.

Note 1: '*.sect'* always starts with a dot in the beginning.

Note 2: The '*.sect'* could only be used in the assembly files.

Step 2: After naming each section of the code inside the assembly file, the exact location of each section is specified in another file called the command file (with the '.cmd' extension). In this example, the location of '*name1'* should be specified in the command file. In the following, a sample command file is shown.

Code 1: A sample command file with '.cmd' extension (e.g. 'memory.cmd')

```
MEMORY
{
   PAGE  0:
      new_name : origin = 1000H, length = 500H
}
SECTIONS
{
       name1    :   {}   >      new_name
}
```

[4] - '.sect' is abbreviation for 'section'.

Chapter 4

SECTIONS TYPES

Kyanoosh Shafaei
www.DSPgig.com

1) Introduction to map file

Until now, no custom command file was added to the previous projects. In this chapter, the command file and its effect on an actual project are studied. In the older version (CCSv3.3), the command file is added to the project manually (like the RTS file). In the new version of CCS, in addition to the proper RTS file, a default command file is added to the project[1].

Open the same version of CCS used in chapter 1 ('CCSv5.5' or 'CCSv10.2'). Reopen one of the projects built for the C55xx series in previous chapters (for opening the project, select the workspace folder of that project as the active workspace)[2]. By default, CCS opens the previous workspace in the 'workspace launcher' when it opens. Or create a new project for TMS320C5509A.

Type the following simple program in the 'main.c' file.

```
int    i;
void   main( )
{
   i= 0;
}
```

Then build[3] the project.

---

[1] - Note that in some versions of CCS, the default '.cmd' files have some problems.

[2] - There are two ways to open a project:
A- Select the workspace of that project as the active workspace: from the 'File' menu, select 'Switch workspace' and then browse to the workspace folder. This will open all the projects inside that workspace.
B- Importing the project from another workspace folder into the current workspace: The 'Import' option is accessible from the 'File' menu. During importing the project, if the copy option (instead of 'Link to') is selected, the project is copied into the workspace folder with the same name as the project. 'Import' is the only way to open an old project.

[3] - The compilation of all files and then linking them into one final '.out' file is called 'build'. For building, select the 'Build all' from the 'Project' menu.

As seen in the previous chapters, each project has a separate folder in the workspace folder. After building, if there is no error, a folder named '*debug'* is generated in the project folder. Check this folder for the two files with the '.out'[4] and '.map'.

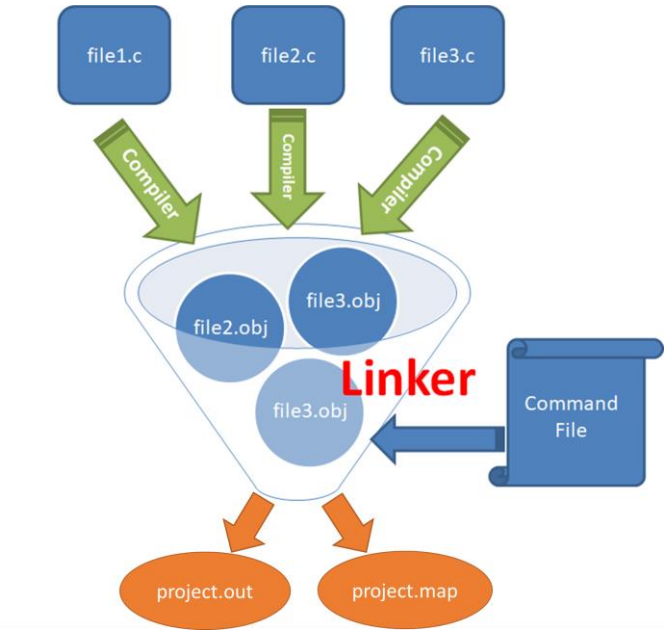


Figure 1: map file is one of the linker's output files

Open the '.map' file using the CCS or any text viewer software (like Notepad). The '.map' file contains the detailed memory information of the project. In the map file, a list of all the sections used in the project is shown. The map file is generated automatically by the linker and contains comprehensive system memory and project file information. Here, the first part of the map file is shown (the numbers/addresses in this file may differ from the version on your computer).

Code 1: The '.map' file containing the system memory map

```
******************************************************************************
              TMS320C55x Linker PC v4.4.1
******************************************************************************
>> Linked Mon Jul 21 20:50:39 2014
```

[4]- The '.out' file is the final binary file that is loaded into the processor memory by the CCS. In this file, in addition to the code and data which should be saved in the memory, the names of the variables and functions are presented. So before loading the '.out' file, the CCS breaks it into two parts: Part one is the binary information that is loaded into the processor memory. Part two is the debug information (like variable names and their address) which is kept by CCS inside the computer and later can be used for debugging purposes. This is how CCS can do a C-code debugging or show a variable value by simply checking the Program Counter (PC) or memory.

Chapter 5

PROCESSOR STRUCTURE AND THE COMMAND FILE

Kyanoosh Shafaei
www.DSPgig.com

1) Introduction

The command file is the method that TI uses for memory management. The command file structure depends on the hardware and software specification, and it is possible to divide the available internal and external memory into hundreds of ways. Two factors affect the command file structure:

1- The processor type (internal memory)
2- The hardware (peripherals+external memory) specification

In this chapter, the effect of the processor type on the command file is studied. It assumes that the hardware only has a DSP and does not have any other external memories.

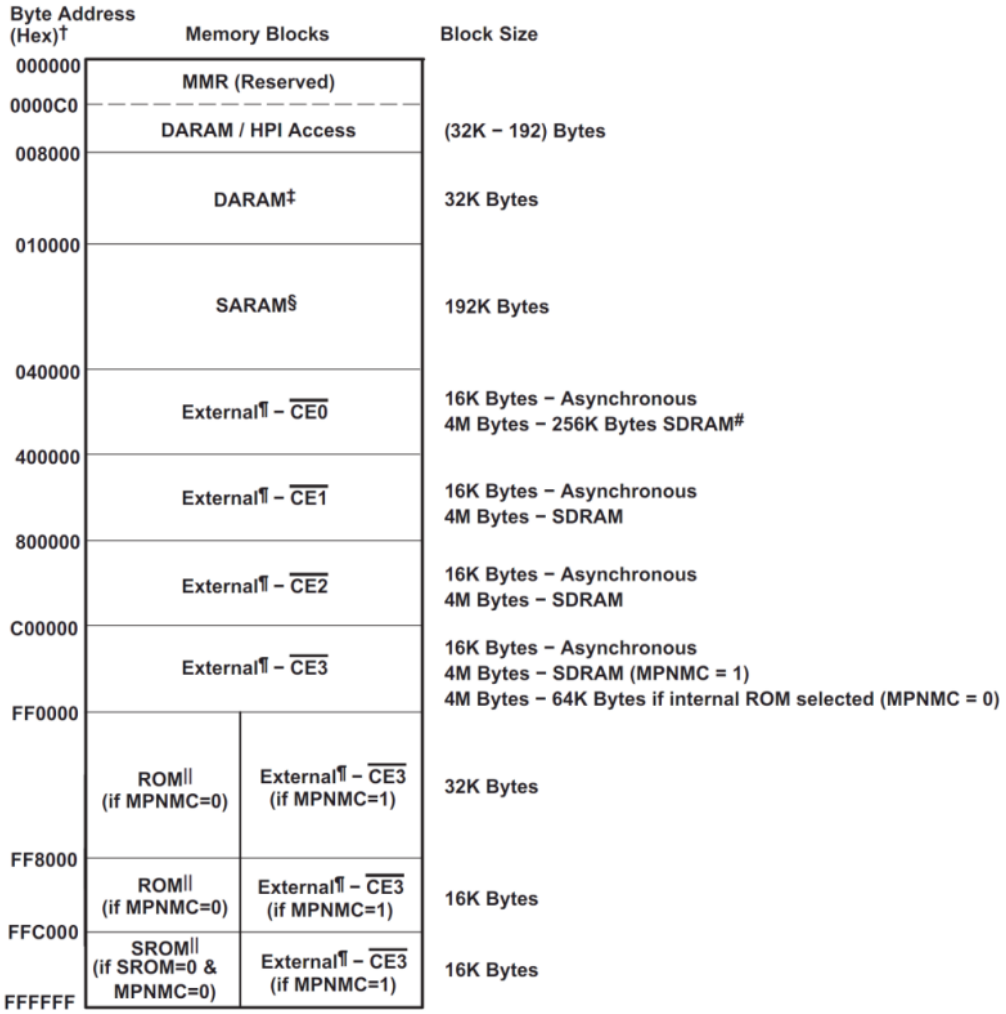
Next, a few TI processors are reviewed, and hypothetical command files are suggested for them.

2) TMS320VC5509A[1] Processor

The C55xx family is one of the best-selling processors manufactured by TI. In the C55xx family, the TMS320VC5509A is one of the most capable processors. TI recently replaced this processor with TMS320C5517. The 5509 and 5517 are almost identical, with few improvements in 5517. But 5517 has a tiny BGA package (196 pins in 10mm by 10mm package), so hardware debugging is limited and may not be suitable for some experimental projects. Also, the 5509 is probably the most famous processor in the C55xx family, and there are numerous examples and free codes available for it. That is why 5509 is selected as the first example for this chapter.

[1] - Whenever there is a hardware bug in a specific version of TI processors, the company adds an ‘A’ or ‘B’ to the end of the part number for the new version.

The clock speed for the 5509 processor is 200MHz, and it can perform up to 400 million mathematical calculations per second. In addition to MAC instruction (for FIR filters), the processor also has dedicated instructions for calculating adaptive filters (LMS instructions). The internal memory of the processor is 256KB. Since the C55xx series has a 16-bit data space, the processor's internal memory is practically 128KWord.



† Address shown represents the first byte address in each block.
‡ Dual-access RAM (DARAM): two accesses per cycle per block, 8 blocks of 8K bytes.
§ Single-access RAM (SARAM): one access per cycle per block, 24 blocks of 8K bytes.
¶ External memory spaces are selected by the chip-enable signal shown ($\overline{CE}[0:3]$). Supported memory types include: asynchronous static RAM (SRAM) and synchronous DRAM (SDRAM).
# The minus 256K bytes consists of 32K-byte DARAM/HPI access, 32K-byte DARAM, and 192K-byte SARAM.
|| Read-only memory (ROM): one access every two cycles, two blocks of 32K bytes.

Figure 3−3. TMS320VC5509 Memory Map (PGE Package)

Figure 1: The memory map section from the 5509 processor datasheet

Chapter 6

INITIALIZATION

Kyanoosh Shafaei
www.DSPgig.com

1) Introduction:

Every time the computer turns on, it starts from a black screen (BIOS). During the early stage, the PC checks the computer hardware, including RAM, graphics card, hard drive, and CD/DVD-ROM. However, where does this code come from? If the hard drive is detached, still the black screen (BIOS) is shown, and it does its job. So the BIOS code is not loaded from the hard drive. It also means the operating system (such as Linux or Windows) has no role on this screen. But there should be a program inside the computer. This program is inside a separate flash memory and is called 'BIOS'. The BIOS is written by the manufacturer of the computer's 'MotherBoard'. After checking the computer hardware, BIOS loads the operating system from the hard drive into RAM and then runs the operating system (by jumping from BIOS code to the operating system code).

The same stages exist in most DSP systems. On a PC, if multiple hard drives are attached, the BIOS uses the proper hard drive (based on the user setting) and looks for the operating system on that hard drive. Similarly, in DSP, the user needs to specify how the system should load the code into memory. Since most DSPs (except the 2000 family) do not have an internal flash, the user code should be stored in an external nonvolatile memory and then loaded into an internal memory before the execution. This task is done by a code running from an internal ROM, called 'bootloader'. The 'bootloader' is like a BIOS. It reads the code from an external flash or EEPROM[1] and then loads it into the memory.

[1] - Electrically Erasable Programmable Read-Only Memory whose contents can be erased and programmed using a voltage.

Some BIOSs support booting from Ethernet. Some DSP ROMs also support loading the code from Ethernet or UART2. So there are multiple options for DSP to load the code. In this chapter and chapter 16 ('Bootloading the processor without JTAG'), we learn how the bootloader works.

2) Another example: a microcontroller

For further explanation, first, the 'Initialization' process for a typical microcontroller such as 'ATMEL' microcontrollers (i.e., ATMEGAs) will be examined. These micros have a RAM and an internal flash memory. Assuming the following code is written for an 'ATMEGA128'[3]:

```
int      M;
int      K=2;
void  main()
{
   int   i;
   for (i=K; i<100; i++)
   {
     …
   }
}
```

This code uses three types of variables:

2.1) Global variables without initial value such as M:

Assume that `'M'` should be stored at the address 0x100. If the code had `'M = 6;'`, it is translated into an assembly instruction that stores number 6 at address 0x100.

But before `'M=6;'` there is another line: `'int M;'`. What does `'int M;'` mean? Does it translate to a specific assembly instruction? The answer is false. The `'int M;'` does not generate any code in the program. However, it informs the compiler that if it comes across the word `'M'` somewhere in the code, it means address 0x100.

[2] - Universal Asynchronous Receiver-Transmitter is a serial protocol like SPI (but with no clock signal).
[3] - ATMEGA128 is one of the famous microcontrollers made by ATMEL and has a maximum operating frequency of 16MHz. It also has 4KB SRAM, 128KB flash, and 4KB of E2PROM memory.

Chapter 7

A PRACTICAL APPROACH TO INITIALIZATION

Kyanoosh Shafaei
www.DSPgig.com

1) Creating a new project

This chapter is about practicing and reviewing previous chapters. Even though this book provides pictures for most steps, it is important to use the software and practice it!

In the 'CCSv5.5' software[1], create a new project by selecting 'New CCS Project' from the 'Project' menu. On the 'New CCS Project' window, choose one of the processors from the C66xx family. In this chapter, one of the most powerful DSP processors ('TMS320C6678') has been selected. The TMS320C6678 can process more than 320 billion calculations per second. With eight parallel cores, this processor can be used in a variety of signal processing applications. Therefore, it is recommended to select the 'TMS320C6657' processor in the 'New CCS Project' window (since the 6678 processor does not exist in 'CCSv5.5', a closer alternative from the same family such as the 6657 can be selected). Figure 1 (next page) outlines the required settings. In this figure, 'XDS100v2' has been selected as the 'Connection', and also, instead of TMS320C6678, the TMS320C6657 is selected.

After creating a new project, because 6657 is chosen instead of 6678, the project settings should be changed. The closer the selected processor's internal structure to the structure of the desired processor, the fewer the number of changes required in the project settings. The 6657 benefits from the same type of core that 6678 has, but 6657 has only two cores instead of the eight cores.

[1] - Use 'CCSv5.5' because only 'CCSv5.5' supports Simulators. Newer versions of CCS do not support Simulators.

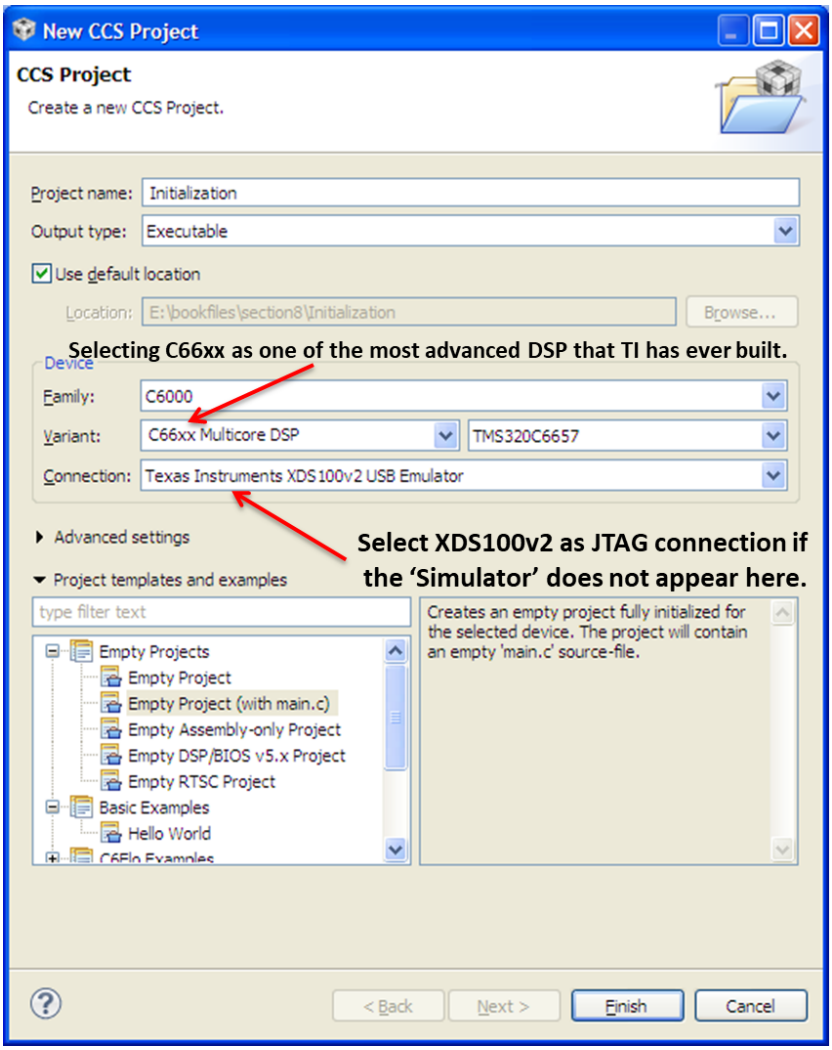


Figure 1: Creating a new project for the C66xx series

Within the newly created project, there are two files:

1) A file with a '.c' extension (which contains the `'main()'` function).
2) A file with a '.ccxml' extension.

Open the '.ccxml' file. Since a board with a 6678 processing core may not be readily available, it is best to use a simulator in the software to complete the exercises in this chapter. Open the '.ccxml' file and modify it, as shown in the following figure. Select 'Texas Instrument Simulator' from the 'Connection' dropdown menu and 'C6678 Device Cycle Approximate Simulator, Little Endian' from the 'Board or Device' options. The concept of 'Little Endian' and 'Big Endian' is discussed in chapter 12 ('Memory models').

Chapter 8

MIXED C AND ASSEMBLY PROGRAMMING

Kyanoosh Shafaei
www.DSPgig.com

1) Introduction

After designing a DSP hardware, a low-level software driver should be designed for external parts such as ADC[1]. The data from the ADC is read and then saved for future processing. There are multiple ways to connect an ADC to a DSP. The simplest way is using GPIOs. In this method, the ADC output is connected to the General Purpose IO pins (GPIO). On the other hand, some ADCs can be connected to the Address and Data buses and mapped into a memory address. The latter is the preferred method and has the fastest access time[2]. This chapter first assumes an external device, such as an ADC or a DAC[3], is mapped into the memory address space and has a unique address in memory. Then it reviews multiple approaches to read from an external device with a memory map address.

2) Addressing a variable in the memory using direct addressing

Suppose the hardware has an ADC at address 0x3FEDC8. One way to read from the ADC is to use an assembly instruction that reads from the 0x3FEDC8 address. This is a good solution but requires knowledge of assembly programming.

There are other solutions as well. For example, a variable (such as 'AD') can be defined in C, with an absolute address of 0x3FEDC8. Because the variable's address exactly matches the external ADC address, reading from this variable triggers the hardware to read from ADC.

---

[1] - Analog to Digital Converter
[2] - Some ADC can connect directly to a serial port and then a DMA reads the ADC data from the serial port and stores it in memory. This has no overhead for the software.
[3] - Digital to Analog Converter

First, a variable is defined, and its address is initialized to 0x3FEDC8:

```
int     *AD = 0x3FEDC8 ;
```

In C programming, the above line generates a syntax error because a variable's address cannot be defined in C without proper 'Casting'.

What is 'Casting a variable' in C programming?

In C language, a 16-bit variable can be copied into a 32-bit variable. When a 16-bit number is copied into a 32-bit variable, the 16-bit number is copied into LSB, and the 16-bit MSB is filled with the sign bit:

```
short  i16=-1; // i16 is -1 or 0xFFFF
void  func()
{
   int  i32=i16; // i32 is now -1 or 0xFFFFFFFF
}
```

Similarly, a 32-bit variable can be copied into a 16-bit one. But the 16-bit MSB is truncated before the copy.

```
int  i32= -2; // i32  is -2 or 0xFFFFFFFE
void  func()
{
   short  i16=i3 ; // i16 is now -2 0r 0xFFFE
}
```

Here the 32-bit and 16-bit signed numbers are stored in two different locations in memory. The C compiler performs such conversions automatically. But it runs into problems with the address pointers:

```
int  *i32 ;// i32 is a pointer
void  func()
{
unsigned  char i8[4]={0xFF,0xFF,0xFF,0xFE};//i8 is an array of length 4
i32=(int *)i8; //i32 is point to i8 and i32[0] is now 0xFFFFFFFE or -2
```

Chapter 9

PREPARING STANDARD C/C++ CODES FOR CCS

Kyanoosh Shafaei
www.DSPgig.com

A computer is required for this chapter. Also install Visual Studio (from Microsoft.com), version 2005 or later, on your computer.

1) Introduction

DSP processors are usually used in complex projects. In such systems, the DSP, as the most powerful system processor, acts like the system's mastermind and performs multiple independent tasks. Most of the tasks are written with C. The C language has increasingly become more popular and has become a standard language for many software packages. It is a common practice to use a standard C code (that is not specifically designed for DSPs) and run it inside a DSP platform. The C code can be used in CCS with minimal modification while keeping its original structure and test tools. This chapter discusses how to run a 'C/C++' code downloaded from the web(www) on a 'DSP' processor.

1.1) Another example: Simulation

In a complex DSP project, work is usually done by a team of engineers. Designers generally prefer to simulate the project first and then test it on the real hardware. In signal processing, the first stage of design starts with MATLAB software, and after ensuring the algorithm works correctly, the C/C++ program is written based on MATLAB code. Then the output of the MATLAB program is compared with C/C++ code (this process will be reviewed in the next chapter). One of the best platforms to simulate a C/C++ application is 'Microsoft's Visual Studio'[1]. This software creates the fastest and the most standard environment for a C/C++ project. There are many other platforms for C/C++ that can be used effectively, and all of them can be used for this chapter.

[1] - Although Visual Studio compared to some other platforms may seem a bit complicated at first, but it only takes an hour to learn how to write C code in it.

After simulation in Visual Studio and making sure the C/C++ project matches the MATLAB algorithm, the C/C++ code is compiled in the CCS before running on the hardware. Usually, when testing the real system, there are multiple design iterations. The MATLAB code experiences multiple revisions, which needs to be tested in DSP hardware. So the designer needs to go back to simulation mode and test the programs in Visual Studio or modify the MATLAB algorithms. So it is a common practice to go back and forth between MATLAB, Visual Studio, and CCS. Even after the first version of the product, newer versions are frequently released. The continuous update means doing the cycle again, which is revising the mathematical algorithm (in MATLAB), initial testing (in MATLAB), rapid simulation (in Visual Studio), and finally, implementation and final optimization on actual boards (in CCS).

In the debugging stage, hardware tests take the most time. It is best to do as much of the project as possible before the CCS stage to minimize the effort when testing applications on the real board. Designers' most common mistake is to remove the Visual Studio[2] step from the design path. The testing of the C code has two steps: The first step is to test the basic functionality (module by module) and make sure the code performs as intended, and the second step is to test it on the real hardware.

By skipping the Visual Studio simulation phase, all the testing steps are done directly on the real board. When the code is first tested in Visual Studio, if bugs appear later in CCS, the designer most likely needs to look for hardware-related bugs and not any software bugs. The bugs are probably not in the code themselves because the code has already been tested. Testing in Visual Studio makes debugging easier. This is especially true in a large project with many design engineers involved. This chapter introduces a way to ultimately help debug large projects quickly and is focussed only on Visual Studio and CCS. The simulation between MATLAB and Visual Studio will also be examined in the next chapter.

What is stated here is only one of the methods used when managing DSP projects. In practice, much work is done to prepare and test a real project, and this chapter is only an introduction. Years of design experience have taught large companies that if, for example, they spend six months designing and building a system, they should spend another year and a half on testing and debugging it. This is especially true for large complex projects with multiple engineering

[2] - The word ‘Visual’ in Visual Studio has similarity to words like ‘Visual C’ and may create the impression that the goal is to run a program with a ‘Visual’ structure (with ‘menus’, ‘push button’, etc.) on DSP. Throughout this book, the word ‘Visual Studio’ has been used for a platform to execute the standard C without the ‘Visual’ capabilities.

Chapter 10

OPTIMIZED ASSEMBLY MATHEMATICAL FUNCTIONS

Kyanoosh Shafaei
www.DSPgig.com

1) Introduction

DSP processors are famous for their dedicated signal processing and math instructions and became popular very quickly after their introduction in 1976. Nowadays, there are multiple computing software on the market, such as MATLAB or GNU Octave. For example, MATLAB is accepted by universities and research centers as the first stage for designing a signal processing algorithm.

MATLAB can directly generate C code for a few TI DSP boards. The first step to convert MATLAB code to C code is the implementation in SIMULINK[1]. Next, MATLAB generates an equivalent C code for the SIMULINK model. Then, a CCS project is automatically created by MATLAB and generated C files are added to the CCS project.

This method of using MATLAB has advantages and disadvantages. The most important advantage is the simplicity and the rapid prototyping for MATLAB algorithms. This method is useful only in universities and for simulation. Also, the generated code is not optimized because MATLAB either auto-generates or uses pre-build C code for each block (which is usually not optimal for DSPs). The generated C code can be 10 to 100 times slower than the optimized C code for DSPs. Besides, due to MATLAB's C code's complexity, it is difficult to change or optimize this code. The MathWorks company, the owner of MATLAB software, has been working hard to improve the C code, but it is not easy to make optimized functions for thousands of supported processors (with different internal structures). This has led professional designers to use different methods.

TI has developed many optimized C and assembly functions for its DSP family. One of the optimized packages is MathLIB, which is made for floating DSP in the 6000 series. MathLIB replaces many standard C functions, such as ‘sin()’ or ‘cos()’, with an optimized assembly function.

[1] - One of the provided tools in MATLAB software

FastRTS is another similar package designed for fixed-point processors and has optimized functions for simulating floating-point computations. Another widely used library is DSPLIB, a collection of signal processing functions, such as FIR or FFT. These packages are the optimized method to implement a signal processing application in DSP. In this chapter, DSPLIB is used as an example to show how to use a TI library. Then a recommended approach for implementing a sample MATLAB code is shown. To better understand this chapter, it is highly recommended to experiment the chapter's content in the software, else it makes the chapter complex at a certain point.

2) Optimized library

TI has optimized assembly functions for each DSP processor. These are high-speed functions which are using specific DSP instructions. It is always better to use these functions instead of similar C functions because they are much faster. The list of PDF files for DSPLIB is shown in the next table. Note TI does not provide a DSPLIB library for the 2000 families.[2]

Table 1: Assembly functions user manual file name

File Name	Book title
SPRU422.PDF	TMS320C55x DSP Library Programmer's Reference
SPRU518.PDF	TMS320C54x DSP Library Programmer's Reference
SPRU402.PDF	TMS320C62x DSP Library Programmer's Reference
SPRU565.PDF	TMS320C64x DSP Library Programmer's Reference

For example, the list of assembly functions for the C55xx series is given in the following table. The table includes the functions for computing FFT, filtering, adaptive filters, correlations, matrices, and mathematical functions. Many of these functions can also be found in other DSP families.

Table 2: List of 55XX assembly functions

Function Name	Description
void cfft (short *x, ushort nx, type i)	Radix-2 complex forward FFT
void cfft32 (long *x, ushort nx, type i)	32-bit forward complex FFT

[2]- There are a few optimized math functions for processors such as 2812 or 28335, but these functions do not exist for other processors in the 2000 series. These libraries can be downloaded from the processor's product page.
Whenever working with a new TI DSP, a lot of valuable resources can be found in the product webpage, so it is highly recommended to spend at least half a day on the product page and review all the documents and libraries one by one even for a few minutes.

Chapter 11

IMAGE PROCESSING AND CCS

Kyanoosh Shafaei
www.DSPgig.com

1) Introduction

One of the most important uses of DSP processors is image processing. The advancement of technology has made image processing algorithms more and more sophisticated. However, a limited number of basic functions such as filtering are used in most of these algorithms. TI provides IMAGELIB for the basic functions. These functions are often written in assembly for each processor's family and optimized to maximum performance. Because these algorithms are implemented by professional designers, they have the best possible cycle count.

In addition to IMGLIB, another optimized library called VLIB contains about 50 higher-level[1] functions for the 6000 series. For some other 6000 processors with an image co-processor, TI created a library called VICP. STK-MDK is another library for medical signal and image processing. One of the most interesting libraries for image processing is called C6EZFlO. This is a software similar to Simulink (MATLAB) with many useful blocks for some particular processors. Unfortunately, TI stopped[2] working on this software.

These libraries can be downloaded from the TI website. In addition, some may come with training videos. In this chapter, the IMAGELIB library is used as an example. Similar to DSPLIB (introduced in the previous chapter), working with IMAGELIB is not straightforward and needs some changes to the project.

Also, this chapter focuses on more exercise. There are multiple exercises in this chapter that can help review and practice some of the concepts from previous chapters.

---

[1] - The IMAGELIB supports basic functions such as FFT, but the VLIB has more high-level functions such as background extraction.

1.1) Installation image processing library

The IMGLIB should be downloaded from the TI website. Each processor family has its own image processing library (because the assembly instructions are different). To find any library, search the library's name along with the processor name inside the TI website.

After downloading the latest version from the TI website, run the installation file (*.exe). During installation, when asked for an installation folder, install the library in the same folder where the CCS is installed. The next time CCS is executed, it should be able to find the installed library and add it to its internal packages. This may not be necessary for all packages, but CCS can usually find the library and install the packages.

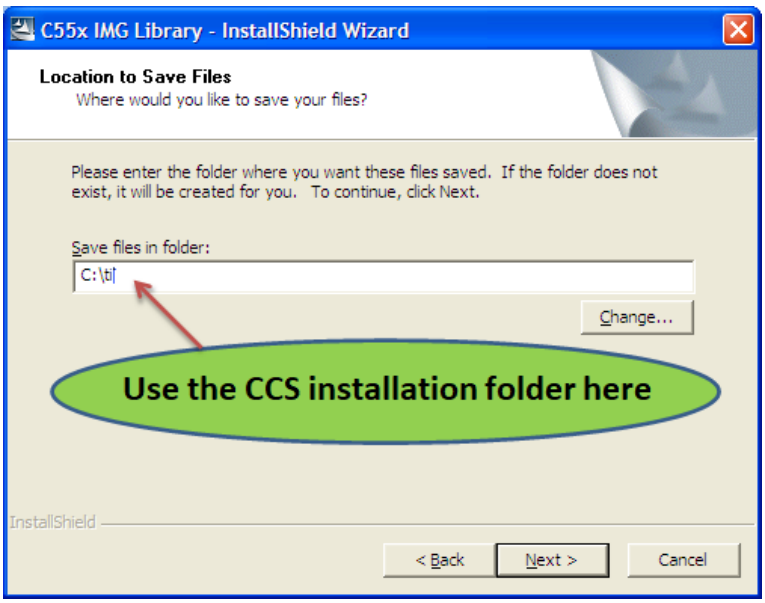


Figure 1: Selecting the CCS software installation address when installing the image processing library

1.2) Transferring the * .h file to the appropriate directory

In the previous chapter, while using DSPLIB, the CCS compiler could not find the library header file ('dsplib.h') by default. Here, the same problem exists. The header file directory address is added to the project inside 'Include search path' in the previous chapter. Here, another approach is introduced.

By default, all the standard header files such as 'stdio.h' are stored in a directory inside the CCS installation folder. TI keeps changing the default location for the header files. However, a simple

Chapter 12

MEMORY MODELS

Kyanoosh Shafaei
www.DSPgig.com

1) Introduction

A project with less than 64KB of memory is compatible with all types of memory models. For bigger projects, some memory models cannot be used. The memory model tells the compiler how to access variables and codes. Some memory models are faster but with some limitations.

All types of memory models and their differences are introduced in this chapter. These memory models have different names for each family. The memory models are called ‘small’ and ‘large’ in the C2000 and C5000 families and ‘near’ and ‘far’ in the C6000 family.

2) Memory models in C2000 and C5000

Both families have two memory models. In the project ‘Properties’ window, from the ‘Compiler’, select the ‘Preprocessor Options’. In the C55xx series, three options can be selected in the ‘Specify Memory Model’ section: ‘small’, ‘large’, and ‘huge’. When changing the memory model to small, another option should also be changed; otherwise, the compiler generates an error. If the small option is selected (for C55xx), the ‘ptrdiff_size’ value must be changed from 32 to 16. ‘ptrdiff_size’ is in ‘Compiler’ → ‘Advance Option’ → ‘Runtime model option’→ ‘specify type size to hold results of pointer math’. The value of this option is 32 when using the large model. In the older version of CCS, when changing the memory model, the value of this option is adjusted automatically. However, in some newer versions of CCS, this must be done manually.

The ‘ptrdiff_size’ specifies the address size. It means that in the small memory model, because the address is 16 bits when performing mathematical calculations such as addition on addresses, the result must remain 16 bits. However, the ‘ptrdiff_size’ must be 32-bit in the large memory model.

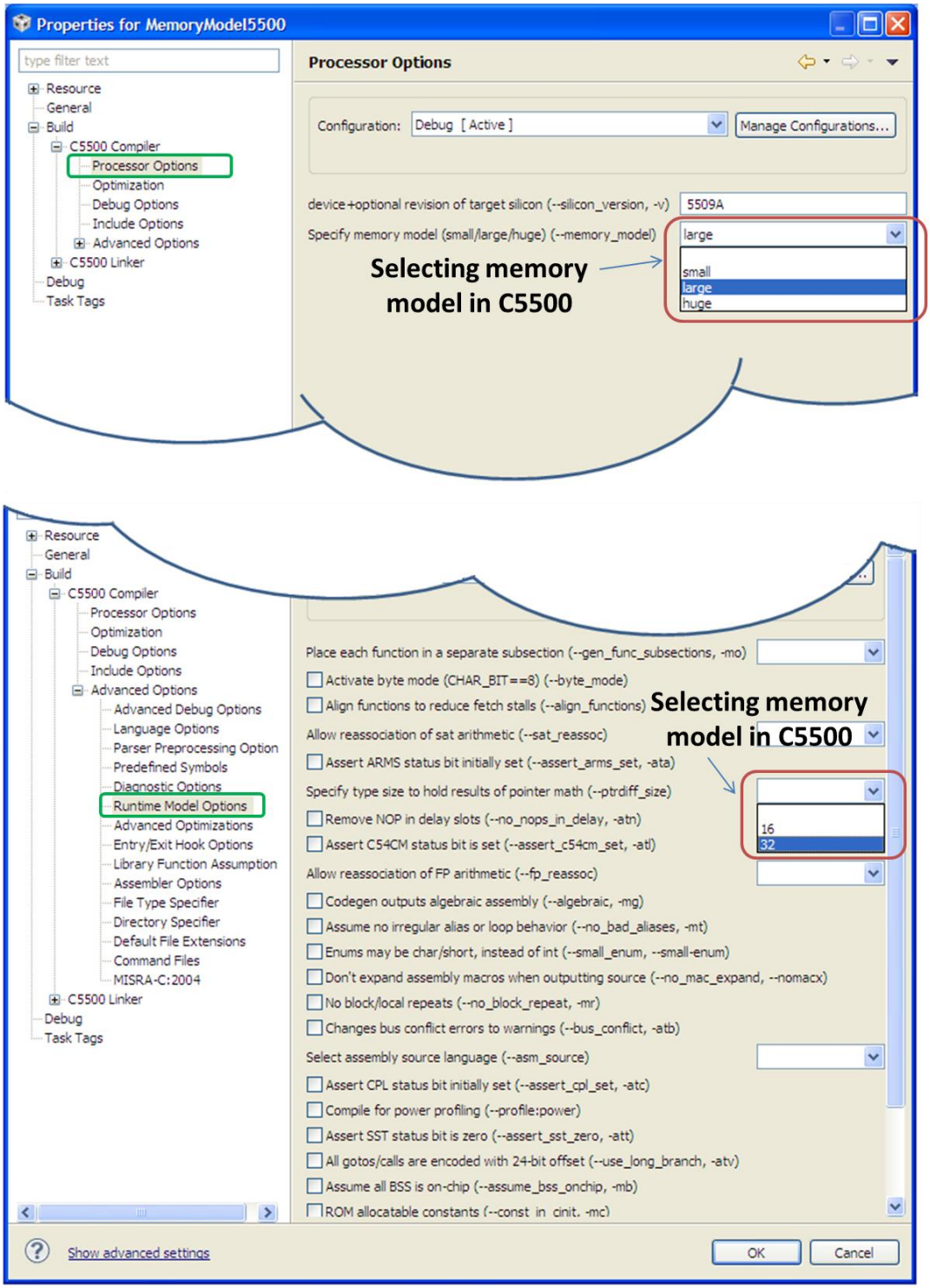


Figure 1: Changing the memory model in the C55xx series

Chapter 13

OPTIMIZATION

Kyanoosh Shafaei
www.DSPgig.com

1) Introduction

The C compiler has multiple levels of optimization. In CCS, by default, the C compiler compiles the code without any significant optimization and generates an '*.obj' file. When the compiler optimizer is turned on, it can effectively optimize the codes. The optimizer improves the '*.obj' file, so the object file consumes fewer CPU cycles or memory. Optimizer is a professional assembly instruction generator that reads the user code and understands it and then generates a very optimized assembly (object file) for the code.

But optimizers sometimes can change the code behavior. Consider the following code:

```
char  simplify(char  j)
{
   char   a = 3;
   char   b = (j*a) + (j*2);
   return  b;
}
```

If optimization is not enabled while compiling, an assembly code for the previous code is generated. But if optimization is enabled, an assembly code for the following code is generated[1]!

```
char  simplify(char  j)
{
   return  j*5;
}
```

[1] - It may seem that the compiler optimizer modifies the C code. However, the C code does not change. The compiler, with a full understanding of the code, optimizes the code like a professional programmer. So it ignores any ineffective code before generating the assembly code.

Since the value of 'a' in the code is equal to 3, the optimizer replaces `'a'` with the number 3. Then, the optimizer replaces `'j*3 + j*2'` with `'j*5'`. The CCS compiler can replace the function body with `'j*5'`, but not all compilers can understand the code intelligently.

Optimizing this code may seem simple, however the compiler can effectively optimize more complex codes. In this example, the optimizer has two direct effects on the code:

1- In many cases, it reduces the size of the code.
2- In many cases, it increases the speed of the program execution.

Both effects are the goal of every professional programmer. The rest of this chapter discusses how to activate optimization and the unwanted side effects of the optimizer on the code.

If the optimizer is so powerful, why is it off by default? It may seem strange that CCS turns the optimizer off for the projects. As a rule of thumb, never change any default setting in the project before understanding the consequence. This chapter shows why the CCS designers decided to keep the optimizer off for a new project.

Why does the compiler not always optimize C codes? In the example, the variables `'a'` and `'b'` were removed after optimization and no longer exist. This intelligent removal combines multiple C lines to a single assembly instruction (e.g., `'j*5'`). However, since `'a'` and `'b'` no longer exist, the assembly program no longer looks like the original C code during the debugging! For example, if `'a'` is added to the Watch window, the CCS can not find `'a'` and does not display any value for `'a'`.

After enabling optimization, on average, the code execution speed is doubled. Even in some cases (especially for-loops), the speed can improve by 20 times. Therefore, activating optimization in many projects is not a choice but a necessity.

So in practice, the optimizer is off during the first stage of code development. After successfully debugging the code and resolving all the bugs, the optimizer is turned on.

2) Enabling optimization in project 'Properties'

To select different levels of optimization, right-click on the project name and select 'Properties'. In the Properties window that opens, different optimization levels can be selected in the 'Compiler' section under 'Optimization'.

Chapter 14

INTERRUPT

Kyanoosh Shafaei
www.DSPgig.com

1) Introduction

An interrupt results from a series of hardware or software events that eventually lead to a pause in normal program execution and then start a dedicated routine (Interrupt Service Routine). For example, if the processor is like an employee doing daily tasks, an interrupt is like a phone call or a client visit that can interrupt the usual routine. In such circumstances, based on the type of request, the employee goes through a specific procedure (ISR) to quickly respond to the event and eventually continues his or her previous work.

This chapter explains the basics of DSP interrupts and some of the differences between DSPs. Although the interrupt structure is very different between DSP families, this chapter enables the reader to understand a particular processor's datasheet better. In the end, the interrupt is set up for a specific processor.

In older processors, there is usually a fixed location for the interrupt vector. This is because the interrupt vector has the start address for all interrupts. One of the interrupts is reset, and many older processors start executing code from the zero address with a hardware reset. So the initial part of the program memory (for example, the first 128 bytes) is reserved for the interrupt vector. However, in newer processors, the location of the interrupt vector can be changed by the software.

This chapter and the next chapter may be a little harder to read because of so many register names and hardware details. However, it is crucial to see a few actual DSP codes (especially when the hardware is involved) before programming a peripheral.

2) Registers that affect interrupts

Multiple registers are involved in interrupt. The most important of them are:

2.1) INTM[1] flag:

INTM is an internal flag that disables all interrupts. The INTM can mask all interrupts. When the code needs to disable all interrupts, the INTM is set. Although each interrupt has a separated Interrupt Enable (IER), but INTM has a higher priority and can temporarily disable all interrupts.

The INTM has always disabled interrupts when the processor enters an ISR. After the processor receives a new interrupt, as soon as it enters the ISR for that interrupt, the INTM flag is set. So, no other interrupt can interrupt the current interrupt!

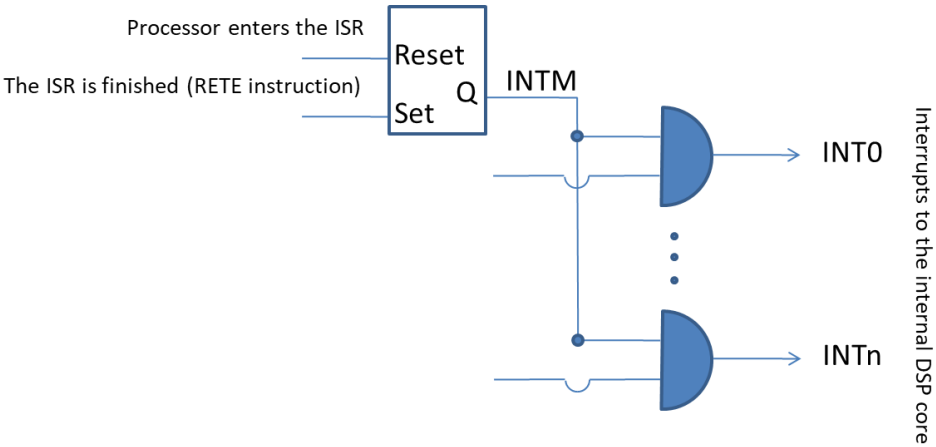


Figure 1: The 'INTM' flag can disable all interrupts

2.2) IER[2] Register:

In the Interrupt Enable Register, there is a separate flag for each interrupt. The flag activates the corresponding interrupt. The name of this register is slightly different between DSP families. For example, in the C55xx series, there are two 16-bit registers called IER0 and IER1 (for a total of 32 interrupts).

[1] - Interrupt Mask
[2] - Interrupt Enable Register

Chapter 15

CSL FOR HARDWARE PROGRAMMING

Kyanoosh Shafaei
www.DSPgig.com

1) Introduction:

Every DSP processor has multiple peripherals such as serial port, Direct Memory Access (DMA), and General-Purpose IO pins (GPIO). There are specific internal registers for each peripheral. The addresses of the registers, as well as their performance, vary between processors in different families. This can be confusing when switching between different processors. For instance, if a code programs DMA in the TMS320C5507, it may not work for the TMS320C5510. Although the DMA registers are very similar in both processors, it may still be necessary to make changes before using the code in another series. So, TI created a library called[1] Chip Support Library (CSL) to program the peripherals by standard C functions.

When using assembly language for programming peripheral registers, the datasheet of the peripheral must be studied carefully, and the values of registers should be extracted from the datasheet. CSL can replace assembly programming for peripheral programming. But in some cases, a CSL function needs the exact value used in assembly programming for the register. As a result, the CSL code may look like assembly instructions.

Because CSL programs the hardware, to understand how to use the CSL functions, the datasheet for each peripheral should be studied. Overall, the expectation is that using C (i.e., CSL) prevents assembly coding. However, due to the nature of hardware programming, the CSL code is similar to assembly. Then, if a designer is familiar with assembly, he or she may choose to use assembly instead of CSL.

---

[1] - The hardware programming libraries have different names. Some companies named them Hardware Abstraction Layer (HAL).

2) Why CSL is the preferred method

There may be tens of registers for some peripherals. In some cases, it may be easier to program these registers directly with assembly[2]. Usually, using CSL does not save time, but it has other advantages:

Advantage 1: The code can be transferred from one DSP to another similar DSP in the shortest possible time.

Advantage 2: When using DSP-BIOS[3], CSL minimizes the possibility of interference between functions. Since there are many graphical user interfaces in DSP-BIOS (and CSL is used at the core of DSP_BIOS), familiarity with CSL helps a lot in learning DSP-BIOS.

Advantage 3: The CSL replaces the assembly coding and improves code readability and maintainability.

Advantage 4: For some peripherals such as USB, there is only one type of configuration, and there is no need to study all the *details* of those peripherals. A sample CSL example can be modified and then be used effectively. For these kinds of peripherals, CSL is the best and fastest practical option.

3) How to use the Chip Support Library (CSL)?

The following steps required to use CSL:

3.1) Step zero: Install CSL package

The CSL is not installed by default in the CCS software. So CSL should be downloaded from TI's website. There is a link to download the CSL installation file on each processor's web page. In many cases, there is a common CSL for multiple DSP families. For example, the CSL installation file is the same for all C55xx processors. The TI website always has the most up-to-date version of CSL. Therefore, first, the latest version of the CSL file should be downloaded and installed.

---

[2] - For most cases only a very few assembly instructions (such as MOV, NOP, RET) are needed for peripheral programming. So, the assembly coding would not be difficult.

[3] - DSP-BIOS is an Operating System (OS) for efficient multitasking coding. It has a graphical interface for TI DSPs and employs JTAG very efficiently to show the processor performance graphically. Despite all its complexity, DSP-BIOS is an effective method to program the processor and some peripherals graphically. The DSP-BIOS is not discussed in this book, but TI has very helpful training videos regarding this subject.

Chapter 16

BOOTLOADING THE PROCESSOR WITHOUT JTAG

Kyanoosh Shafaei
www.DSPgig.com

1) Introduction

There is a fundamental difference between DSP processors and microcontrollers. Because Flash memory has a lower clock speed compared to traditional RAM, microcontrollers can use Flash memory (their clock is slower than DSPs). But using a slow memory such as Flash memory for a high-speed core such as a DSP does not provide any benefits. TI has only used Flash memory for the C2800 family, which has a maximum clock speed of 150 MHz. Even for C2800, the Flash memory speed is still limited. The most important advantage of the internal Flash is the simplicity and the safety of the code programmed into the DSP.

In microcontrollers, the user code is written into the internal Flash. Therefore, even after the power supply is disconnected, the program is still in the Flash. However, most DSPs have SRAM instead of Flash.

The C2800 family is classified more as a microcontroller than a DSP processor.

2) Bootloader

In previous chapters, two methods for loading programs onto DSP memory were provided. Here, these methods are first reviewed:

2.1) Computer + CCS + JTAG

In this method, after compiling the user code, CCS transfers the code to the processor memory using the JTAG debugger. This method can be used for debugging purposes in the laboratory. However, it should not be used by a customer for the final product.

2.2) ROM (bootloader)

There is no Flash in most DSPs. Instead, DSPs have an internal ROM[1]. This ROM can only be programed by the manufacturer[2]. In this ROM, TI stores multiple useful programs. For example, a few math look-up tables such as sine-wave is in the ROM[3]. Another important code in the ROM is 'Bootload'. The bootload code copies the user code from outside DSP into the internal RAM after reset.

The bootloader can use various peripherals. For example, available boot options for C55xx are provided in the next table. In the 55xx series, there are 8 GPIOs (General-Purpose Input / Output) pins. After reset, the ROM code (bootloader) first checks the first four GPIOs (GPIO0 to GPIO3 pins) and detects the bootloading type. The hardware designer always sets these 4 pins based on the available boot option. Multiple boot options can be used when designing hardware.

Table 1: Five out of 11 methods available for loading the program into the internal RAM for C5509

Method description	GPIO	GPIO	GPIO	GPIO
Using USB port to load the user code from a PC.	**0**	**0**	**1**	**0**
Using an I2C Flash and connecting it to the processor I2C pins.	**0**	**1**	**1**	**0**
Using an SPI Flash (with more than 64KB capacity) and connecting it to the MCBSP0.	**0**	**1**	**0**	**0**
Using an SPI Flash (with less than 64KB capacity) and connecting it to the MCBSP0.	**1**	**1**	**0**	**0**
Using the "Standard Serial" method with MCBSP0.	**1**	**0**	**1**	**1**

[1] - Internal ROM is a read-only memory that can only be programmed by the factory at the time of manufacture and cannot be programmed or modified by existing programmers or other methods. The advantage of ROM is that it has faster access speed than Flash memory.
[2] - For large orders, TI accepts user code and can program them into internal ROM.
[3] - If the values of the sine function for 0 to 90 degrees is stored in ROM, all other degrees can be calculated easily.

Chapter 17

SUMMARY

Kyanoosh Shafaei
www.DSPgig.com

1) Introduction

This book is designed for students with little to no background in signal processing and firmware programming. It encourages the reader to use any DSP from the Texas Instruments regardless of its family with multiple steps for using these signal processors. This chapter reviews all the steps briefly again. Therefore, it can be used as a checklist for doing any DSP project to estimate the timeline of the project.

2) The steps to work with any new DSP

The steps to get started with DSP processors are summarized as follows:

1- After running CCS, select a directory as the workspace.

2- Inside CCS, from the 'Project' menu, select 'New CCS Project' to create a new project. The type of DSP family must match the processor type. For example, if the processor's name is TMS320C6713, the '67XX' family type should be selected. Also, the connection method (simulator or JTAG debugger) can be specified.

3- Copy any assembly ('.asm'), C ('.c'), or C ++ ('.cpp' or '.C') file to the project directory. Another method to add files to the project is to link them without copying them to the project folder.

4- If the type of the project is a C/C++ project, CCS adds the RTS file (RTS file contains the standard C functions). The assembly projects do not need an RTS file. In order to convert an assembly project to a C project, add the proper RTS file to the project.

Each processor has multiple RTS files, which are added based on the memory model of the project. The RTS file defines the following:

1- The standard C functions such as 'malloc()' or 'printf()'.

2- The pre-initialization code which is executed before the 'main()' function (i.e. '_c_int00').

5- If CSL functions are used, the appropriate CSLxxxx.lib file must be added to the project (xxxx refers to the family part number).

6- For any other TI library, such as the image library, the appropriate library file ('.lib') should be added to the project. Sometimes, instead of the '.lib', only the assembly source code of the library functions can be added.

Like the RTS library, any other library added to the project should have a matching memory model.

7- Up to this point, C, C ++, and assembly files are added to the project. Now, the project can be compiled and linked (built). One of the generated files is the '.map' file. Map file lists all the memory sections in the project. Create a command file and specify addresses for all the sections.

When making the command file, make sure the memory used by the bootloader is not used for any initialized section.

8- It is essential to review the processor's datasheet to find the address of the available memories. Some memories are faster and can be used for more important sections. To improve system performance, change the source files, and place important functions and variables in special sections.

9- If the execution starts from an address other than '_c_int00', the address must be specified in the 'Specify program entry point' inside the project 'Properties' window.

10- The size of '.stack' and '.sysstack' must be specified for the project. The stack size is critical and can cause unpredictable bugs. The stack size is determined based on the maximum local variables used by the nested called functions. The '.sysstack' size has nothing to do with local

variables but is determined by the maximum number of nested functions, and usually, the default value of 512 bytes is sufficient. Not all DSPs have '.sysstack'.

11- The interrupt vector can be created in assembly, or C. CSL functions are very useful to set up the interrupt.

12- The '.ccxml' file is determined by the hardware connection. If there were any connection issues, check the '.gel' file for each core and make sure the cores have a proper '.gel' file. The '.gel' file is used to program some of the internal registers before loading the code by the JTAG into the memory.

13- Using software such as MATLAB can greatly reduce the debugging time. Plan the project testing carefully to detect any possible bugs. As a rule of thumb, avoid testing the project in real hardware if possible. Use simulators or other methods for the testing.

14- It may be needed to turn on the compiler optimizer for a few files or even the whole project. Review the code carefully to make sure the optimizer does not generate any new bugs. Pay attention to any code performance changes after enabling optimization. This can be a sign of a new bug.

15- After debugging, it is time to store the hex file in the final hardware. The '.out' file is converted to the hex file using the HEX utility (such as 'hex55.exe'). The bootloader document explains the HEX utility and the bootloader process.

3) Final words

This book does not explain the hardware design for the DSP processors. The hardware design is one of the challenging steps in signal processing projects. TI has many useful application notes for hardware design. Also, one of the documents of the DSPLab is for hardware design. If you are a DSPLab user, study the 'DSPLab Hardware manual' to learn the required steps to design a DSP hardware.

TABLE OF FIGURES:

Chapter 1