



TMDSEVMC6472/TMDSEVMC6474L/TMD SEVMC6457L

High Performance DSP Utility Application

Version 1.00.00

User's Guide

October 2010

Document License

This work is licensed under the Creative Commons Attribution-Share Alike 3.0 United States License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/3.0/us/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

Contributors to this document

Copyright (C) 2010 Texas Instruments Incorporated - <http://www.ti.com/>

Getting Help

For technical discussions and issues, please visit

- **C64x Multicore forum:** http://e2e.ti.com/support/dsp/c6000_multi-core_dsps/f/439.aspx
- **BIOS Embedded Software forum:** <http://e2e.ti.com/support/embedded/f/355.aspx>
- **Embedded Processors wiki:** <http://processors.wiki.ti.com>
- **Eclipse RTSC:** <http://www.eclipse.org/rtsc/>

Note: When asking for help in the forum you should tag your posts in the Subject with “MCSDK”, the part number (e.g. “C6472”) and additionally the component (e.g. “NDK”).

Definitions and Acronyms

The following terms are used frequently in this document.

CCS	Texas Instruments Code Composer Studio
CSL	Texas Instruments Chip Support Library
DSP	Digital Signal Processor
HPDSPUA	The name of this Utility: High Performance Digital Signal Processor Utility Application
IP	Internet Protocol
JTAG	Joint Test Action Group
MCSDK	Texas Instruments Multi-Core Software Development Kit
NDK	Texas Instruments Network Development Kit (IP Stack)
PDK	Texas Instruments Programmers Development Kit
RTSC	Eclipse Real-Time Software Components
TCP	Transmission Control Protocol
TI	Texas Instruments
UART	Universal Asynchronous Receiver/Transmitter
UDP	User Datagram Protocol
Unit	Your hardware platform from Texas Instruments; i.e. the TMDSEVM6472, TMDSEVM6474L or TMDSEVM6457L.
Utility	This application: High Performance Digital Signal Processor Utility Application.

Note: We use the abbreviation TMS when referring to a specific TI device (processor) and the abbreviation TMD when referring to a specific platform that the processor is on. For example, TMS320C6457 refers to the C6457 DSP processor and TMDSEVM6457L refers to the actual hardware EVM that the processor is on.

Table of Contents

1. Getting Started	7
1.1. Running the Utility	7
1.2. Architecture.....	11
2. Assigning an IP Address to the Unit.....	14
2.1. Static IP Address.....	14
2.2. DHCP IP Address	14
2.3. User Switch 1	14
3. UART and Console Output.....	17
4. Building the Utility	20
4.1. Building the Project	20
4.2. Build Options for the Utility	26
4.3. Building the JAVA Applet.....	27
4.3.1. Setting up the Build Environment.....	27
4.3.2. Signing the JAR.....	27
4.3.3. Converting the JAR for the Utility.....	28
4.4. Building the Web Pages.....	28
4.4.1. PHP Web Conversion Tool.....	29
4.4.2. Linux Web Conversion Tool	30
4.5. Big Endian	30
5. Configuration File and Memory Map	33
5.1. Startup Tasks.....	33
5.2. Memory Map	34
6. Custom Platform Definitions	35
6.1. Platform Editor.....	35
6.2. Custom Platform for the TMDSEVM6472.....	37
6.3. Custom Platform for the TMDSEVM 6457L	38
6.4. Custom Platform for the TMDSEVM 6474L	39
7. Benchmarks.....	41
7.1. NDK.....	41
Figure 1 - Mini-USB JTAG Connection.....	8
Figure 2 - Finding the HPDSPUA out file.....	9
Figure 3 - Utility Welcome Page	10
Figure 4 - HPDSPUA Architecture.....	12
Figure 5 - TMDSEVM6472 (pre rev 5) User Switch Block.....	15
Figure 6 - TMDSEVM6472 (rev 5 +) User Switch Block	15
Figure 7 - TMDSEVM6474L User Switch Location	16
Figure 8 - TMDSEVM6457L User Switch Location	16
Figure 9 - Shunts for Serial Port	18
Figure 10 - Serial Port Enumeration USB Serial Port	19

Figure 11 - Code Composer Studio Import Menu	21
Figure 12 - Code Composer Studio Import Existing Eclipse Project	22
Figure 13 - Code Composer Studio Browse Menu	23
Figure 14 - CCS Build Properties	25
Figure 15 - Library Build Settings	31
Figure 16 - CCS Build Properties	32
Figure 17 - RTSC Platform Editor	36
Figure 18 - Selecting the Platform Package Repository	37
Figure 19 - Custom Platform Definition for the TMDSEVM6472	38
Figure 20 - Custom Platform Definition for the TMDSEVM 64571	39
Figure 21 - Custom Platform Definition for the TMDSEVM 64741	40
Table 1- Serial Port Settings	17
Table 2 - Build Option for the Utility	27

1. Getting Started

This is the High Performance DSP Utility Application. It demonstrates, through illustrative code and web pages, how you can interface your own DSP application to the various Texas Instruments Software Development Kits.

The software development kits supported with this release are:

- Multi-Core Software Development Kit (MCSDK)
- SYS/BIOS 6
- Network Development Kit (NDK)
- Chip Support Library (CSL)
- Platform Library.

You can learn more about these components by reading the documentation that came with your MCSDK.

1.1. *Running the Utility*

To Load and Run the Utility you should perform the following steps. These steps are also covered in more detail in the MCSDK Getting Started Guide:

http://processors.wiki.ti.com/index.php/BiosMulticoreSDK_1.0_GettingStartedGuide

Step 1

Connect a JTAG to the Unit using the Mini-USB cable that came with your Unit. If you are using Code Composer Studio, you can connect to the JTAG with that.

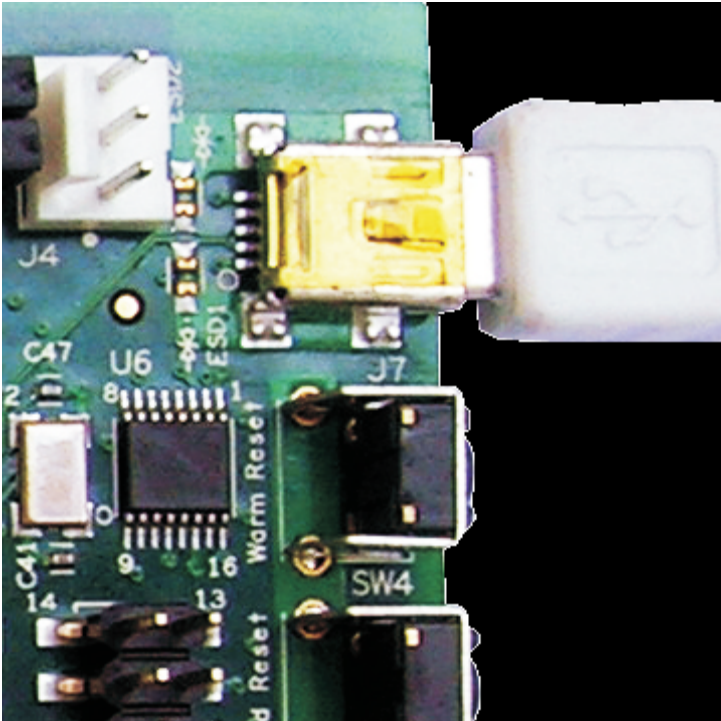


Figure 1 - Mini-USB JTAG Connection

Step 2

Set the Dip Switches on the Unit telling it how to obtain its IP Address. Refer to the Chapter on [Assigning an IP Address to the Unit](#).

Step 3

When the application starts up it writes messages to both the CCS Console (over JTAG) and to the UART to appear in a console window. You can get the Unit's IP address from one of these two locations. See the section [UART and Console Output](#) if you are using a serial cable.

Step 4

Load the pre-built HPDSPUA.out file for the platform you have. You can find the out file in the HPDSPUA project which was installed with the Multi-Core SDK, see Figure 2 - Finding the HPDSPUA out file. If you did the typical install, the SDK will be installed under C:\Program Files\Texas Instruments\mcSDK_1_00_00_xx.

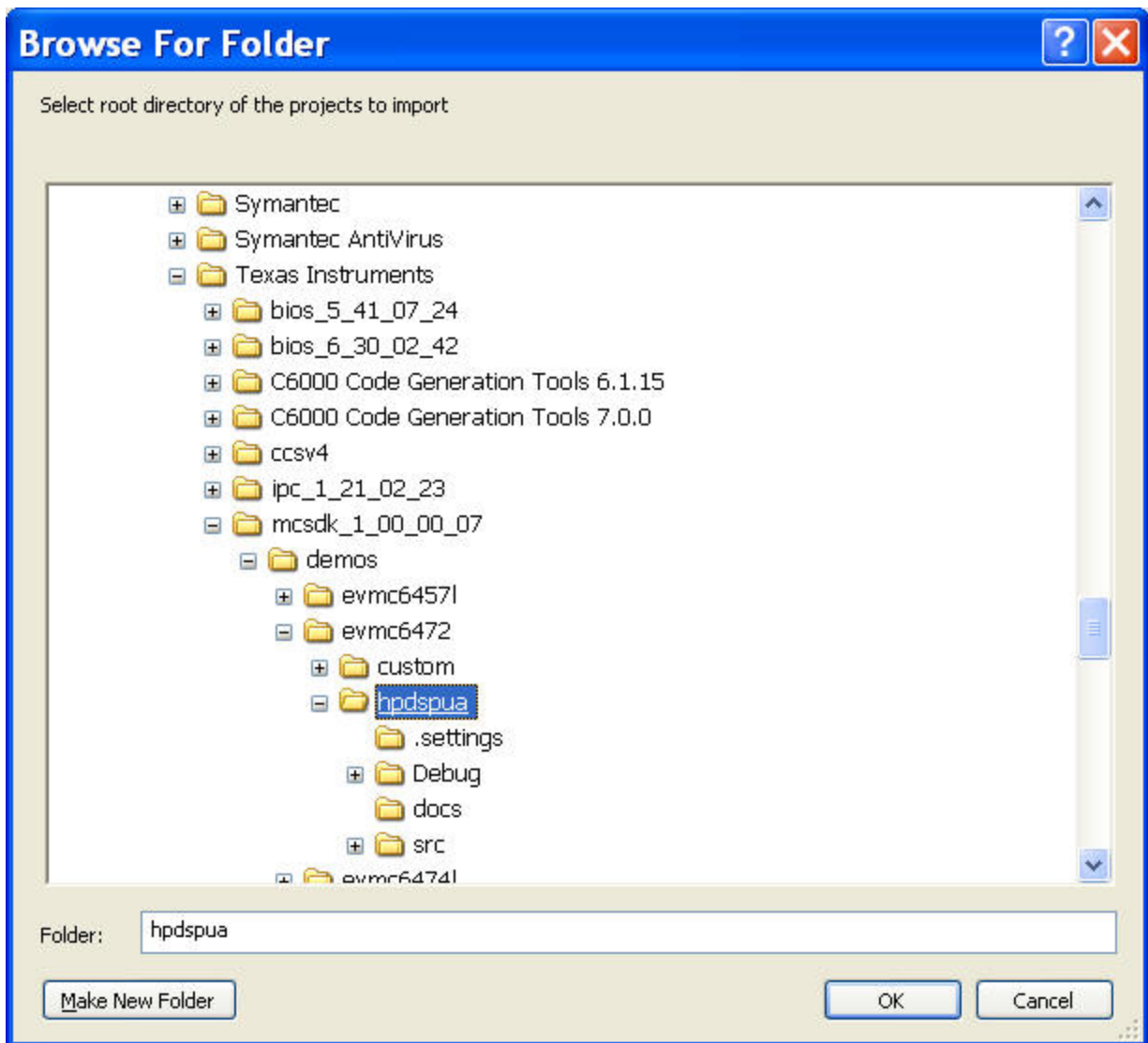


Figure 2 - Finding the HPDSPUA out file

Under the MCSDK directory select the evmcxxxx directory that corresponds to your platform and look inside the HPDSPUA directory. In the Debug and Release directories there will be a corresponding pre-built HPDSPUA.out file. All out files are built for Little Endian.

Step 5

Run the program. Once the Utility starts up and has its IP address you will be able to connect to the web pages as shown in Figure 3 - Utility Welcome Page.

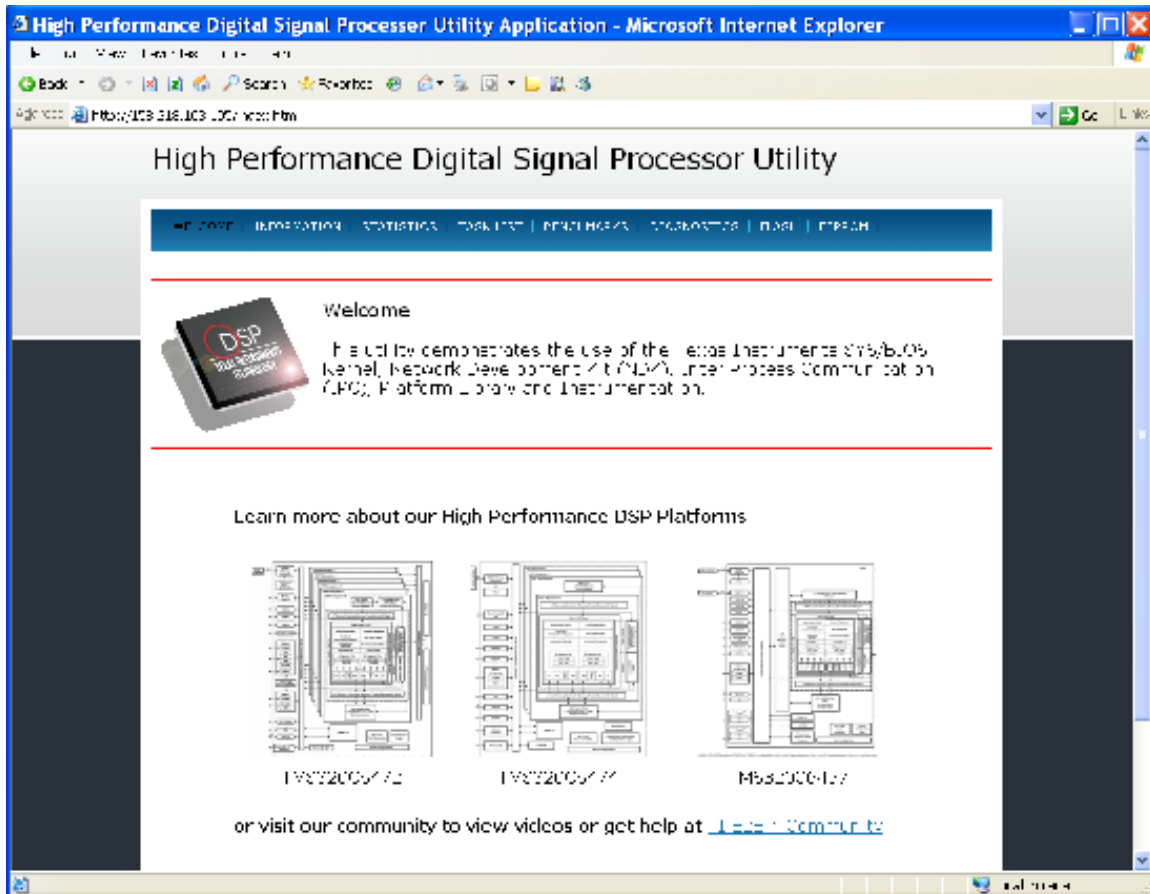


Figure 3 - Utility Welcome Page

1.2. *Architecture*

The architecture for the Utility is shown in Figure 4 - HPDSPUA Architecture.

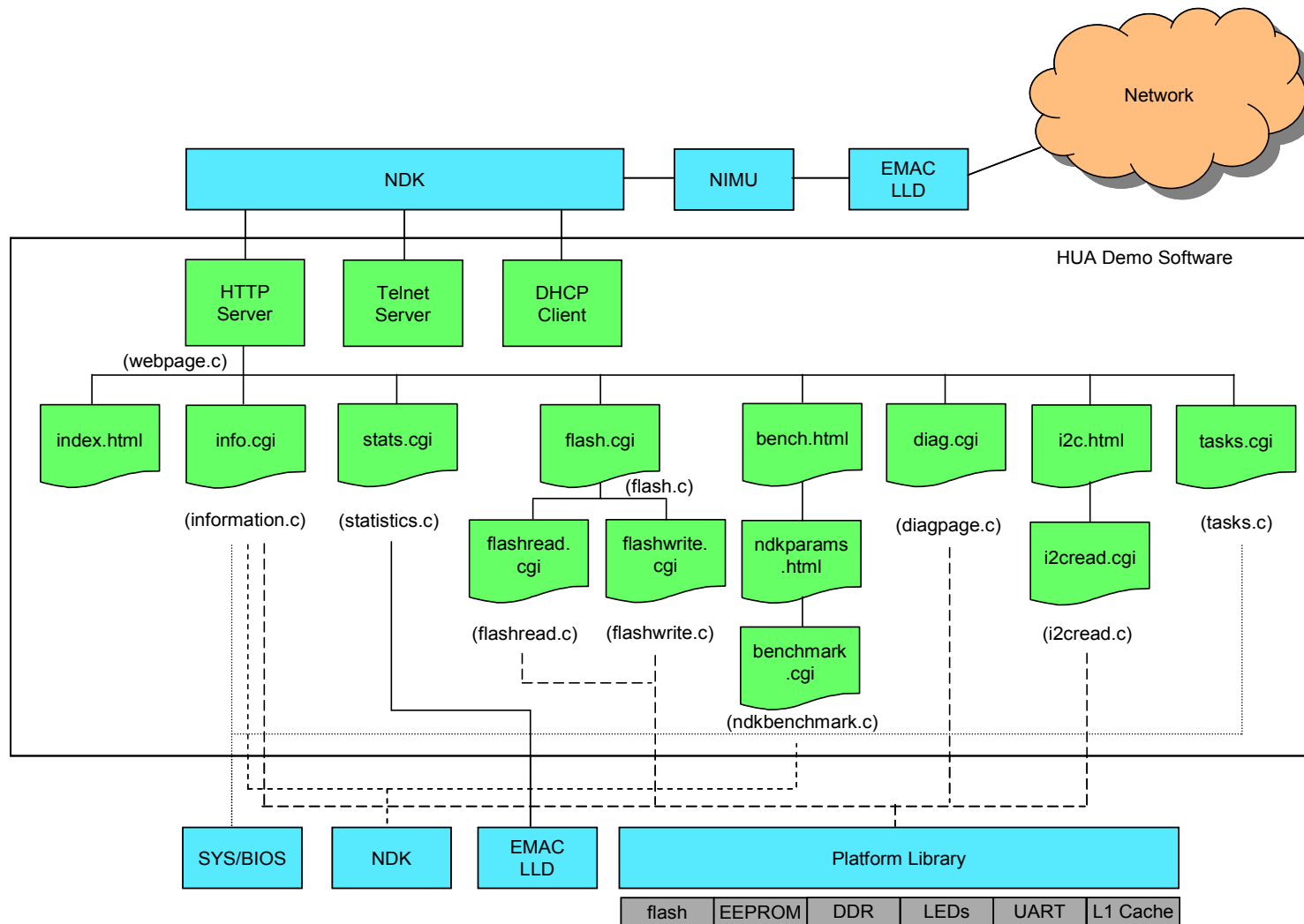


Figure 4 - HPDSPUA Architecture

As can be seen in the diagram, the Utility provides an HTTP and Telnet Server. These servers use standard socket interfaces to the IP stack (NDK) which in turn interfaces to the Ethernet through the NIMU and EMAC Driver components.

The HTTP server serves pages that allow either various operations to be performed on the Unit (e.g. diagnostics) or provide information (e.g. statistics). The web pages are either dynamically created through a CGI-BIN interface (.cgi) or are static pages that are served directly back (.html).

Tasks

As this is an embedded system, it uses SYS/BIOS to provide tasking and OS primitives such as semaphores, timers and so forth. The main thread is the task hpdspuaStart. This task will configure the IP stack and bring the system up into a free running state. Note: The main for the Utility simply start SYS/BIOS. SYS/BIOS in turn will run the task.

Platform Initialization

Platform initialization is performed by a function within the utility called EVM_init(). This function is configured to be called by SYS-BIOS before it starts up. Platform initialization configures DDR, the I2C bus, clocking and all other things platform dependent.

2. Assigning an IP Address to the Unit

To get access to the Utilities Web Pages the Unit must be assigned an IP address. This can be done statically (pre-configured) or through DHCP. When the Utility starts up, it reads User Switch 1. If User Switch 1 is OFF then the application will use a static (pre-configured) IP address. If SW 1 is ON then it will use DHCP. To set User Switch 1 consult the Hardware Manuals for your platform that came with the MCSDK.

2.1. Static IP Address

If configured for a static IP address, the Utility will use: 192.168.2.100. To connect, you will need to be sure your workstation is configured on the same network. You can set it to 192.168.2.1 with a subnet mask of 255.255.255.0.

2.2. DHCP IP Address

If you configure the Utility to use DHCP, then it will get its address from your DHCP Server. To connect, you will need to know what this address is. The easiest way to do this is to [connect to the serial port](#) on the Unit before you start it up. The Utility will write messages to the console including the IP address it was assigned.

2.3. User Switch 1

The location of User Switch 1 is dependent upon your platform.

TMDSEVM6472 EVM (version 4 and below)

User switch 1 is in the User Switch block on the EVM. It's the block with just two switches on it.

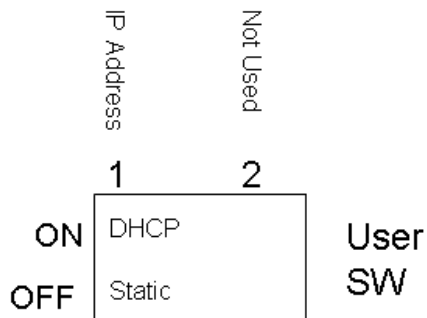


Figure 5 - TMDSEVM6472 (pre rev 5) User Switch Block

TMDSEVM6472 EVM (version 5+)

User switch 1 is in the User Switch block on the EVM. It's the block with just two switches on it.

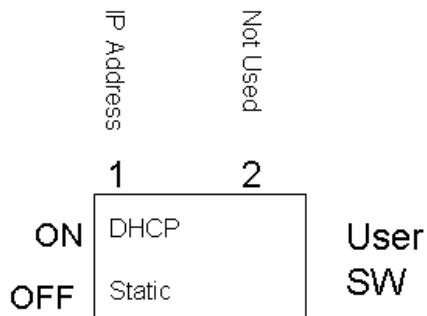


Figure 6 - TMDSEVM6472 (rev 5 +) User Switch Block

TMDSEVM6474L

The user switches are poles 6 and 7 of the Switch 3 block as shown below.

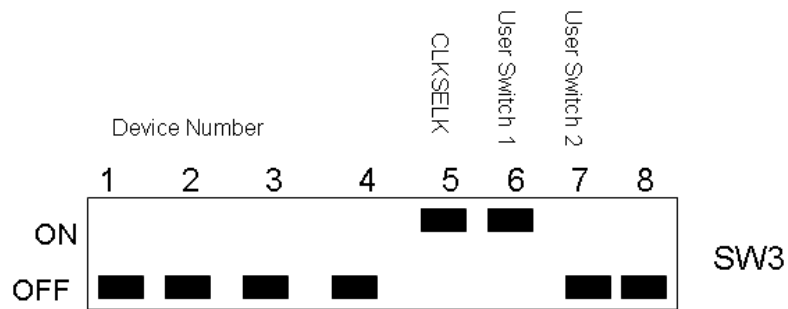


Figure 7 - TMDSEVM6474L User Switch Location

TMDSEVM6575L

The location of the user switches are poles 7 and 8 of Switch Block 4 as shown below.

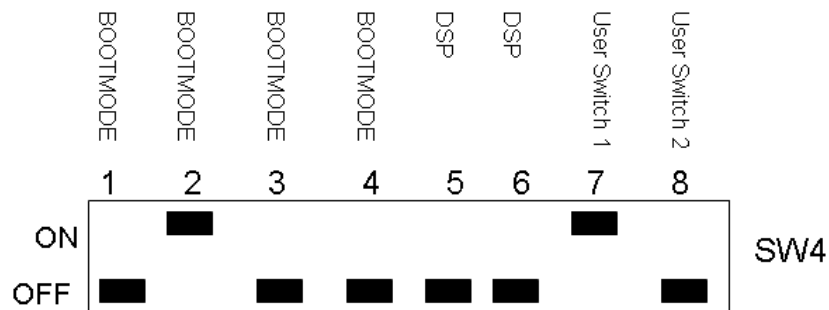


Figure 8 - TMDSEVM6457L User Switch Location

3. UART and Console Output

The Utility will display its messages over the UART when it starts up. To connect to the UART and see the messages you will need a serial port cable for the Unit and a serial port program (i.e. HyperTerminal, putty, etc) to act as the console.

The settings for the serial port should be:

Baud Rate	115200
Data Bits	8
Stop Bits	1
Parity	None
Hardware Flow Control	None

Table 1- Serial Port Settings

It is preferable to use the COM serial port vs. the USB serial port on the Units. To use the COM serial port you will need to attach the 3-pin serial port connector shipped with your platform to the connector on the EVM and the other end to the COM port on your workstation. You will also need to change the “shunt” settings on the Unit.

As you can see in Figure 9 - Shunts for Serial Port, the shunts are located right next to the 3 pin connector; you should move both shunts to the right, as shown, to configure the Unit to use the PC COM port setting it is plugged into (and NOT the USB serial port).



Figure 9 - Shunts for Serial Port

Note: Version prior to 5 of the TMDSRVM6472 only shipped with a COM serial port and has a DB-9 connector (vs. the 3-pin serial).

USB Serial Port

NOTE: When using the USB UART watch power cycling the platform. This works when using the serial port interface (i.e. COM1), but does not when using the USB interface. This is because the Utility starts trying to write to the UART before Windows has assigned the virtual COM port. However, if you do a Cold or Warm reset via the buttons on the Unit, you should see the correct output to the terminal.

For those platforms supporting a USB serial port it can be connected the mini USB cable and connector that came with the platform. As this is a USB serial port you will not know its exact COM port name until it enumerates. If you are in Microsoft Windows, HyperTerminal will show all COM port enumerations and generally the Unit is the latest one in the list (e.g. it might show COM1, COM2 and COM49). The right one in that list will be COM49. To verify the serial port in use, you can use the Device Manger if you are running within Microsoft Windows. In Device Manager, view the Ports and then the properties of each. You will be looking for the Port that

has its Location set something like “on TI XDS100 Channel B”. See Figure 10 - Serial Port Enumeration USB Serial Port .

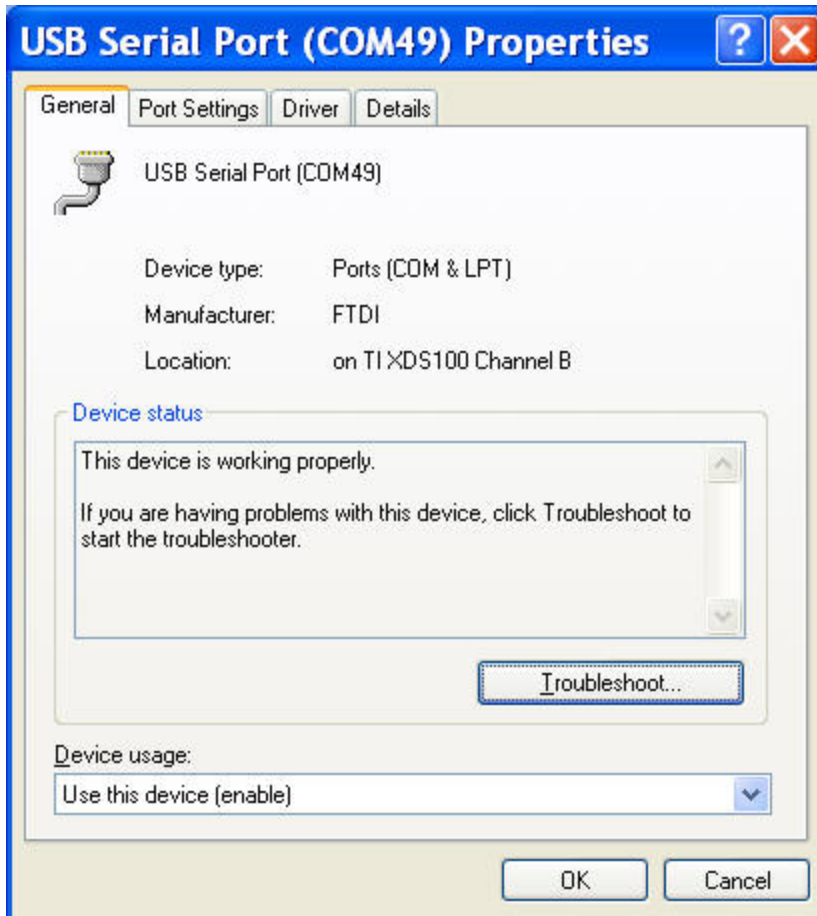


Figure 10 - Serial Port Enumeration USB Serial Port

4. Building the Utility

The following instructions will help with setting up the environment to import and build the Utility. It should be noted that there are two pieces that require building or prep work outside of the CCS project for HPDSPUA if you want to modify them. These two pieces are the Java Applet and the static (e.g. i2c.html) web pages. If you wish to modify these, then you will need to make your changes and prep them before you re-build this Utility. See the respective sections in this Chapter on how to do that.

This Utility is a SYS/BIOS 6 and RTSC Project. The RTSC portion consists of the “evm.cfg” file which allows us to bind more easily to SYS/BIOS and the NDK. It also allows us to automatically resolve include paths and libraries for these components based on the version selected to build with.

4.1. *Building the Project*

In order to build the Utility, you must do the following:

1. Import the Project into CCS
2. Verify the Build Settings.
3. Compilation

Each of these steps is discussed further.

Step 1 Importing the Project

Use CCSv4 "Project->Import Existing CCS/CCE Eclipse Project" to add the project to your workspace. The project should build whether you choose to create a copy in the workspace or not. Creating a copy within the workspace is a safe way to start becoming familiar with the program.

1a Select Import

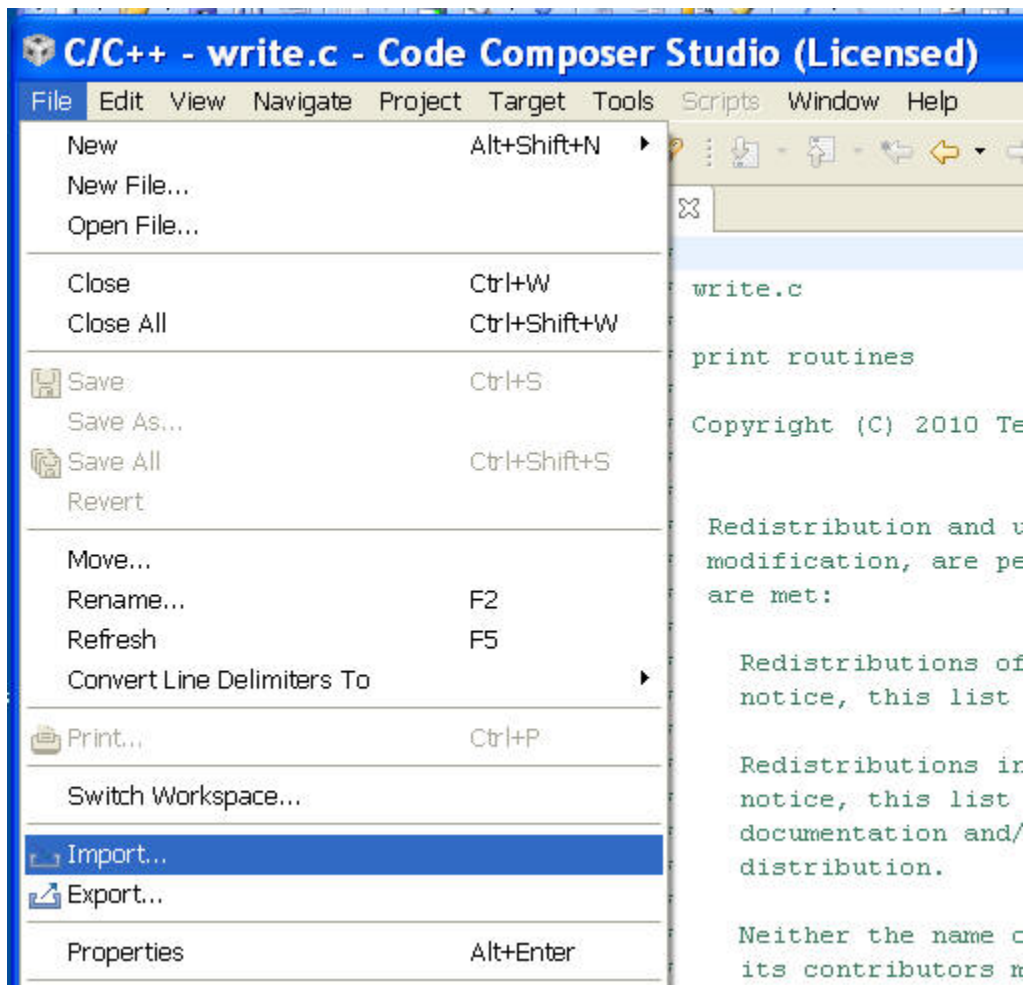


Figure 11 - Code Composer Studio Import Menu

1b Select Existing CCS Projects

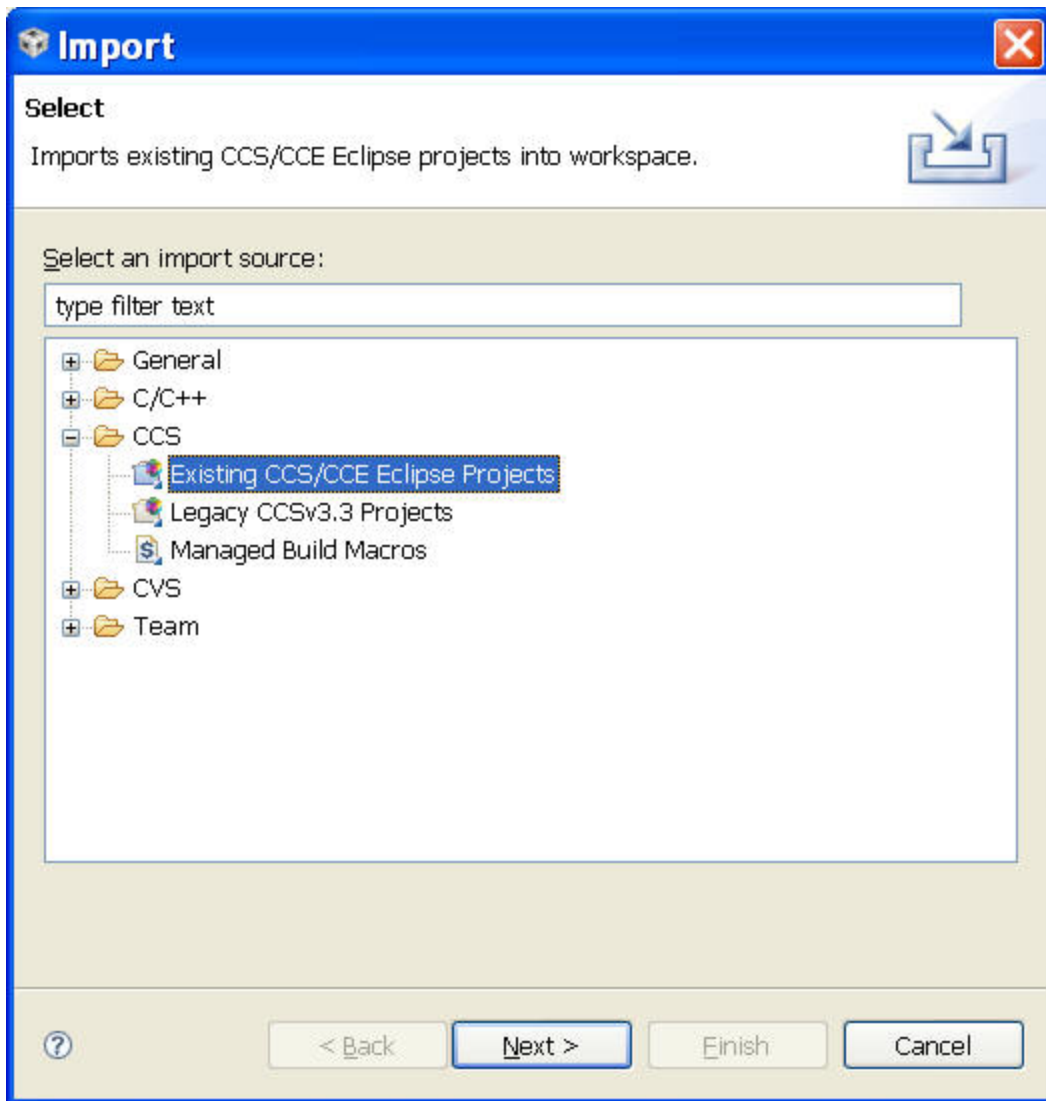


Figure 12 - Code Composer Studio Import Existing Eclipse Project

1c Browse to the MCSDK Folder

Browse to the folder where you installed the MCSDK. If you did a typical install this will be the C:\Program Files\Texas Instruments folder. Under the MCSDK directory go into demos and then select the HPDSPUA that corresponds to your platform:

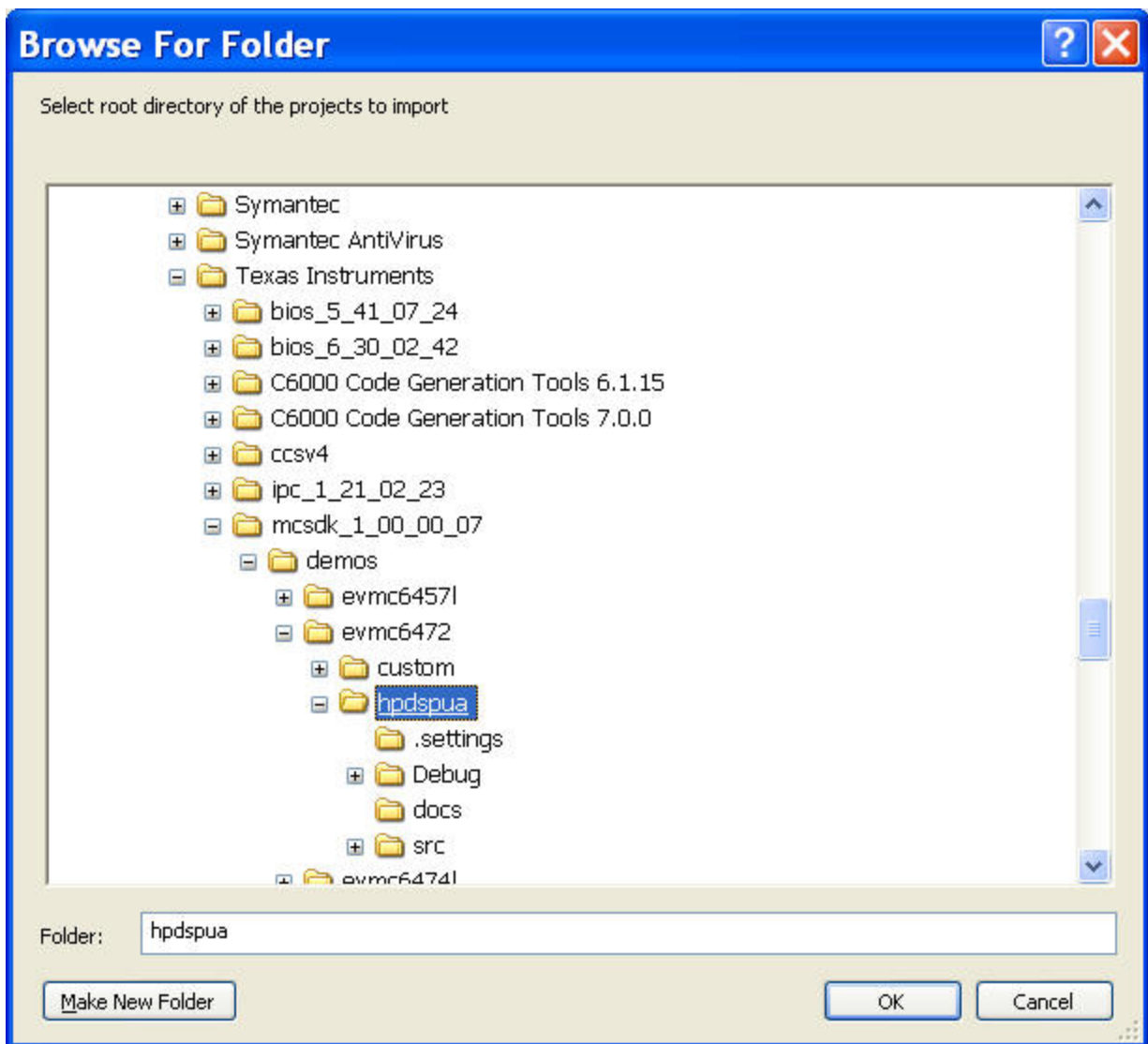


Figure 13 - Code Composer Studio Browse Menu

1d Done

Click Finish. The project should be imported.

Step 2 Verifying the Build Settings

When the project is imported it should resolve all dependencies automatically (assuming you have installed the correct versions of the other

components) so this is an optional step. If you experience problems building the application then you may wish to consult this section.

To verify the settings perform the following steps.

2a Check that the custom platform is set

Check that the custom platform is set. To do this, right click on the project and go to its Build Properties. Select CCS Build. Click on the RTSC Tab. You should see something similar to the window shown in Figure 14 - CCS Build Properties.

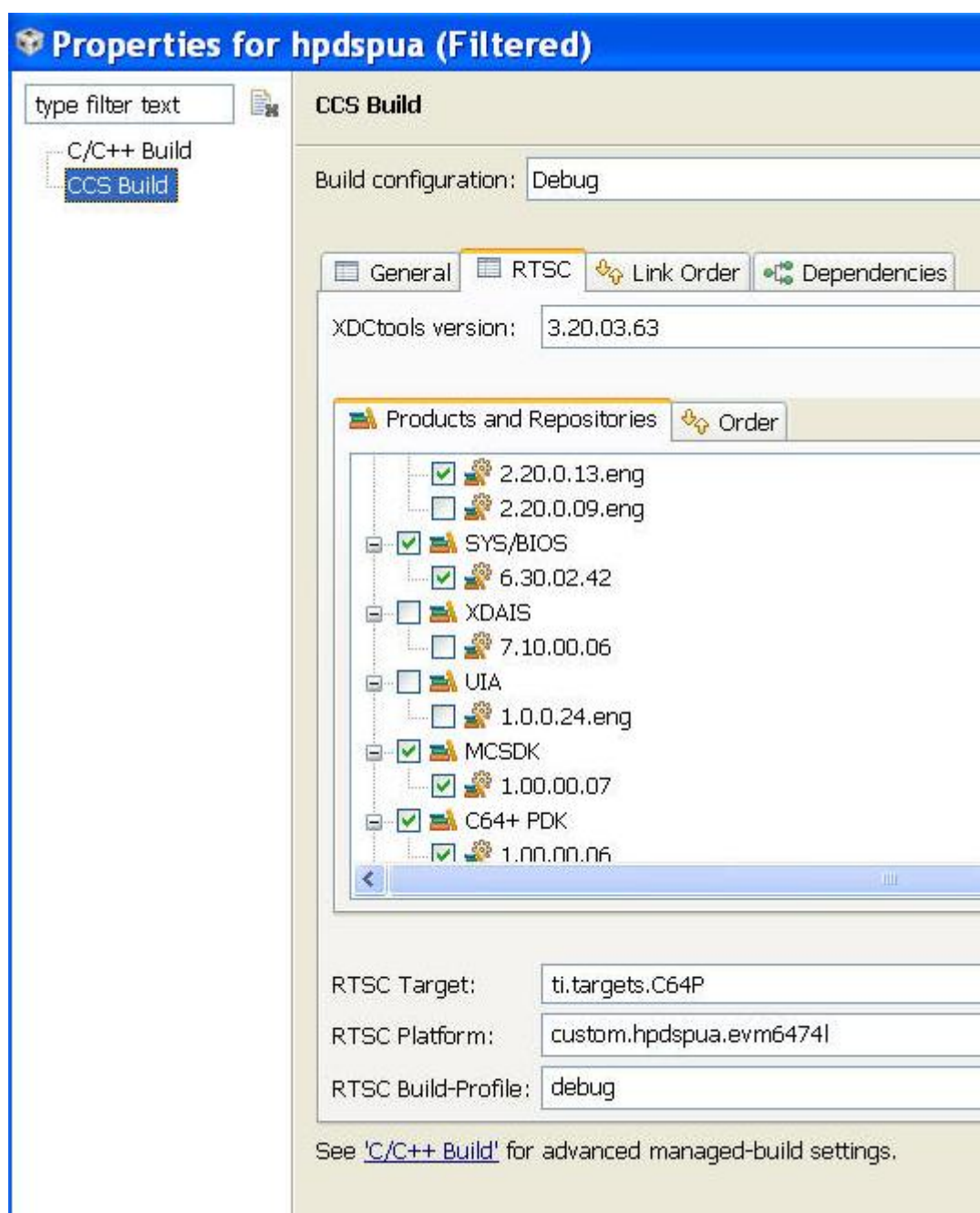


Figure 14 - CCS Build Properties

The RTSC platform should be set the platform for which you are using. Possible choices are evm6472, evm6457l or evm6474l. Make sure the version of the MCSDK you are building with is checked in the boxes. This should

resolve the Macro used to find the repository which has the custom platform definition.

2b Make sure the correct component and versions are checked

Make sure the correct versions of SYS/BIOS, PDK, MCSDK and NDK are checked (See the Release Notes). If the right versions are not selected then some of the Macro (environment) names used for include and library paths will not properly resolve.

Step 3 Compilation

You are now ready to build the project. To do so, right click on the HPDSPUA project and select Build. After compilation the executable, HPDSPUA.out, will be created in the Debug or Release directory depending on which version you chose to build.

4.2. Build Options for the Utility

The Utility uses the following build options.

Option	Comments
<code>_INCLUDE_NIMU_CODE</code>	This is a pre-defined symbol within the Build properties. It tells NDK to use the NIMU interface. It generally has no effect on this Utility unless you bring in NDK source files that use that define.
<code>_NDK_EXTERN_CONFIG</code>	This is a pre-defined symbol within the Build properties. It generally has no effect on this Utility unless you bring in NDK source files that use that define.
<code>C6472</code> or <code>C6457</code> or <code>C6474</code>	This is a pre-defined symbol within the Build properties. It is used by the Ethernet driver to conditionally compile support for various platforms. It generally has no effect on this Utility

	unless you bring in Ethernet driver source files that use that define.
INCLUDE_TELNET_SERVICE	This define is in the HPDSPUA.c file. You can turn it on to compile the Telnet Server. By default, it is compiled out.

Table 2 - Build Option for the Utility

4.3. Building the JAVA Applet

The Java Applet used for the NDK Benchmark test must be built and packaged into a JAR using a Java Build Environment. You can download one from Eclipse for free and we use the ones from there as well. You can create a Java project and then add a new package called “ndkbenchmark” and copy the java source file to the folder and make any changes to file and then export it as a Java JAR file NDKBenchmark.jar.

Note: Java Applets run with restricted access by default, which means every rebuild of the same applet requires the developer to re-“sign” the applet, since our applet requires socket permissions.

4.3.1. Setting up the Build Environment

Download Eclipse from <http://www.eclipse.org/downloads/> and install it.

Warning: Eclipse Workspaces and CCS 4.2 workspaces are currently not compatible with each other. Use separate workspaces for each.

To export the project into an Applet, click this button **[graphic TBD]** in the toolbar area in Eclipse select NDKBenchmark, click finish. A window should pop up and you will see the jar file which is your applet.

4.3.2. Signing the JAR

The NDKBenchmark applet needs to be signed, because it needs permission to do socket calls. To sign the Applet, do the following steps.

1. Make sure you have Java SDK installed; this is not the JRE (Java Runtime Environment).
2. Open Command Prompt or Terminal, and “cd” into the directory of the jar file. Type in the following commands:
 - a. `keytool -genkey -alias <name of your public key>`
 - b. `keytool -selfcert -alias <name of your public key>`
 - c. `jarsigner <jar file> <name of your public key>`
3. The keytool calls will create certified keys, while the jarsigner command will place the certificate in the META-INF directory in your jar file.

4.3.3. Converting the JAR for the Utility

You must convert the JAR file into a format that the Utility can use. This is done using the same utilities you use for converting a web page. To convert the JAR use the Web Conversion tools (PHP or Linux) as discussed in 4.4.

4.4. Building the Web Pages

The Web pages used by this Utility can be created in two ways: Dynamically or Statically.

Dynamic web pages are created by source code as needed. Web pages that are dynamic end with a “.cgi” extension. An example of a dynamic web page in this Utility is `flash.cgi` which implements the Flash Utility Web Page. The source file that implements that web page is `flash.c`.

Static web pages are created by using HTML files (and more than likely an HTML editor). An example of a static web page in this Utility is `index.html` which implements the Welcome Page. Static web pages can not be used as is within the Utility (as we don’t yet have a flash file system). As such, they need to be converted into a format that is useable. This format is a hexadecimal representation of the page.

Following our example above, the `index.html` page is converted into a hexadecimal representation and stored in an array in .h file that is created.

File indexhtml.h:

```
#define INDEXHTML_SIZE 4859
unsigned char INDEXHTML[] = {
0x3c, 0x21, 0x44, 0x4f, 0x43, 0x54, 0x59, 0x50,
0x45, 0x20, 0x48, 0x54,
0x4d, 0x4c, 0x20, 0x50, 0x55, 0x42, 0x4c, 0x49,
0x43, 0x20, 0x22, 0x2d,
0x2f, 0x2f, 0x57, 0x33, 0x43, 0x2f, 0x2f, 0x44,
0x54, 0x44, 0x20, 0x48,
0x54, 0x4d, 0x4c, 0x20, 0x34, 0x2e, 0x30, 0x31,
0x20, 0x54, 0x72, 0x61,
0x6e, 0x73, 0x69, 0x74, 0x69, 0x6f, 0x6e, 0x61,
0x6c, 0x2f, 0x2f, 0x45,
0x4e, 0x22, 0x20, 0x22, 0x68, 0x74, 0x74, 0x70,
0x3a, 0x2f, 0x2f, 0x77,
0x77, 0x77, 0x2e, 0x77, 0x33, 0x2e, 0x6f, 0x72,
0x67, 0x2f, 0x54, 0x52,
0x2f, 0x68, 0x74, 0x6d, 0x6c, 0x34, 0x2f, 0x6c,
0x6f, 0x6f, 0x73, 0x65,
...
}
```

This array is then placed into an Embedded File System and associated with any access to the page index.html. Two tools are provided for converting a HTML file into this hexadecimal representation. See the source file webpage.c for implementation details.

4.4.1. PHP Web Conversion Tool

There is a PHP script to convert web content (pages, images, etc) into something useable by this Utility. The script is located in the webpages/tool/php directory. The name of the script is hexwriter.php. To run the script you will need to have installed PHP on your workstation and have the PHP command line tool be accessible.

Usage: php hexwriter.php *file*

File is the name of the content to convert. The output is a header file (.h). The name of this file will be the filename+extension.h.

As an example, let's say we want to convert index.html. To do so using this page we would run `php hexwriter.php index.html`. The output from that would be the file `indexhtml.h`.

4.4.2. Linux Web Conversion Tool

There is a Linux script and java class to convert web content (pages, images, etc) into something useable by this Utility. The script is located in the `webpages/tool/linux` directory. The name of the script is `comp.sh` and the java class it uses is called `hexwriter.class`.

Usage: `comp.sh file`

File is the name of the content to convert. The output is a header file (.h). The name of this file will be the filename+extension.h.

As an example, let's say we want to convert index.html. To do so using this page we would run `comp.sh index.html`. The output from that would be the file `indexhtml.h`.

4.5. Big Endian

This Utility provides binaries in the Little Endian format. To use Big Endian you will need to re-build the application and link it with the Big Endian versions of the libraries it uses.

To build for Big Endian you should do the following:

1. Change the libraries the Utility links with to use their Big Endian counterparts. See Figure 15 - Library Build Settings. You do not need to change anything for NDK or SYS/BIOS.

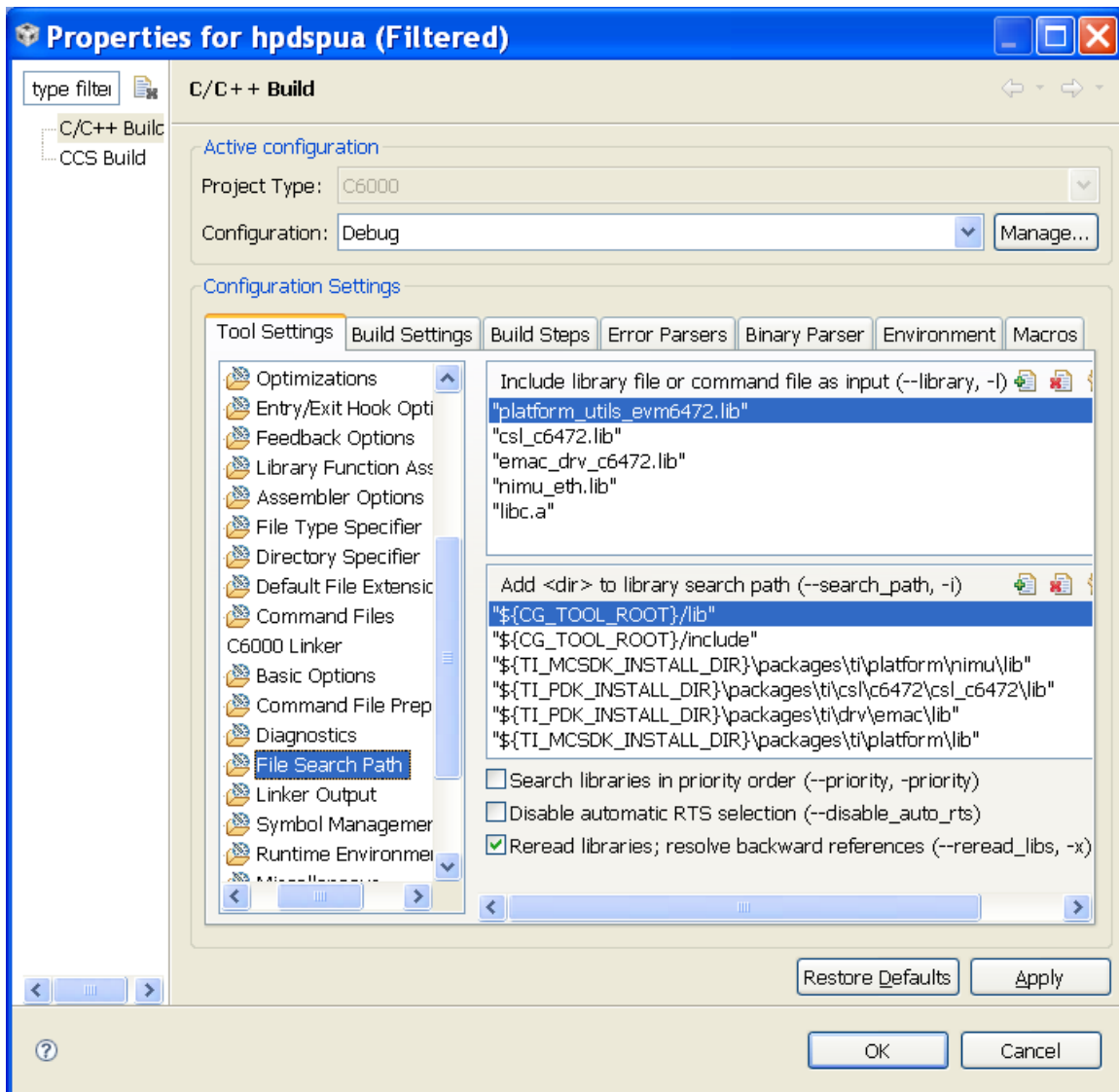


Figure 15 - Library Build Settings

2. Change the build setting for the project from Little Endian to Big Endian and re-build. To change the setting, highlight the project, right click and select Build Properties. On the Build Properties screen, select CCS Build. See Figure 16 - CCS Build Properties.

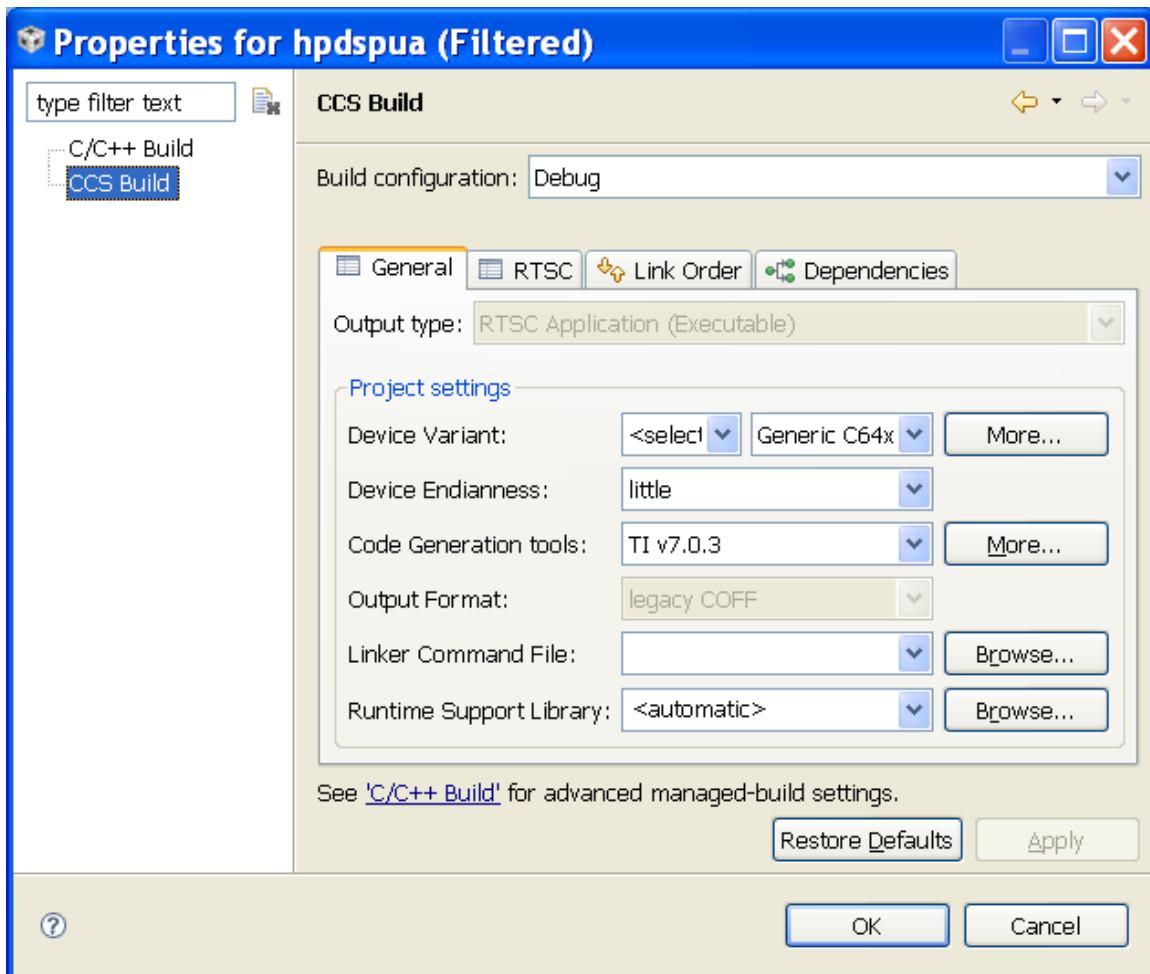


Figure 16 - CCS Build Properties

5. Configuration File and Memory Map

This Utility uses a configuration file called `evm.cfg`. This configuration file does several important things. They are:

1. It creates the linker command file for the memory sections.
2. It defines some of the initial start-up tasks for the Utility.
3. It configures SYS/BIOS and NDK.

The file is fairly well commented and explains the configuration options.

5.1. *Startup Tasks*

EVM_init

`EVM_init` is the first function called by BIOS when the Utility starts up. It executes before `Main` is called and provides the platform initialization. If you are debugging with CCS, then this function will execute as CCS loads it if the option in your Target Configuration file (`.ccxml`) has the option set to execute all code before `Main` (which is the default). The `EVM_init` code from the configuration file is:

```
Startup.firstFxn$. $add('&EVM_init');
```

You can find this function implemented in the source file `hpdspua.c`.

HPDSPUASStart

The function `hpdspuaStart` is the Main Thread (Task) for the Utility. It configures the IP stack and creates the necessary services like HTTP Server, DHCP Client and so forth. The code fragment from the configuration file for the Thread is:

```
var tskNdkMainThread = Task.create("&HPDSPUASStart");  
tskNdkMainThread.stackSize = 0x2000;
```

```
tskNdkMainThread.priority = 0x5;  
tskNdkMainThread.instance.name = "HPDSPUAStart";
```

You can find this function implemented in the source file HPDSPUA.c.

5.2. Memory Map

The memory map takes the section names used by the Utility and pieces of the Multi-Core SDK and maps them to three different memory sections. These memory sections are LL2RAM, SL2RAM and DDR2.

While these names are made up they do have specific meaning as follows:

- DDR2 - DDR2 always goes into the external RAM on the platform.
- LL2RAM – LL2 goes into the Local L2 of the core (cache).
- SL2RAM - This is data that should go into the shared L2 (if it exists on the platform) or could be placed into LL2 if it doesn't.

In turn these section names are mapped to specific memory areas on the Platform by the [custom Platform Definition file the Utility](#) uses.

Some examples from the configuration file using these section names are:

```
/* NDK Structures */  
Program.sectMap[".far:NDK_OBJMEM"] = {loadSegment: "LL2RAM", loadAlign:  
8};  
  
/* Upload buffer used by the Web Server*/  
Program.sectMap[".gBuffer"] = {loadSegment: "DDR2", loadAlign:8};  
  
/* Web Pages and web server structures */  
Program.sectMap[".far:WEBDATA"] = {loadSegment: "DDR2", loadAlign: 8};  
  
/* EMAC Buffer Pool */  
Program.sectMap[".emacComm"] = {loadSegment: "LL2RAM", loadAlign:128};  
  
/* NDK Buffer Pool */  
Program.sectMap[".far:NDK_PACKETMEM"] = {loadSegment: "LL2RAM", loadAlign:  
128};
```

6. Custom Platform Definitions

This Utility uses a set of custom platform definitions that define the Clock Speed and the start and size of the different memory areas (External RAM, Internal RAM, Cache and so forth) for the actual hardware.

The definitions used in these platform files match the memory names used in the Configuration file, `evm.cfg`. For example, the section name `DDR2` will have a `DDR2` in the platform file that maps to the specific hardware region where external RAM is. There is one custom platform definition for each hardware platform as follows:

- `Custom.HPDSPUA.evmc6472` – The TMDSEVM6472 Platform
- `Custom.hpdpsua.evmc6474l` – The TMDSEVM6474L Platform
- `Custom.hpdpsua.evmc6457l` – The TMDSEVM6457L Platform

Note: The boot loader on the platforms uses the last 64K L2 for the code and then two sections of external DDR at the high end (around 9Mb) for heap space and to hold the boot image. The memory maps do not protect these areas so you should be aware that if you extend anything into these areas there may be some conflicts at boot time.

6.1. Platform Editor

The Platform definitions were created using the RTSC Platform Tool plug-in for CCS; see Figure 17 - RTSC Platform Editor.

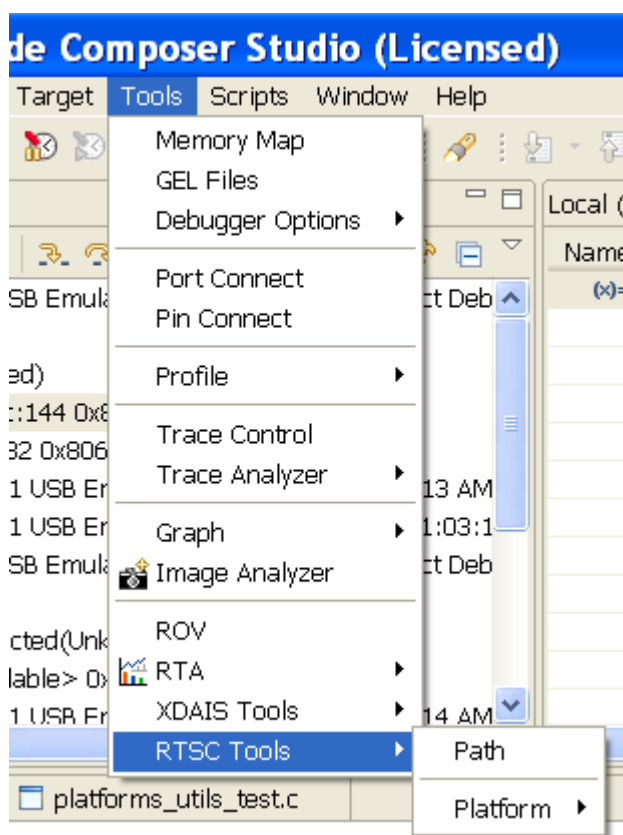


Figure 17 - RTSC Platform Editor

Note: The Platform Editor is provided by the XDC Tools Package. If that package is not installed or is not properly configured, CCS will not display the RTSC Tools option in the Tools Menu.

When you start the editor up, you will need to point the Repository path to the directory where the platform definitions files are located. This will be under the MCSDK demo directory as shown in Figure 18 - Selecting the Platform Package Repository. Make sure you select the directory at the platform level (i.e. evm64571) for the platform you want the map for. If you select the custom directory then the definition files will not be found.

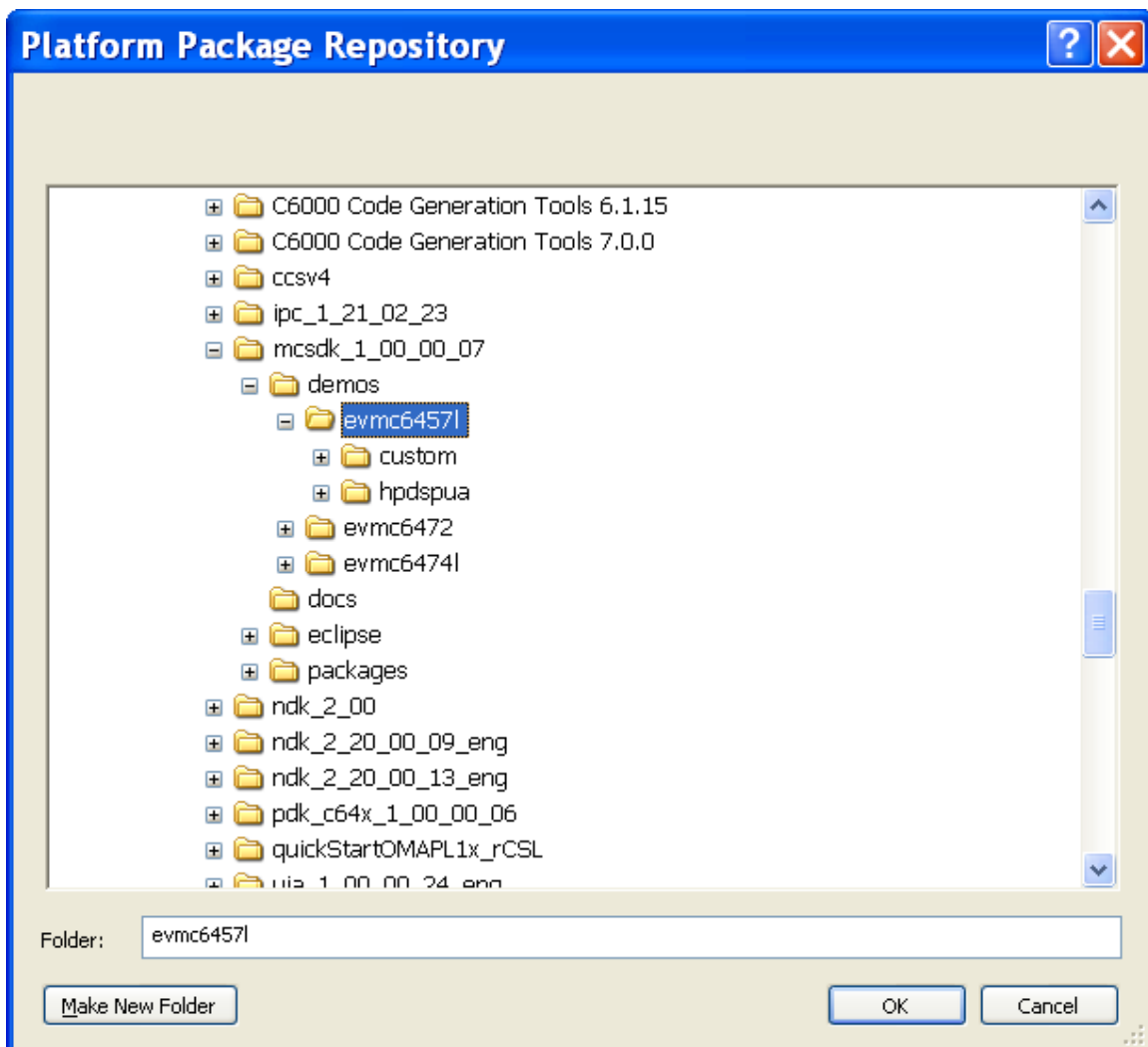




Figure 18 - Selecting the Platform Package Repository

6.2. Custom Platform for the TMDSEVM6472

The Custom Platform name for the TMDSEVM6472 is “custom.HPDSPUA.evm6472”.


Edit Platform


Page 2 of 2 - Device Page
Enter Details for device

Device Details

Device Name:
Device Family:
Clock Speed (MHz):

Device Memory

Name	Base	Length	Space	Access
L1PSRAM	0xE00000	0x00000000	code	RWX
L1DSRAM	0xF00000	0x00000000	data	RW
LL2RAM	0x00800000	0x00058000	code/data	RWX
SL2RAM	0x00200000	0x000c0000	code/data	RWX

L1D Cache: L1P Cache: L2 Cache:
☐ Customize Memory

External Memory

Name	Base	Length	Space	Access
DDR2	0xe0000000	0x10000000	code/data	RWX

Memory Sections

Code Memory: Data Memory: Stack Memory:




Figure 19 - Custom Platform Definition for the TMDSEVM6472

6.3. Custom Platform for the TMDSEVM 6457L

The Custom Platform name for the TMDSEVM 6457L is “custom.HPDSPUA.evm6457l”.

Edit Platform
✕

Page 2 of 2 - Device Page

Enter Details for device

Device Details

Device Name:

Device Family:

Clock Speed (MHz): Import...

Custom Memory

Name	Base	Length	Space	Access
LL2RAM	0x00800000	0x00100000	code/data	RWX
L1PSRAM	0x00e00000	0x00008000	code	RWX
L1DSRAM	0x00f00000	0x00008000	data	RW
SL2RAM	0x00900000	0x00100000	code/data	RWX
DDR2	0xe0000000	0x10000000	code/data	RWX

L1D Cache: ▼

L1P Cache: ▼

L2 Cache: ▼

Memory Sections

Code Memory: ▼

Data Memory: ▼



Stack Memory: ▼

?
< Back
Next >
Finish
Cancel

Figure 20 - Custom Platform Definition for the TMDSEVM 6457I

6.4. Custom Platform for the TMDSEVM 6474L

The Custom Platform name for the TMDSEVM 6474L is “custom.HPDSPUA.evm6474I”.


Edit Platform


Page 2 of 2 - Device Page
 Enter Details for device

Device Details

Device Name:

Device Family:

Clock Speed (MHz):

Custom Memory

Name	Base	Length	Space	Access
LL2RAM	0x00800000	0x00100000	code/data	RWX
L1PSRAM	0x00e00000	0x00008000	code	RWX
L1DSRAM	0x00f00000	0x00008000	data	RW
DDR2	0x80000000	0x10000000	code/data	RWX

L1D Cache:
 L1P Cache:
 L2 Cache:

Memory Sections

Code Memory:
 Data Memory:
 Stack Memory:


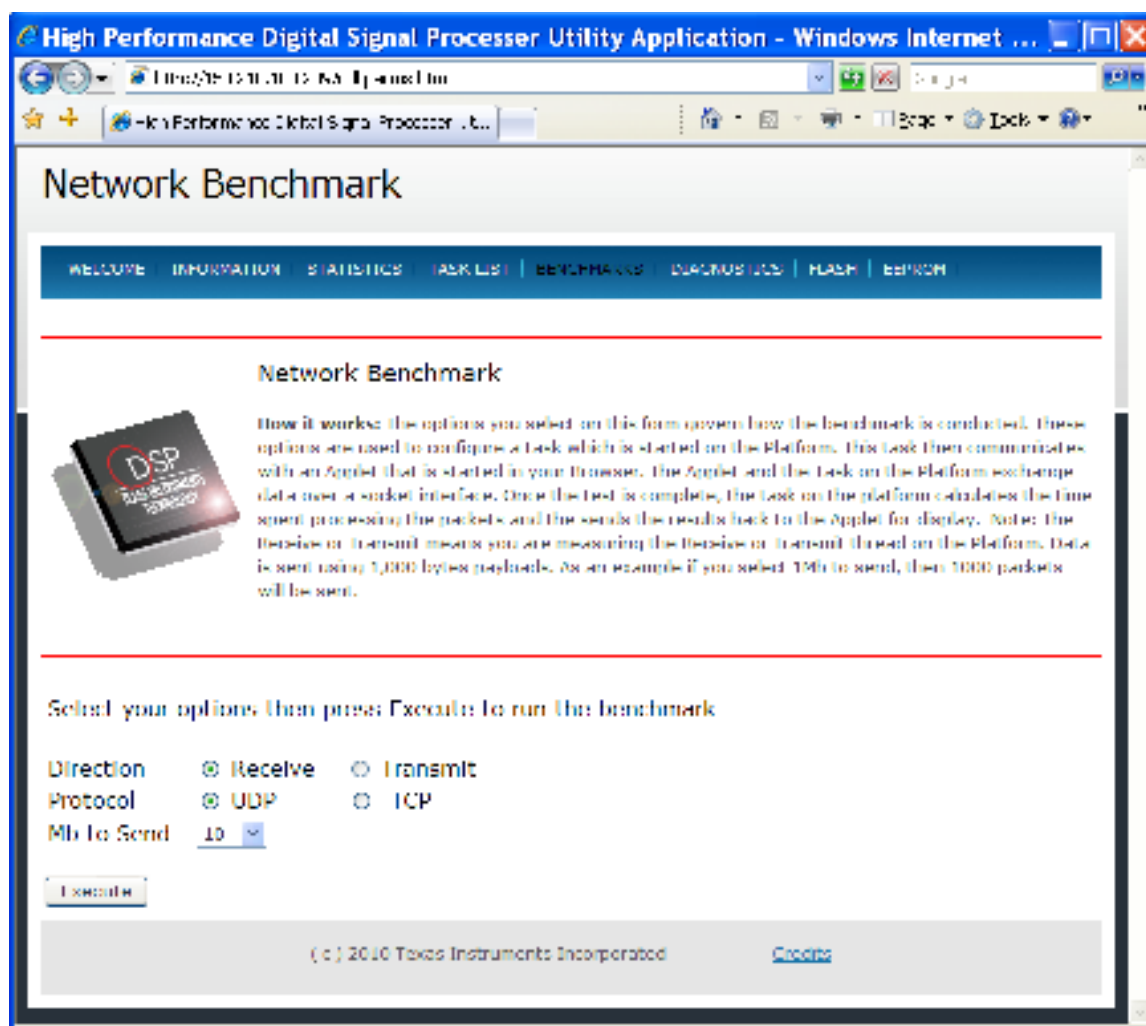


Figure 21 - Custom Platform Definition for the TMDSEVM 6474I

7. Benchmarks

7.1. NDK

Benchmark tests use Java Applets so your Browser must allow them to run. You may get a Security Warning concerning the Applet. If so, you must allow it to run to perform the benchmark.



The options allow you to do UDP/TCP send or receive throughput test between the platform and the Applet. Upon completion the results are displayed on the console and the browser. Note: Data is sent using 1,000 bytes payloads. As an example if you select 10 Mb to send, then 10000 packets will be sent. The test is for demonstration purpose without any optimizations.