Lightning (DSPC-8681E) User Guide

Revision v0.7.6

Initiated by	Sungyi Chen Holland Huang	**Job Title**	Senior Engineer Senior Engineer	**Signature**	
Approved by	Dick Lin	**Job Title**	Software Manager	**Signature**	
2nd Approved		**Job Title**		**Signature**	
Release Status				**Release Date**	

Revision History

Version	Date	Author	Description
0.1	08/08/11	Sungyi Chen Holland Huang	Initial draft.
0.2	09/26/11	Holland Huang	The version number of this document is change to synchronize with SW package 0.2.
0.3	10/20/11	Jason Hsueh	1. The SW package 0.3 support MCSDK version 2.0.3.15. 2. Add image processing example.
0.4	01/11/12	Holland Huang	1. Add memory read/write function in DSP loader. 2. Driver modification to ensure stability of memory read/write 3. Support both Legacy and MSI interrupt in ipc example.
0.4.6	14/12/12	David Wang	1. Windows driver with 0.6 binary release
0.7.6	21/03/13	David Wang	1. Add LibTi667x Library

Content

1. Introduction 6
1.1. Hardware Description 6
1.2. DSPC-8681E Block Diagram 7
1.3. DDR3 Interface 7
1.4. PCIe Interface 8
1.5. HyperLink Interface 8
1.6. Serial RapidIO Interface 8
1.7. SGMII Interface 8
1.8. DSP Identification 9
1.9. Hardware Environment Setting 9
2. Package Content 13
2.1. Windows Driver 13
2.2. DSP Program Loader Utility 13
2.3. Example: DDR3 Initialization 13
2.4. Example: Simple Web Server 14
2.5. Example: PC/DSP Communication 14
2.6. Example: Image Processing 14
2.7. LibTi667x Library 14
2.8. Patch: Platform Library and NDK Library 15
3. Windows Driver 16
3.1. Host System Requirement 16
3.2. Installation 16
4. DSP Program Loader 18

4.1. DSP Loader Utility 18
4.1.1. Query DSP Information 18
4.1.2. Download DSP Program Image 19
4.1.3. DSP memory read 20
4.1.4. DSP memory write 20
4.1.5. Download DSP binary file 21
4.1.6. Save DSP memory as a binary file 21
5. Reference Implementations 23
5.1. Patch of Platform Library and NDK Library 23
5.1.1. How to Use Patch and Pre-built Library 23
5.1.2. Build Instruction 23
5.2. DSP DDR3 Initialization 25
5.2.1. Build Instruction 25
5.2.2. Load Procedure and Loader Script 25
5.3. Ethernet and Simple Web Server 27
5.3.1. Build Instruction 27
5.3.2. Load Ethernet Example 27
5.4. Communication between PC and DSP 30
5.4.1. Usage 31
5.4.2. dsp_demo.exe Usage 33
5.4.3. DSP Demo Program 34
5.4.4. IPC Demo on SYS/BIOS 35
5.5. Image Processing Demonstration 39
5.5.1. Build Instruction 41

5.5.2. Load Image Processing Example 41

1. Introduction

This document describes how to set up the software configurations for quad-DSP PCIe board, called Lightning (DSPC-8681E), before using it. The Lightning board contains four Texas Instruments TMS320C6678 DSPs with PCIe, HyperLink, Serial RapidIO, and SGMII interfaces.

1.1. Hardware Description

The placement of the Lightning broad is shown in Figure 1−1. Each Lightning board contains four TMS320C6678 (codename Shannon) DSPs, one PLX PEX8624 PCIe switch, and one Xilinx XC3S200AN FPGA. The TMS320C6678 multi-core fixed and floating point digital signal processor is based on advanced KeyStone architecture from Texas Instruments. Each TMS320C6678 on Lightning board is supported by external DDR3 (four 2Gb x16b, 64bits) devices for data and program storage. The four TMS320C6678 devices are connected through PEX8624 PCIe device, which is 24-lane, 6-port PCIe Gen2 switch. The XC3S200AN FPGA device provides the required control signals to the Lightning board.

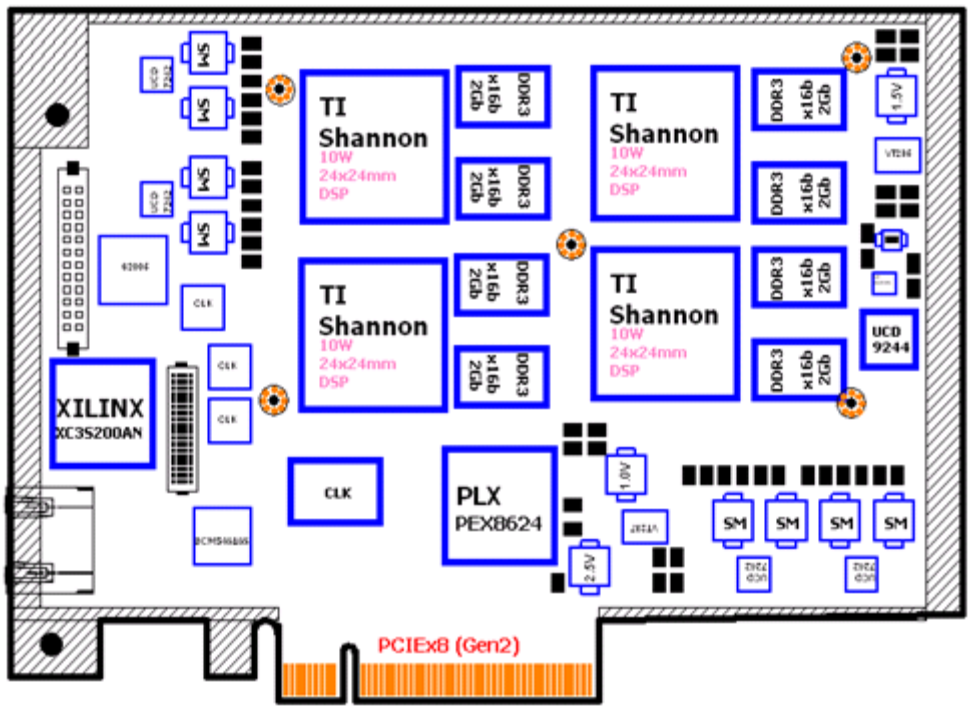

Figure 1-1 DSPC-8681E Placement

1.2. DSPC-8681E Block Diagram

An interface block diagram for the Lightning broad is shown in Figure 1−2. Each TMS320C6678 DSP contains several interfaces such as DDR, HyperLink, Serial RapidIO, and SGMII.

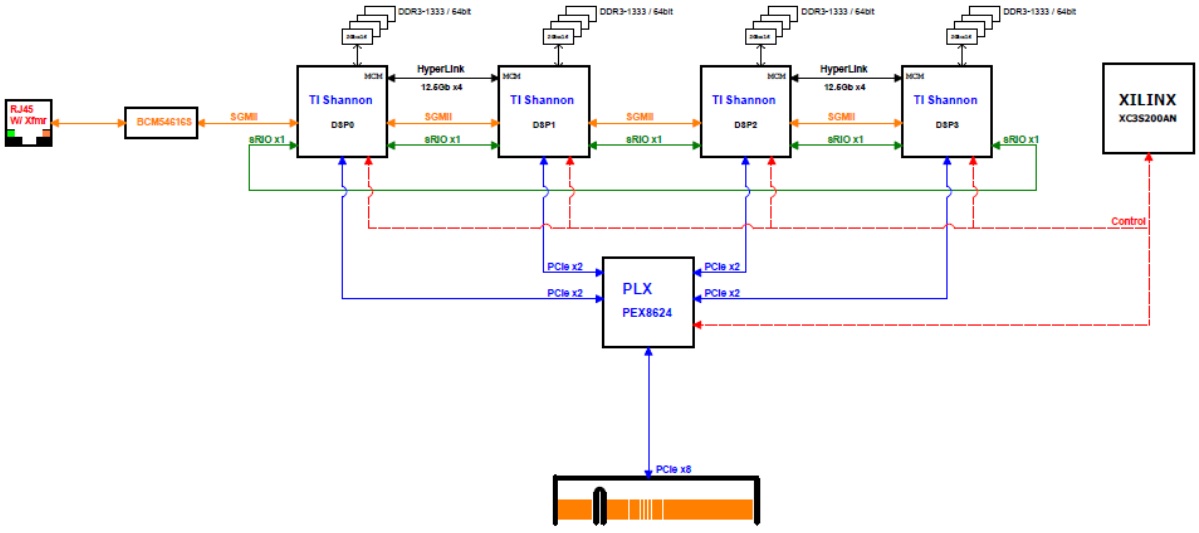

Figure 1-2 DSPC-8681E Interface Block Diagram

1.3. DDR3 Interface

Each TMS320C6678 DSP is connected to four 4Gbit DDR3 memory devices with 64-bit data and 2GB capacity at current implementation. The DDR memory space is ranging from 0x80000000 to 0xFFFFFFFF at DSP device.

Note: Some A101 version board is mounted 2Gbit DDR3 memory devices and 1GB capacity. The DDR memory space of those boards will be ranging from 0x80000000 to 0xBFFFFFFF at DSP device. User can distinguish the HW version by bar code label: 9692868100E is A101, 9692868102E is A103.

1.4. PCIe Interface

Each TMS320C6678 DSP is connected to PEX8624 switch by x2-lane of PCIe Gen2 with 5Gb speed per lane. The PEX8624 PCIe switch will connect the Lightning board to host PC through x8-lane interface.

1.5. HyperLink Interface

Each pair of TMS320C6678 DSP devices are connected by four lanes of HyperLink interface with 50Gbaud rate in between. DSP0 and DSP1 is the first DSP pair and DSP2 and DSP3 is the second DSP pair. DSP0 can exchange data to DSP1 via HyperLlink interface while DSP2 can exchange data to DSP3 via HyperLink interface as well.

1.6. Serial RapidIO Interface

The Lightning board contains a two-lane Serial RapidIO (sRIO) chaining through TMS320C6678 DSP sRIO lane0 and lane1 at 5 Gbaud rate. Each DSP can communicate to the other DSPs through the sRIO interface.

1.7. SGMII Interface

TMS320C6678 DSP contains an on-chip Ethernet switch with two Ethernet interfaces, EMAC0 and EMAC1. TMS320C6678 DSP can connect to another DSP by Ethernet interface without extra Ethernet switch in between. The SGMII interface connection and the topology of the Ethernet link on the Lightning broad is shown in Figure 1−2. The DSP0 on Lightning board contains two SGMII interfaces and EMAC0 is connected to Broadcom BCM54616 Ethernet PHY for external Ethernet access and EMAC1 is connected to EMAC0 of DSP1. EMAC1 of DSP1 is connected to EMAC0 of DSP2. EMAC1 of DSP2 is connecting to EMAC0 of DSP3. Programmers only need to enable Ethernet switch feature of TMS320C6678 DSP

and Ethernet packet will forward to the matched DSP by hardware accelerator of on-chip Ethernet switch without intervention of DSP cores inside.

1.8. DSP Identification

The Lightning board use GPIO[1:2] pins to identify each DSP and the assignment of DSP ID is shown below:

	GPIO 2	GPIO 1
DSP 0	0	0
DSP 1	0	1
DSP 2	1	0
DSP 3	1	1

1.9. Hardware Environment Setting

The Lightning board supports two boot modes: Emulation mode and I2C mode. The user can select boot mode by Switch-1 which is shown in Figure 1-3. The Emulation mode is mainly for JTAG debug. The I2C boot mode is usually selected by Switch-1. DSPC-8681E includes four I2C EEPROMs to support the TMS320C6678 DSPs and each I2C EEPROM contains program for 2-stage boot loader. The 2-stage boot loader will configure PLL and PCIE BAR window when DSP boots up from I2C EEPROM. Table 1-1 and Table 1-2 show the detailed configuration of Switch-1.

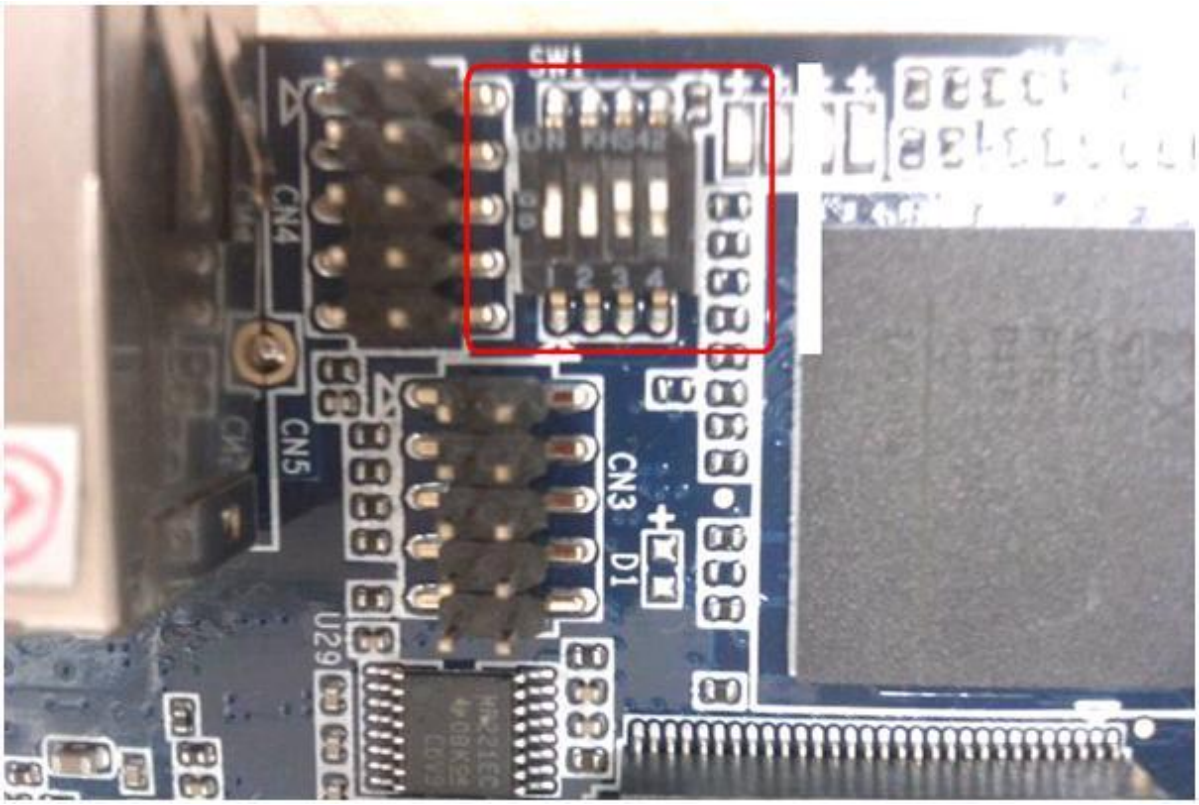

Figure 1-3 I2C Boot Mode (PCIE boot) Setting

Switch 1 pins			
4	3	2	1
Boot mode			Endian

Table 1-1 Switch 1 pin decoding

Bit	Field	Value	Description
4~2	Boot mode	111	Emulation boot mode
		110	I2C boot mode(Boot from address 0x51) 32bits address BAR setting
		100	I2C boot mode(Boot from address 0x50)

		000~101	64bits address BAR setting Reserved
1	Endian	0 1	Little endian Big endian

Table 1-2 Switch-1 Configuration Bit Field Description

CAUTION! It is a known issue when DSPC-8681E boots through secondary boot loader by I2C boot mode the DSP may not complete boot process before BIOS scanning PCIe device tree. Usually DSPC-8681E can be detected after restart BIOS or reboot Linux system. The four PCIe switch LEDs should begin flashing to indicate the status of PCIe interface connection to individual DSP. The placement of 4 LEDs is shown in Figure 1-4 (for DSP 2 and DSP 3) and Figure 1-5 (for DSP 0 and DSP 1).

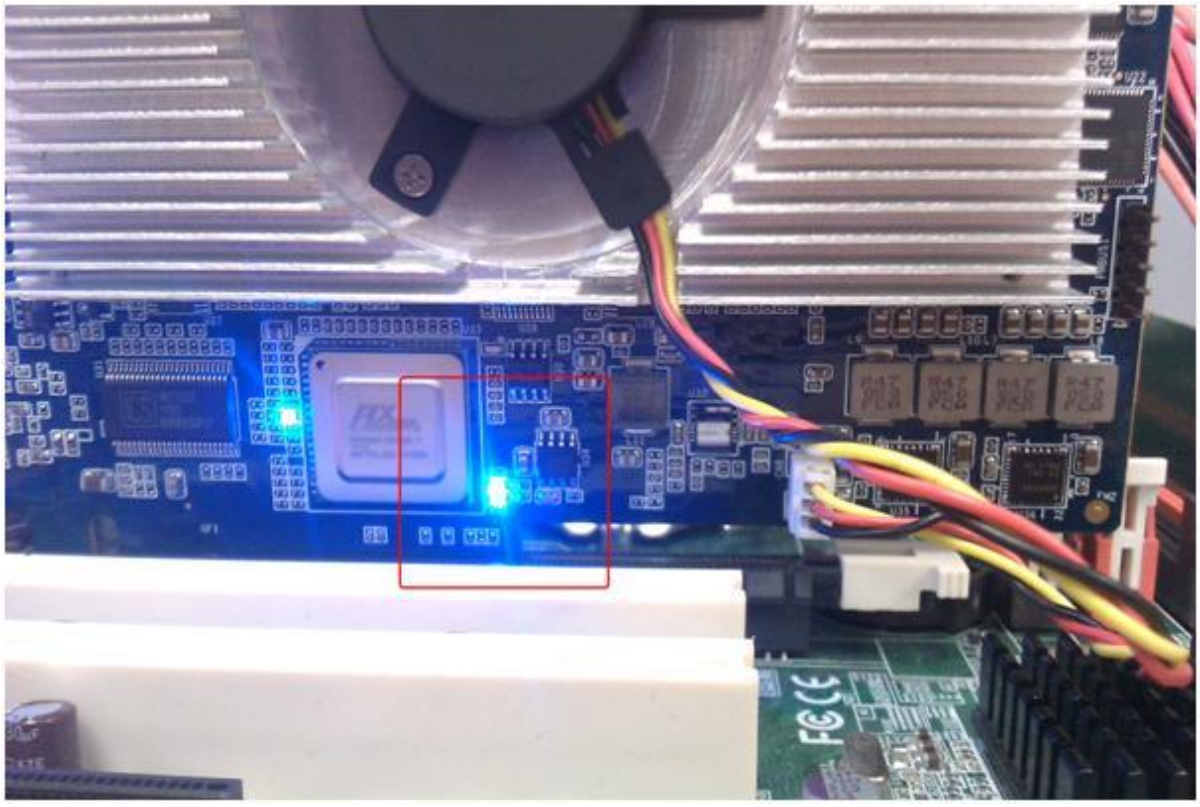

Figure 1-4 Two PCIe Switch LEDs on Front Side

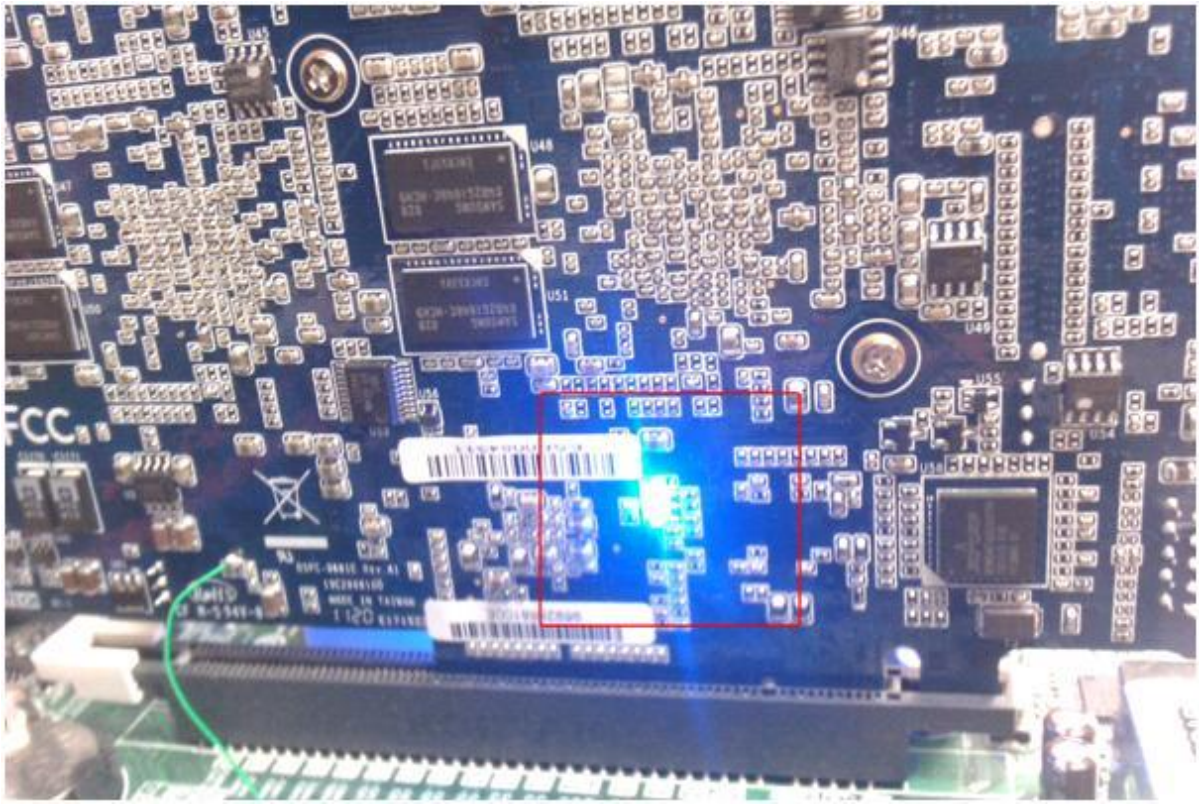

Figure 1-5 Two PCIe Switch LEDs on Back Side

2. Package Content

This package is created to help customer quickly boot DSP through PCIE, the package includes:

Path	Purpose
driver	1. Windows Driver 2. DSP Program Loader Utility
examples\ddr3	Example: DDR3 Initialization
examples\image_processing	Example: Image Processing using Multi-Core
examples\ipc	Example: PCDSP Communication
examples\script	Common demo related scripts
examples\web	Example: Simple Web Server
LibTi667x	Library for DSPC-868x Operation
patch	Patch: Platform Library and NDK Library of PDK C6678 1.1.2.5 (inside MCSDK 2.1.2.5)

Table 2-1 Package content list

2.1. Windows Driver

The driver package contains headers and x86, x64 Windows driver.

2.2. DSP Program Loader Utility

DSP program loader utility contains the source of the loader and a hex parser which is used to load hex files into DSPs and notify DSPs to run program.

2.3. Example: DDR3 Initialization

The DDR3 initialization example contains CCS project settings to build a boot image. This program will initialize DDR and wait loader utility to load the next program. A file format conversion tool provided by TI is also included and can be used to convert .out file format into .hex file format.

2.4. Example: Simple Web Server

A web demo example contains CCS project settings to build an image. It can set up a web server so user can use network browser to access the web page stored in the DSP. This program is modified from TI MCSDK example which is located in the mcsdk_2_1_2_5\examples\ndk\client. The each DSP will be configured with a static IP instead of DHCP.

2.5. Example: PC/DSP Communication

This example contains two parts, a DSP image and a PC utility (sourced included). The dsp folder included contains CCS project settings of building an image. This example provides sample codes on how to communicate between PC and DSP.

2.6. Example: Image Processing

The image processing demo example contains two CCS project settings to build the demo images. This application will run TI image processing kernels (imagelib) on multiple cores to do image processing (eg: edge detection, etc) on an input image. This program is modified from TI's MCSDK example in the mcsdk_2_1_2_5\demos\image_processing\ipc. The each DSP will be configured with a static IP instead of DHCP.

2.7. LibTi667x Library

LibTi667x is a library that helps customers to operate DSPC-868xE on Windows. With the library, customers can work in both adapter and chip perspectives (Figure 2-1). The base class is Ti667xDevice, which stands for a TI C6678 chip. Dspc868xCard is a set of 4/8 C6678. Dspc8681Card extends Dspc868xCard, so does Dspc8682Card. All the classes provide utility functions to help users enumerate, instantiate, and manipulate the hardware layer. Related documentations are located in LibTi667x\doc\html subfolder. Please refer to the index.htm for API manual and class diagrams.

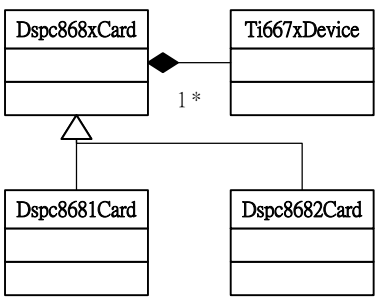

Figure 2-1 LibTi667x Class Diagram

2.8. Patch: Platform Library and NDK Library

There are some differences between the Lightning board and C6678 EVM, hence, developer should patch these files in the TI PDK before using it. The modification is listed as below:

1. DSPC-8681E uses EMAC0 to connect to BCM54616 Ethernet PHY. This patch adds the initialization of SGMII port 0 and change settings of SGMII port 0 and port 1 for BCM54616 Ethernet PHY.
2. DSPC-8681E uses different DDR memory devices, the parameters for initialization of DDR controller is not the same as C6678 EVM. This patch supports the DDR memory device which is mounted on DSPC-8681E.
3. The reference clocks of DDR and SGMII is not the same as C6678 EVM and this patch modifies the relevant MPY settings.

3. Windows Driver

3.1. Host System Requirement

A reference of the OS used to develop and execute this software release is:

1. **Microsoft Windows**: Compatible with Windows XP (x86/x64), Vista(x86/x64), Windows 7(x86/x64), and Windows 8(x86/x64).
2. **DSP development tool**: TI Code Composer Studio v5.1 or higher, TI MCSDK for TMS320C66x Processors V2.1.2.5, please refer to web site: http://software-dl.ti.com/sdoemb/sdoemb_public_sw/bios_mcsdk/02_01_02_05/index_FDS.html

3.2. Installation

User can install driver in two ways, one is to run the Driver Installation Wizard, and the other is the traditional PnP way, especially for Windows XP.

1. Driver Wizard Installation

   Double click on Install.bat in the folder of your platform (i.e., x86 or x64), and Device Driver Installation Wizard will help you install the driver. See Figure 3-1 and Figure 3-2.

2. Manual Installation

   Browse to the driver's folder in Found New Hardware Wizard or Update Driver (Figure 3-3) dialog, and go through the driver installation procedure.

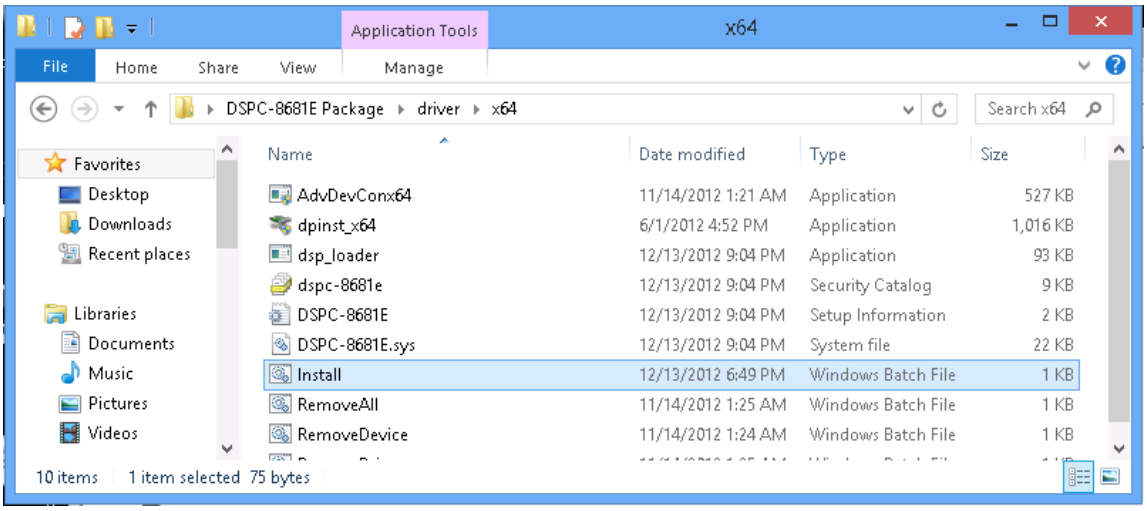

Figure3-1 Install.bat

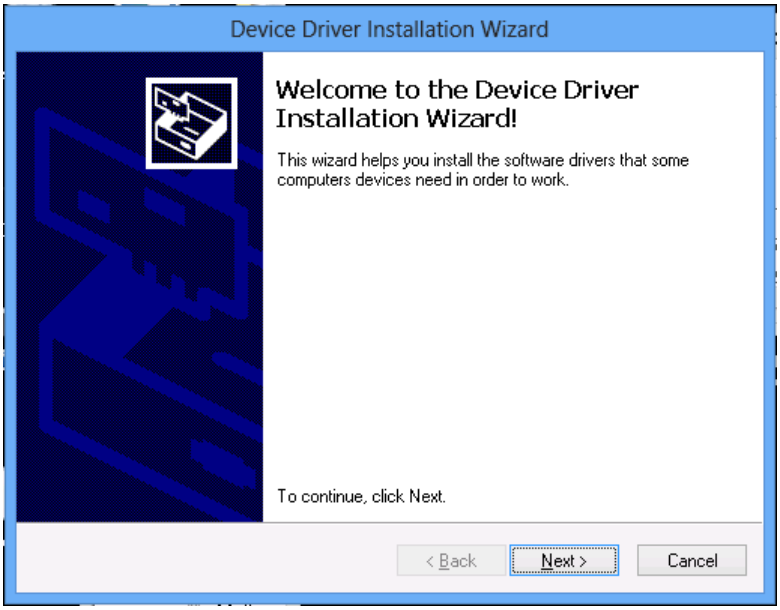

Figure 3-2 Device Driver Installation Wizard

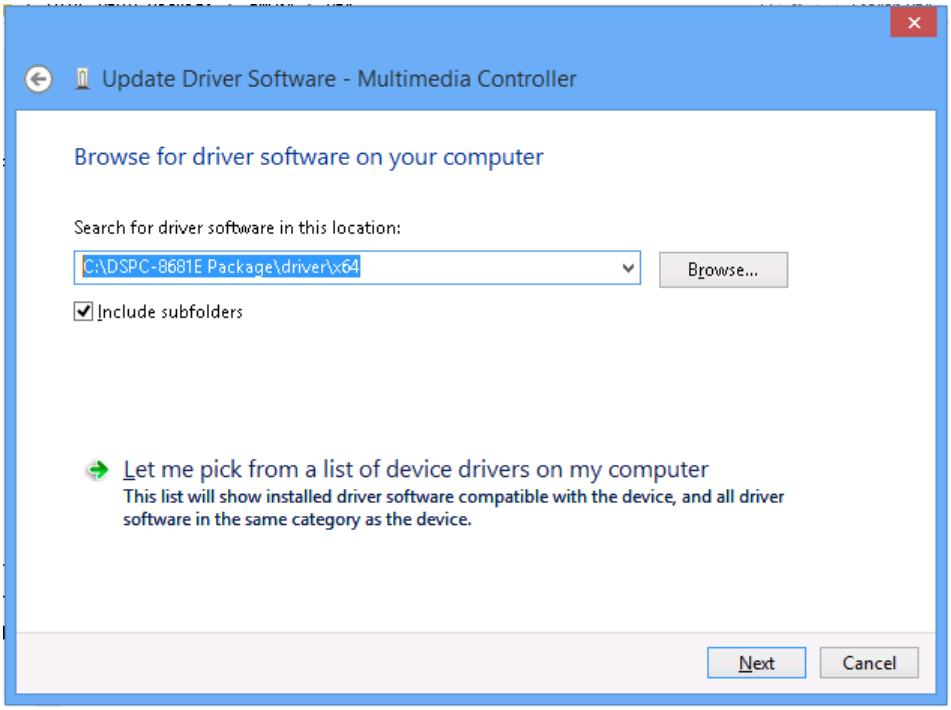

Figure 3-3 Update Driver dialog

4. DSP Program Loader

After the whole system booting up, all DSP chips stay in idle mode. The PC is responsible to download DSP codes to every chip and awaken DSPs to execute the loaded codes. The loader consists of a driver and a utility running in Windows environment. This package contains source code of the program loader. The developer must rebuild and install them before starting using the Lightning board.

4.1. DSP Loader Utility

DSP loader offers the functions to load the program into DSP memory and notify the DSP to run program.

4.1.1. Query DSP Information

The command syntax is:

dsp_loader query list or dsp_loader query -l

dsp_loader query [chip#]

The command is to display the PCI information of DSP which are installed in the system. The more detailed information will be displayed when user specify the [chip#] parameter.

The following two examples demonstrate the result of query command when PC system install with two Lightning card and query the detailed information of DSP#7.

```
\>dsp_loader query -l
Card 0:
       [Chip 0]
       [Chip 1]
       [Chip 2]
       [Chip 3]

\>dsp_loader query 3
============================================
Chip: 3
       PCI Bridge from 8 to 0
       PCI Bus Num: 7
       Vendor ID: 0x104C      Device ID: 0xB005
       Class: 0x00000004
       Header Type: 0  Irq Pin: 1
       BAR Configuration:
       Start                 |      Length
       0x00000000F7400000      |      00004096
       0x00000000D7000000      |      16777216
       0x00000000D6000000      |      16777216
```

```
        0x00000000D4000000      |       33554432
================================================
```

4.1.2. Download DSP Program Image

The command syntax is:

dsp_loader load [chip#] [core#] [image entry point] [image file name (hex)]

The command is to download a DSP program (DSP image) into to RAM of a specified DSP. The detailed description of each parameter is shown below:

1. **[chip#]:** the [chip#] are the number of DSPs attached to the PC. Since there are four DSP devices on the Lightning board, this parameter can be set into 0 ~ 3 for those PC systems installed with one Lightning card. For those PC systems installed with two Lightning cards, there will be eight chips available to the PC systems and the parameter can be set into 0 ~ 7.
2. **[core#]:** [core#] is used to notify individual core (range from 0 to 7) within DSP to run.
3. **[image entry point]:** [image entry point] is the start address of the loaded image. User can find the “entry point symbol” of "_c_int00" in the map file. For example, init.map information is displayed in List 3-1. The reader can find the entry point of the program in the top of map file.

```
******************************************************************************
                  TMS320C6x Linker PC v7.2.1
******************************************************************************
>> Linked Mon Aug 15 15:03:07 2011

OUTPUT FILE NAME:     <../bin/init.out>
ENTRY POINT SYMBOL: "_c_int00"    address: 008362a0
```

List 4-1 entry point in the init.map

4. **[image file name]:** [image file name] is the full path of hex file name which is loaded into DSP.

The following example demonstrates how to load C:\Lightning_PCIE\bin\DSPC8681E\init.hex (DSP image for DDR initialization) into DSP#1 and use CPU#0 to run DSP image.

```
\>dsp_loader load 1 0 0x008362A0 C:\Lightning_PCIE\bin\DSPC8681E\init.hex
Load HEX image: C:\Lightning_PCIE\bin\DSPC8681E\init.hex to 1:0, start address
0x008362A0
Load HEX OK
```

Note: Image entry point depends on DSP image. The image entry point of init.hex (DSP image) uses address 0x008362A0 as local address for each CPU. Individual local CPU address can also be transferred to DSP global address with offset. For example, CPU#0 local address 0x00800000 is equal to DSP global address 0x1080000. CPU#1 local address 0x00800000 is equal to DSP global address 0x1180000.

4.1.3. DSP memory read

The command syntax is:

dsp_loader rmem [chip#] [address]

The command is to read a 32bits-DWORD from DSP. The detailed description of each parameter is shown below:

1. **[chip#]:** the [chip#] are the number of DSPs attached to the PC.
2. **[address]**: read data address

The following example is to read DSP#2 data at address 0x10800000.

```
\>dsp_loader rmem 2 0x10800000
0x01bc54f6
```

4.1.4. DSP memory write

The command syntax is:

dsp_loader load [chip#] [address][value]

The command is to write a 32bits-DWORD into DSP memory. The detailed description of each parameter is shown below:

1. **[chip#]:** the [chip#] are the number of DSPs attached to the PC.
2. **[address]**: read data address
3. **[value]**: written data

The following example writes data 0x55AA55AA into DSP#2 at address 0x10800000.

```
\>dsp_loader wmem 2 0x10800000 0x55aa55aa

\>dsp_loader rmem 2 0x10800000
0x55aa55aa
```

4.1.5. Download DSP binary file

The command syntax is:

dsp_loader loadbinary [chip#][address][size][transfer type][bin file name]

The command is to write a bin file into DSP memory. The detailed description of each parameter is shown below:

1. **[chip#]:** the [chip#] are the number of DSPs attached to the PC.
2. **[address]**: written data address
3. **[size]**: written data size, 0 for all data of file.
4. **[transfer type]**: 0 for CPU memcpy, 1 for DMA.
5. **[bin file name]**: [bin file name] is the full path of binary file name which is loaded into DSP.

The following example writes a jpg file into DSP#1 at DDR beginning address 0x80000000 by using DMA.

```
\>dsp_loader loadbinary 1 0x80000000 0 1 test_image.jpg
Load Binary file: test_image.jpg to DSP1, start address 0x80000000, Size
0x00000000
Written to dsp 7496169 bytes
Time measured: 16225 us
Load Binary OK
```

4.1.6. Save DSP memory as a binary file

The command syntax is:

dsp_loader savebinary [chip#][address][size][transfer type][bin file name]

The command is to read a DSP memory section and save the data as a binary file. The detailed description of each parameter is shown below:

6. **[chip#]:** the [chip#] are the number of DSPs attached to the PC.
7. **[address]**: read data address
8. **[size]**: read data size
9. **[transfer type]**: 0 for CPU memcpy, 1 for DMA.
10. **[bin file name]**: [bin file name] is the full path of binary file name which is saved.

The following example read 7496169 bytes from DSP#1 at DDR beginning address 0x80000000 by using DMA, and saves as test_image_output.jpg.

```
\>dsp_loader loadbinary 1 0x80000000 0 1 test_image.jpg
```

```
Load Binary file: test_image.jpg to DSP1, start address 0x80000000, Size
0x00000000
Written to dsp 7496169 bytes
Time measured: 16225 us
Load Binary OK

\>dsp_loader savebinary 1 0x80000000 7496169 1 test_image_output.jpg
Save Binary file: test_image_output.jpg from DSP 1, start address 0x80000000 Size
0x007261e9
Saved from dsp 7496169 bytes
Time measured: 21871 us
Save Binary OK
```

5. Reference Implementations

5.1. Patch of Platform Library and NDK Library

The example programs have to link with DSPC8681 platform library and NDK library. A developer has to install MCSDK first and applies the provided patch. The default path of MCSDK in Windows is “C:\Program Files\Texas Instruments\” or “C:\ti\”.

5.1.1. How to Use Patch and Pre-built Library

1. Copy Lightning_PCIE\patch\pdk_C6678_1_1_2_5.
2. Paste to C:\Program Files\Texas Instruments\pdk_C6678_1_1_2_5.
3. The pre-built libraries are included in the provided patch, a developer can use these libraries directly.

5.1.2. Build Instruction

Steps to build platform lib are listed below:

1. Import the CCS project.
   1.1 Click Project->Import Existing CCS/CCE Eclipse Projects.
   1.2 Select pdk_C6678_1_1_2_5\packages\ti\platform\dspc8681\platform_lib
2. Refer to Figure 5-1 ~ Figure 5-3 and select "Lite" as active configuration (in CCSv5, Project->Properties)

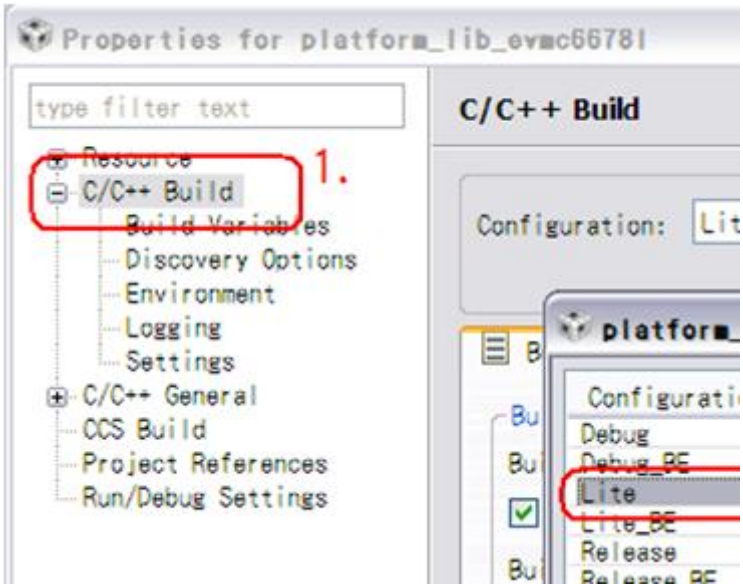

Figure 5-1 Select Lite as Active Configuration (Step 1)

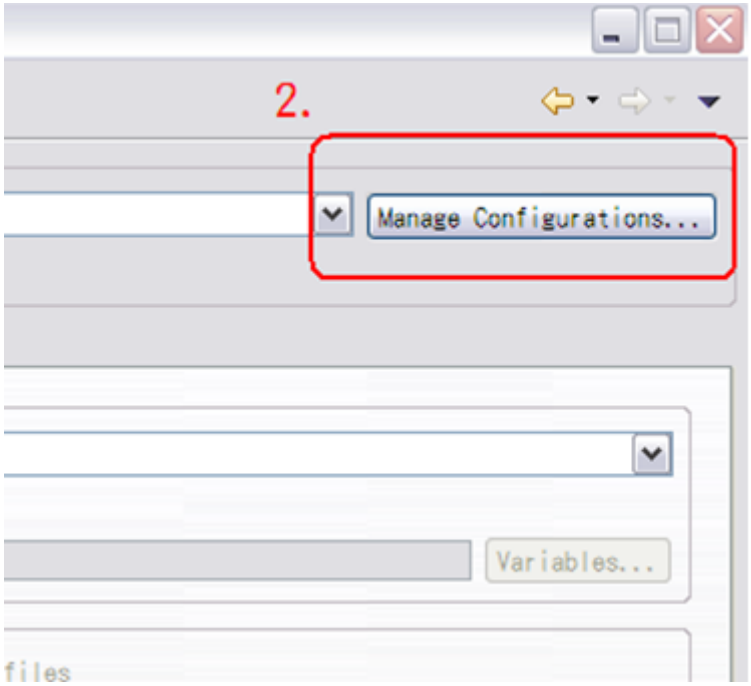

Figure 5-2 Select Lite as Active Configuration (Step 2)

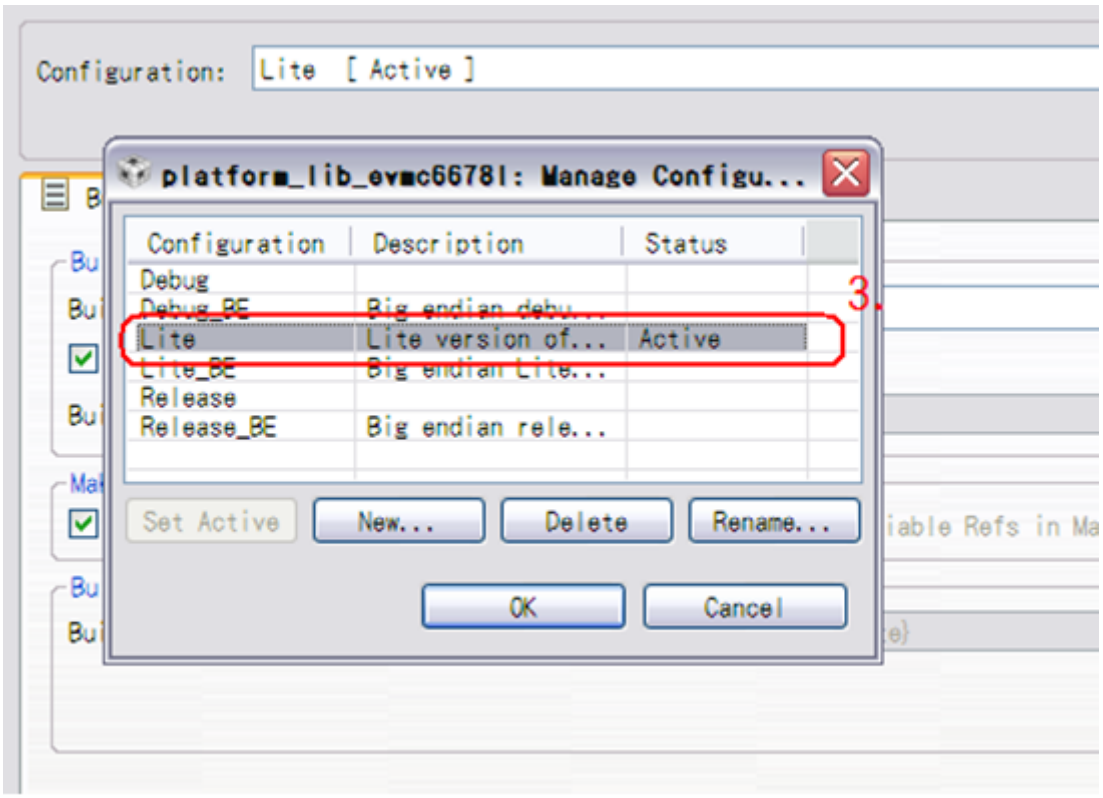

Figure 5-3 Select Lite as Active Configuration (Step 3)

3. Clean the platform_lib_dspc8681 project and re-build the project. After build is completed the ti.platform.dspc8681.lite.lib will be generated under the directory: "pdk_C6678_1_1_2_5\packages\ti\platform\dspc8681\platform_lib\lib\debug"
4. Repeat step 2 and step 3, select "Debug" as active configuration and re-build the project. ti.platform.dspc8681.ae66 will be generated under the same directory

Steps to build ndk lib are listed below:

1. Import the CCS project from "pdk_C6678_1_1_2_5\packages\ti\transport\ndk\nimu" directory (in CCSv5, Project->Import Existing CCS/CCE Eclipse Projects)
2. Clean the nimu_eth_evmc6678l project and re-build the project. After build is completed, ti.transport.ndk.nimu.ae66 will be generated under the directory: "pdk_C6678_1_1_2_5\packages\ti\transport\ndk\nimu\lib\debug"

5.2. DSP DDR3 Initialization

The Boot ROM codes only initialize L2 internal memory when booting from PCIE boot mode. The on-board DDR3 control registers need to be explicitly initialized by this supplied example program. User has to initialize DDR3 control registers before loading the application into DSP. After initialization of DDR3 control registers, this program will clear boot address and wait dsp_loader to write new entry point in boot address. When boot address is updated, this program will jump to new entry point and start to run the next program.

5.2.1. Build Instruction

Steps to build DDR3 initialization program are listed below:

1. Import CCS project
   1.1 Click Project->Import Existing CCS/ CCE Eclipse Projects.
   1.2 Select Lightning_PCIE\examples\ddr3\evmc6678l directory.
2. Select DSPC8681E as active configuration.
3. Clean and rebuild
   3.1 When the rebuild succeeds, init.out and init.map will be generated in Lightning_PCIE\examples\ddr3\evmc6678l\bin\DSPC8681E.

5.2.2. Load Procedure and Loader Script

The procedure is composed of three jobs,

1. Convert .out to .hex (Hex6x.exe)
2. Set PLL Multiplier (dsp_loader.exe)
3. Load .hex to DSP (dsp_loader.exe)

There are two loader scripts, init_1000.bat and init_1250.bat for users to initialize DSP. They reside in the directory Lightning_PCIE\examples\script\DSPC8681E. They load the init.hex in Lightning_PCIE\bin. DSP initialization can be done by invoking the loader scripts. The difference between init_1000.bat and init_1250.bat is that the former runs DSP at 1GHz, the latter overclocks DSP to 1.25GHz. There is the prebuilt binary bundled in Lightning_PCIE\bin. Users can initialize DSP DDR module without building the image from source. However, when running the script, it will show your Windows processor architecture (x86 and x64), version of your DSP (PG1 and PG2), and running frequency, e.g., 1GHz, 1.2GHz, or 1.25GHz. Double click the desired loader script and the initialization will be done after the execution.

Notice: At present, we only guarantee the stability for PG2 version of C6678 to run at 1GHz.

```
Microsoft (R) Windows Script Host Version 5.8
Copyright (C) Microsoft Corporation. All rights reserved.

Windows Platform: x64

Loading DSP0
DSP type: silicon Version = PG2.0
Frequency at 1GHz

Load HEX image: ..\..\..\bin\DSPC8681E\init.hex to 0:0, start address
0x00830000Load HEX OK

Loading DSP1
DSP type: silicon Version = PG2.0
Frequency at 1GHz

Load HEX image: ..\..\..\bin\DSPC8681E\init.hex to 1:0, start address
0x00830000Load HEX OK

Loading DSP2
DSP type: silicon Version = PG2.0
Frequency at 1GHz

Load HEX image: ..\..\..\bin\DSPC8681E\init.hex to 2:0, start address
0x00830000Load HEX OK

Loading DSP3
DSP type: silicon Version = PG2.0
Frequency at 1GHz

Load HEX image: ..\..\..\bin\DSPC8681E\init.hex to 3:0, start address
0x00830000Load HEX OK
```

5.3. Ethernet and Simple Web Server

The Ethernet program is modified from the example codes in TI MCSDK. This example implements a simple web server running on DSP. The Ethernet port on the bracket of the Lightning board must be connected to an external Ethernet switch (support gigabit rate) before running this example. Each DSP has a fixed IP number that is determined by its order. The pre-given IP addresses are shown below. The user can use a browser to view the simple web page provided by this simple web server.

	IP
DSP 0	192.168.1.101
DSP 1	192.168.1.102
DSP 2	192.168.1.103
DSP 3	192.168.1.104

5.3.1. Build Instruction

Steps to build web server program are listed below:

1. Import CCS project
   1.1 Click Project->Import Existing CCS/ CCE Eclipse Project.
   1.2 Select Lightning_PCIE\examples\web\client\evmc6678l directory.
2. Select DSPC8681E as active configuration.
3. Clean and rebuild
   3.1 Clean and rebuild the client_evmc6678l project.
   3.2 After build is completed, client_evmc6678l.out and client_evmc6678l.map will be generated in Lightning_PCIE\examples\web\client\evmc6678l\DSPC8681E directory

5.3.2. Load Ethernet Example

The batch script Lightning_PCIE\examples\script\DSPC8681E\ethernet.bat is already there to setup Ethernet program on each DSP automatically. In order to load this example, there are three steps to follow:

1. Initialize DDR3 module (init_1000.bat).
2. Run Ethernet loader script (ethernet.bat).
   2.1 Convert .out to .hex
   2.2 Load the image to each DSP.

```
Translating to Intel format...

"C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\web\client\evmc6678l\D
SPC8681E\client_evmc6678l.out"   ==> .text:_c_int00

"C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\web\client\evmc6678l\D
SPC8681E\client_evmc6678l.out"   ==> .text

"C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\web\client\evmc6678l\D
SPC8681E\client_evmc6678l.out"   ==> .const

"C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\web\client\evmc6678l\D
SPC8681E\client_evmc6678l.out"   ==> .switch.1

"C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\web\client\evmc6678l\D
SPC8681E\client_evmc6678l.out"   ==> .vecs

"C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\web\client\evmc6678l\D
SPC8681E\client_evmc6678l.out"   ==> .switch.2

"C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\web\client\evmc6678l\D
SPC8681E\client_evmc6678l.out"   ==> .cinit
Load HEX image:
C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\..\bin\DSPC8681E\client
_evmc6678l.hex to 0:0, start address 0x80300000
Load HEX OK
Load HEX image:
C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\..\bin\DSPC8681E\client
_evmc6678l.hex to 1:0, start address 0x80300000
Load HEX OK
Load HEX image:
C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\..\bin\DSPC8681E\client
_evmc6678l.hex to 2:0, start address 0x80300000
Load HEX OK
Load HEX image:
C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\..\bin\DSPC8681E\client
_evmc6678l.hex to 3:0, start address 0x80300000
Load HEX OK
```

3. Check the result with a web browser. The URL of DSPs are http://192.168.1.10X, X=1~4. The result is shown in Figure 5-4 and Figure 5-5.

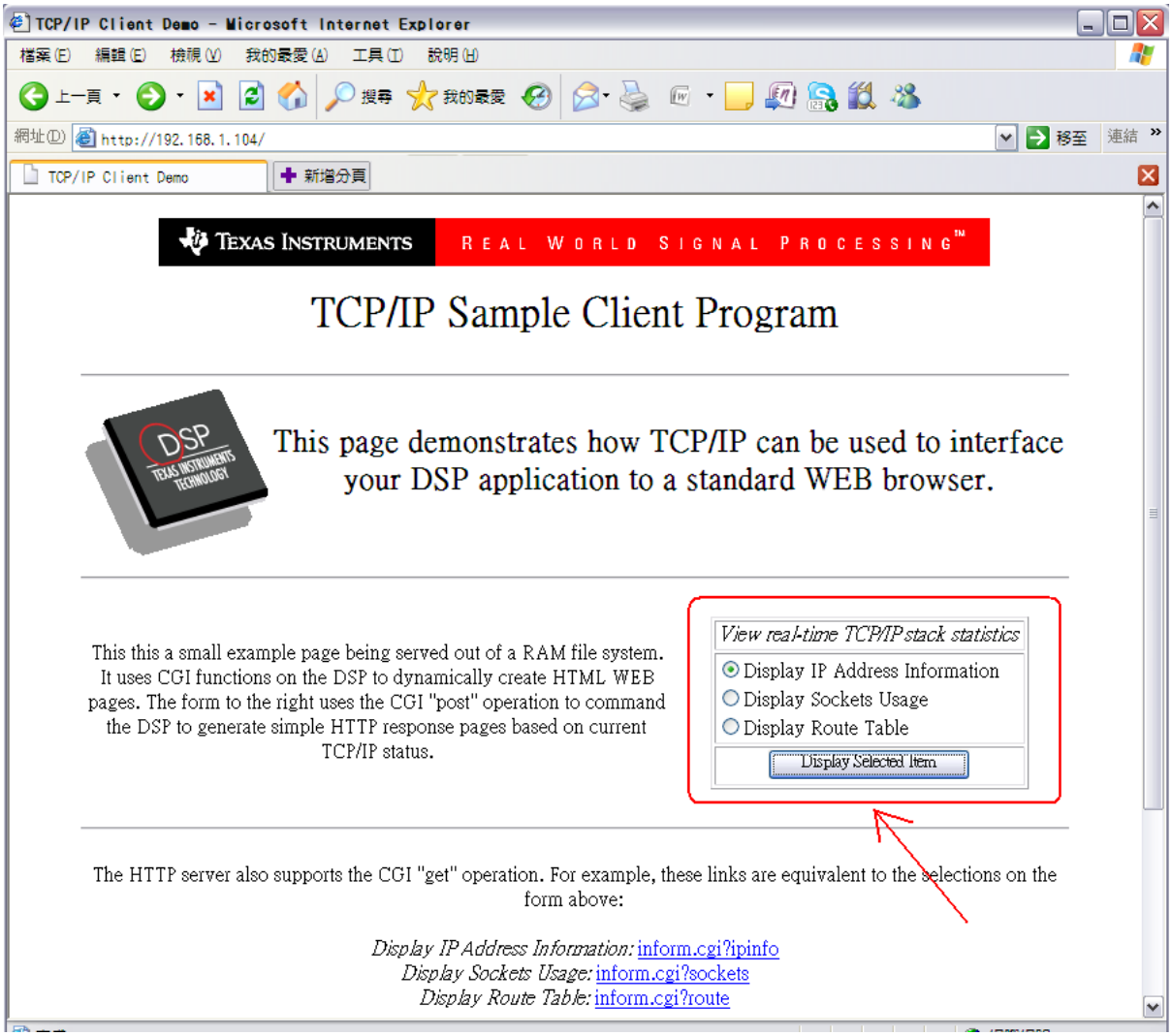

Figure 5-4 TCP/IP Demo Page

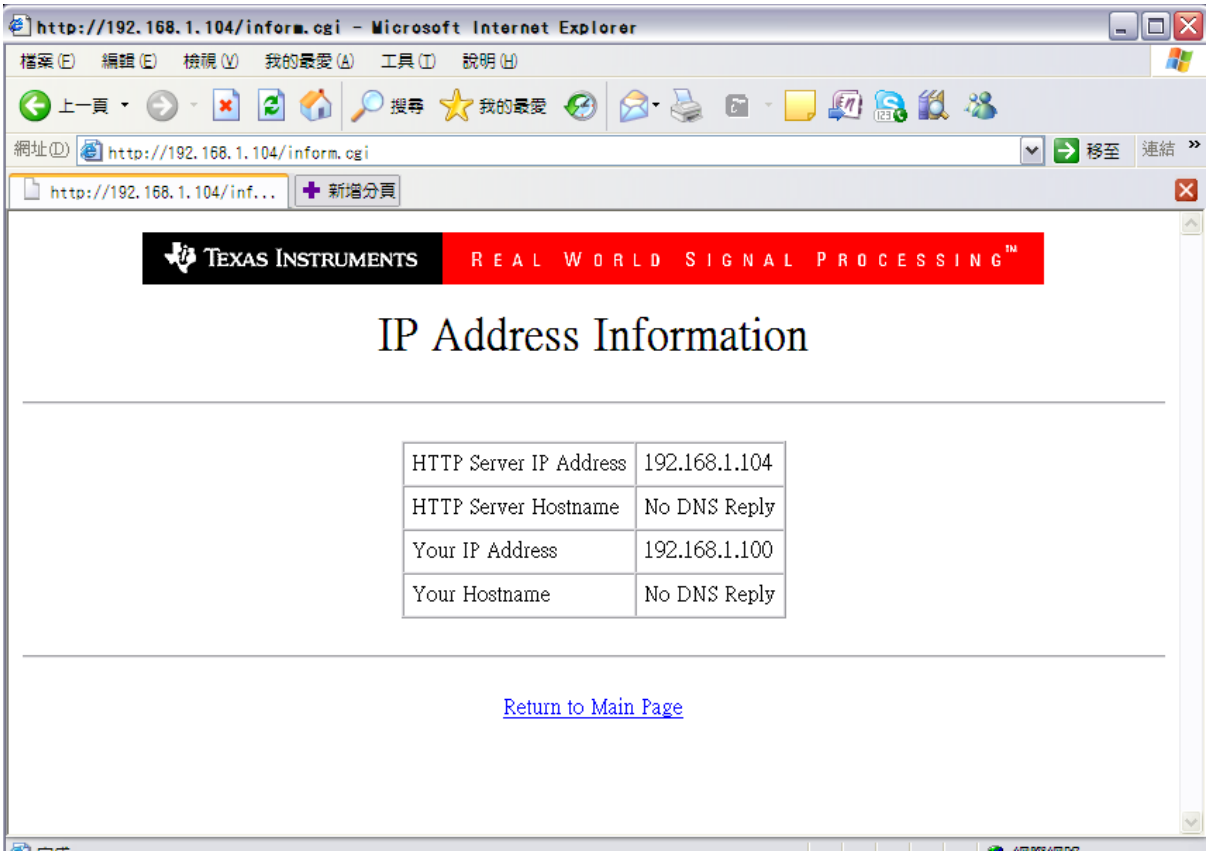

Figure 5-5 IP Address Information page

5.4. Communication between PC and DSP

This example demonstrates several DSP manipulation functions, including:

1. PC – DSP Memory Write
2. DSP – PC Memory Read
3. PC – DSP Interrupt
4. DSP – PC Interrupt
5. Console Output Emulation

   The implementation enables the DSP to display messages to PC. This could be helpful when developing and debugging DSP applications.

5.4.1. Usage

Lightning_PCIE\examples\DSPC8681E\ipc.bat will do following four jobs.

1. Convert .out to .hex.
2. Load demo_evm6678l.hex to the specified DSP.
3. Run the PC – DSP intercommunication demo (repeat 1000 times).
4. Run the console output demo.

Run DDR initialization batch (init_1000.bat) first, and then execute ipc.bat, it will ask you for the DSP ID you desire to load. The following example captures the result of running Lightning_PCIE\examples\script\ DSPC8681E\ipc.bat.

```
Enter DSP ID [0~3]:1
1
Translating to Intel format...

"C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\ipc\dsp\evmc6678l\bin\DSP
C8681E\demo_evm6678l.out"   ==> .text:_c_int00

"C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\ipc\dsp\evmc6678l\bin\DSP
C8681E\demo_evm6678l.out"   ==> .text

"C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\ipc\dsp\evmc6678l\bin\DSP
C8681E\demo_evm6678l.out"   ==> .const

"C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\ipc\dsp\evmc6678l\bin\DSP
C8681E\demo_evm6678l.out"   ==> .csl_vect

"C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\ipc\dsp\evmc6678l\bin\DSP
C8681E\demo_evm6678l.out"   ==> .switch

"C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\ipc\dsp\evmc6678l\bin\DSP
C8681E\demo_evm6678l.out"   ==> .cinit
Load HEX image:
C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\..\bin\DSPC8681E\demo_evm6
678l.hex to 3:0, start address 0x00840000
Load HEX OK
DDR of DSP is initialized, ready to write dummy data to DSP
dump dummy_buffer before DSP operation:

000000013F1C5020       00000000 00000001 00000002 00000003 00000004 00000005
00000006 00000007
000000013F1C5040       00000008 00000009 0000000a 0000000b 0000000c 0000000d
0000000e 0000000f
000000013F1C5060       00000010 00000011 00000012 00000013 00000014 00000015
00000016 00000017
000000013F1C5080       00000018 00000019 0000001a 0000001b 0000001c 0000001d
0000001e 0000001f
000000013F1C50A0       00000020 00000021 00000022 00000023 00000024 00000025
00000026 00000027
```

```
000000013F1C50C0      00000028 00000029 0000002a 0000002b 0000002c 0000002d
0000002e 0000002f
000000013F1C50E0      00000030 00000031 00000032 00000033 00000034 00000035
00000036 00000037
000000013F1C5100      00000038 00000039 0000003a 0000003b 0000003c 0000003d
0000003e 0000003f
000000013F1C5120      00000040 00000041 00000042 00000043 00000044 00000045
00000046 00000047
000000013F1C5140      00000048 00000049 0000004a 0000004b 0000004c 0000004d
0000004e 0000004f
000000013F1C5160      00000050 00000051 00000052 00000053 00000054 00000055
00000056 00000057
000000013F1C5180      00000058 00000059 0000005a 0000005b 0000005c 0000005d
0000005e 0000005f
000000013F1C51A0      00000060 00000061 00000062 00000063 00000064 00000065
00000066 00000067
000000013F1C51C0      00000068 00000069 0000006a 0000006b 0000006c 0000006d
0000006e 0000006f
000000013F1C51E0      00000070 00000071 00000072 00000073 00000074 00000075
00000076 00000077
000000013F1C5200      00000078 00000079 0000007a 0000007b 0000007c 0000007d
0000007e 0000007f
000000013F1C5220      00000080 00000081 00000082 00000083 00000084 00000085
00000086 00000087
000000013F1C5240      00000088 00000089 0000008a 0000008b 0000008c 0000008d
0000008e 0000008f
000000013F1C5260      00000090 00000091 00000092 00000093 00000094 00000095
00000096 00000097
000000013F1C5280      00000098 00000099 0000009a 0000009b 0000009c 0000009d
0000009e 0000009f
dummy data has already been changed by DSP
dump dummy_buffer after DSP operation:

000000013F1C5020      00000001 00000002 00000003 00000004 00000005 00000006
00000007 00000008
000000013F1C5040      00000009 0000000a 0000000b 0000000c 0000000d 0000000e
0000000f 00000010
000000013F1C5060      00000011 00000012 00000013 00000014 00000015 00000016
00000017 00000018
000000013F1C5080      00000019 0000001a 0000001b 0000001c 0000001d 0000001e
0000001f 00000020
000000013F1C50A0      00000021 00000022 00000023 00000024 00000025 00000026
00000027 00000028
000000013F1C50C0      00000029 0000002a 0000002b 0000002c 0000002d 0000002e
0000002f 00000030
000000013F1C50E0      00000031 00000032 00000033 00000034 00000035 00000036
00000037 00000038
000000013F1C5100      00000039 0000003a 0000003b 0000003c 0000003d 0000003e
0000003f 00000040
000000013F1C5120      00000041 00000042 00000043 00000044 00000045 00000046
00000047 00000048
000000013F1C5140      00000049 0000004a 0000004b 0000004c 0000004d 0000004e
0000004f 00000050
```

```
000000013F1C5160       00000051 00000052 00000053 00000054 00000055 00000056
00000057 00000058
000000013F1C5180       00000059 0000005a 0000005b 0000005c 0000005d 0000005e
0000005f 00000060
000000013F1C51A0       00000061 00000062 00000063 00000064 00000065 00000066
00000067 00000068
000000013F1C51C0       00000069 0000006a 0000006b 0000006c 0000006d 0000006e
0000006f 00000070
000000013F1C51E0       00000071 00000072 00000073 00000074 00000075 00000076
00000077 00000078
000000013F1C5200       00000079 0000007a 0000007b 0000007c 0000007d 0000007e
0000007f 00000080
000000013F1C5220       00000081 00000082 00000083 00000084 00000085 00000086
00000087 00000088
000000013F1C5240       00000089 0000008a 0000008b 0000008c 0000008d 0000008e
0000008f 00000090
000000013F1C5260       00000091 00000092 00000093 00000094 00000095 00000096
00000097 00000098
000000013F1C5280       00000099 0000009a 0000009b 0000009c 0000009d 0000009e
0000009f 000000a0
.
.
.

Synchronizing ...
 done.
=====================
PCIe Hello World Example, this is DSP1
Debug: GEM-INTC Configuration Completed
Debug: CPINTC-0 Configuration...
Debug: CPINTC-0 Configuration Completed
```

5.4.2. dsp_demo.exe Usage

dsp_demo.exe provides two demo functions, one is the intercommunication demo, the other is the virtual console demo.

The demo command is used to display the negotiation between DSP and PC host. dsp_demo.exe will wait the interrupt signal which is sent from PCIe driver and perform the data blocks read/write. The demo repeats 1000 times. [chip#] (From DSP#0 to DSP#3) parameter selects which DSP will be accessed by PC.

The command syntax is:

dsp_demo demo [chip#]

The virtual console demo creates a virtual console to display the debug message by the program running in specific DSP (chip# from DSP#0 to DSP#3) and cores (core# from CPU core#0 to CPU core#7).

The command syntax is:

dsp_demo console [chip#] [core#]

The following example displays the debug message of demo_evm6678l.hex (DSP demo program) which is executed by CPU#0 in DSP#0.

```
\>dsp_demo console 0 0
Synchronizing ...
 done.
=========================
PCIe Hello World Example, this is DSP0
Debug: GEM-INTC Configuration Completed
Debug: CPINTC-0 Configuration...
Debug: CPINTC-0 Configuration Completed
```

Note: DSP program demo_evm6678l.hex should be downloaded to DSP device first before issuing this virtual console command. Refer to the source code of DSP demo program to get detailed implementation.

5.4.3. DSP Demo Program

DSP demo program configures DSP Legacy IntB and MSI0 to receive PCIE interrupt from PC host. The procedure of demo program is illustrated below with flow chart displayed in Figure 5-6:

1. Set up INTC for ISR handler for PCIe Legacy IntB and MSI0, then wait the interrupt sent from PC host
2. PC host writes test data pattern whose length is 640-byte to DSP DDR and sends an interrupt to DSP after finishing the writing of the test data pattern
3. The test data pattern in DDR will be added by 1 when DSP receives the interrupt from PC host. After finishing the operation, DSP will send an interrupt back to PC host.
4. PC Host receives the interrupt from DSP as the indication that the test data pattern has already been changed and prints the test data pattern
5. Repeat the communication 1000 times.

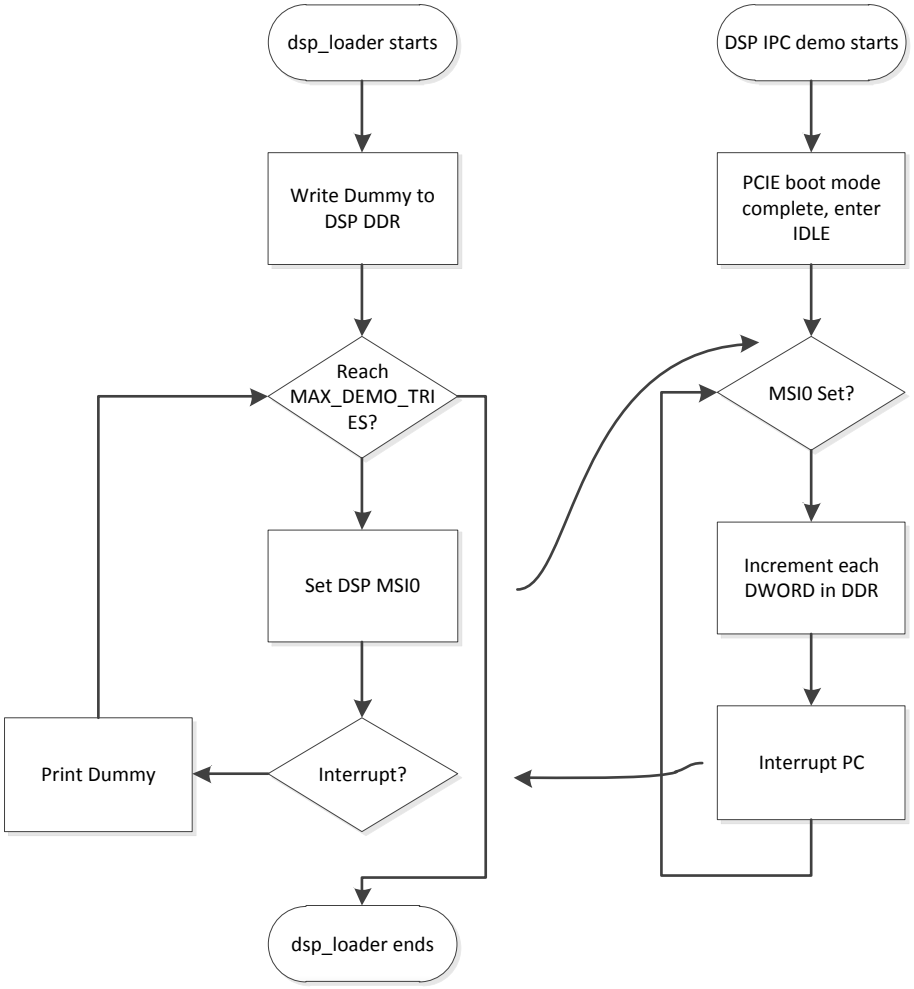

Figure 5-6 Flow Diagram of IPC Example

Besides the interrupt demo, the demo code also contains the virtual console implementation and the debug message will be written into L2 memory. PC host can use dsp_demo console command to dump these messages for debug purpose.

5.4.4. IPC Demo on SYS/BIOS

The IPC feature also works in TI's SYS/BIOS architecture. The demo program is embedded in the Ethernet example to show how to register an interrupt in SYS/BIOS architecture. In order to run this IPC demo, users can follow two steps below.

1. Initialize DDR3 module (init_1000.bat).
2. Run SYS/BIOS IPC demo script (ipc-SYSBIOS.bat)

```
Enter DSP ID [0~3]: 0
0
```

```
Translating to Intel format...

"C:\Users\sd\Desktop\Lightning_PCIE\examples\script\DSPC8681E\..\..\web\clie
nt\evmc6678l\DSPC8681E\client_evmc6678l.out"   ==> .text:_c_int00

"C:\Users\sd\Desktop\Lightning_PCIE\examples\script\DSPC8681E\..\..\web\clie
nt\evmc6678l\DSPC8681E\client_evmc6678l.out"   ==> .text

"C:\Users\sd\Desktop\Lightning_PCIE\examples\script\DSPC8681E\..\..\web\clie
nt\evmc6678l\DSPC8681E\client_evmc6678l.out"   ==> .const

"C:\Users\sd\Desktop\Lightning_PCIE\examples\script\DSPC8681E\..\..\web\clie
nt\evmc6678l\DSPC8681E\client_evmc6678l.out"   ==> .switch.1

"C:\Users\sd\Desktop\Lightning_PCIE\examples\script\DSPC8681E\..\..\web\clie
nt\evmc6678l\DSPC8681E\client_evmc6678l.out"   ==> .vecs

"C:\Users\sd\Desktop\Lightning_PCIE\examples\script\DSPC8681E\..\..\web\clie
nt\evmc6678l\DSPC8681E\client_evmc6678l.out"   ==> .switch.2

"C:\Users\sd\Desktop\Lightning_PCIE\examples\script\DSPC8681E\..\..\web\clie
nt\evmc6678l\DSPC8681E\client_evmc6678l.out"   ==> .cinit
Load HEX image:
C:\Users\sd\Desktop\Lightning_PCIE\examples\script\DSPC8681E\..\..\..\bin\DS
PC8681E\client_evmc6678l.hex to 0:0, start address 0x80300000
Load HEX OK
DDR of DSP is initialized, ready to write dummy data to DSP
dump dummy_buffer before DSP operation:

000000013F405020       00000000 00000001 00000002 00000003 00000004 00000005
00000006 00000007
000000013F405040       00000008 00000009 0000000a 0000000b 0000000c 0000000d
0000000e 0000000f
000000013F405060       00000010 00000011 00000012 00000013 00000014 00000015
00000016 00000017
000000013F405080       00000018 00000019 0000001a 0000001b 0000001c 0000001d
0000001e 0000001f
000000013F4050A0       00000020 00000021 00000022 00000023 00000024 00000025
00000026 00000027
000000013F4050C0       00000028 00000029 0000002a 0000002b 0000002c 0000002d
0000002e 0000002f
000000013F4050E0       00000030 00000031 00000032 00000033 00000034 00000035
00000036 00000037
000000013F405100       00000038 00000039 0000003a 0000003b 0000003c 0000003d
0000003e 0000003f
000000013F405120       00000040 00000041 00000042 00000043 00000044 00000045
00000046 00000047
000000013F405140       00000048 00000049 0000004a 0000004b 0000004c 0000004d
0000004e 0000004f
000000013F405160       00000050 00000051 00000052 00000053 00000054 00000055
00000056 00000057
000000013F405180       00000058 00000059 0000005a 0000005b 0000005c 0000005d
0000005e 0000005f
```

```
000000013F4051A0        00000060 00000061 00000062 00000063 00000064 00000065
00000066 00000067
000000013F4051C0        00000068 00000069 0000006a 0000006b 0000006c 0000006d
0000006e 0000006f
000000013F4051E0        00000070 00000071 00000072 00000073 00000074 00000075
00000076 00000077
000000013F405200        00000078 00000079 0000007a 0000007b 0000007c 0000007d
0000007e 0000007f
000000013F405220        00000080 00000081 00000082 00000083 00000084 00000085
00000086 00000087
000000013F405240        00000088 00000089 0000008a 0000008b 0000008c 0000008d
0000008e 0000008f
000000013F405260        00000090 00000091 00000092 00000093 00000094 00000095
00000096 00000097
000000013F405280        00000098 00000099 0000009a 0000009b 0000009c 0000009d
0000009e 0000009f
dummy data has already been changed by DSP
dump dummy_buffer after DSP operation:

000000013F405020        00000001 00000002 00000003 00000004 00000005 00000006
00000007 00000008
000000013F405040        00000009 0000000a 0000000b 0000000c 0000000d 0000000e
0000000f 00000010
000000013F405060        00000011 00000012 00000013 00000014 00000015 00000016
00000017 00000018
000000013F405080        00000019 0000001a 0000001b 0000001c 0000001d 0000001e
0000001f 00000020
000000013F4050A0        00000021 00000022 00000023 00000024 00000025 00000026
00000027 00000028
000000013F4050C0        00000029 0000002a 0000002b 0000002c 0000002d 0000002e
0000002f 00000030
000000013F4050E0        00000031 00000032 00000033 00000034 00000035 00000036
00000037 00000038
000000013F405100        00000039 0000003a 0000003b 0000003c 0000003d 0000003e
0000003f 00000040
000000013F405120        00000041 00000042 00000043 00000044 00000045 00000046
00000047 00000048
000000013F405140        00000049 0000004a 0000004b 0000004c 0000004d 0000004e
0000004f 00000050
000000013F405160        00000051 00000052 00000053 00000054 00000055 00000056
00000057 00000058
000000013F405180        00000059 0000005a 0000005b 0000005c 0000005d 0000005e
0000005f 00000060
000000013F4051A0        00000061 00000062 00000063 00000064 00000065 00000066
00000067 00000068
000000013F4051C0        00000069 0000006a 0000006b 0000006c 0000006d 0000006e
0000006f 00000070
000000013F4051E0        00000071 00000072 00000073 00000074 00000075 00000076
00000077 00000078
000000013F405200        00000079 0000007a 0000007b 0000007c 0000007d 0000007e
0000007f 00000080
000000013F405220        00000081 00000082 00000083 00000084 00000085 00000086
00000087 00000088
```

```
000000013F405240      00000089 0000008a 0000008b 0000008c 0000008d 0000008e
0000008f 00000090
000000013F405260      00000091 00000092 00000093 00000094 00000095 00000096
00000097 00000098
000000013F405280      00000099 0000009a 0000009b 0000009c 0000009d 0000009e
0000009f 000000a0
dummy data has already been changed by DSP
dump dummy_buffer after DSP operation:

000000013F405020      00000002 00000003 00000004 00000005 00000006 00000007
00000008 00000009
000000013F405040      0000000a 0000000b 0000000c 0000000d 0000000e 0000000f
00000010 00000011
000000013F405060      00000012 00000013 00000014 00000015 00000016 00000017
00000018 00000019
000000013F405080      0000001a 0000001b 0000001c 0000001d 0000001e 0000001f
00000020 00000021
000000013F4050A0      00000022 00000023 00000024 00000025 00000026 00000027
00000028 00000029
000000013F4050C0      0000002a 0000002b 0000002c 0000002d 0000002e 0000002f
00000030 00000031
000000013F4050E0      00000032 00000033 00000034 00000035 00000036 00000037
00000038 00000039
000000013F405100      0000003a 0000003b 0000003c 0000003d 0000003e 0000003f
00000040 00000041
000000013F405120      00000042 00000043 00000044 00000045 00000046 00000047
00000048 00000049
000000013F405140      0000004a 0000004b 0000004c 0000004d 0000004e 0000004f
00000050 00000051
000000013F405160      00000052 00000053 00000054 00000055 00000056 00000057
00000058 00000059
000000013F405180      0000005a 0000005b 0000005c 0000005d 0000005e 0000005f
00000060 00000061
000000013F4051A0      00000062 00000063 00000064 00000065 00000066 00000067
00000068 00000069
000000013F4051C0      0000006a 0000006b 0000006c 0000006d 0000006e 0000006f
00000070 00000071
000000013F4051E0      00000072 00000073 00000074 00000075 00000076 00000077
00000078 00000079
000000013F405200      0000007a 0000007b 0000007c 0000007d 0000007e 0000007f
00000080 00000081
000000013F405220      00000082 00000083 00000084 00000085 00000086 00000087
00000088 00000089
000000013F405240      0000008a 0000008b 0000008c 0000008d 0000008e 0000008f
00000090 00000091
000000013F405260      00000092 00000093 00000094 00000095 00000096 00000097
00000098 00000099
000000013F405280      0000009a 0000009b 0000009c 0000009d 0000009e 0000009f
000000a0 000000a1
dummy data has already been changed by DSP
dump dummy_buffer after DSP operation:

000000013F405020      00000003 00000004 00000005 00000006 00000007 00000008
```

```
00000009 0000000a
000000013F405040       0000000b 0000000c 0000000d 0000000e 0000000f 00000010
00000011 00000012
000000013F405060       00000013 00000014 00000015 00000016 00000017 00000018
00000019 0000001a
000000013F405080       0000001b 0000001c 0000001d 0000001e 0000001f 00000020
00000021 00000022
000000013F4050A0       00000023 00000024 00000025 00000026 00000027 00000028
00000029 0000002a
000000013F4050C0       0000002b 0000002c 0000002d 0000002e 0000002f 00000030
00000031 00000032
000000013F4050E0       00000033 00000034 00000035 00000036 00000037 00000038
00000039 0000003a
000000013F405100       0000003b 0000003c 0000003d 0000003e 0000003f 00000040
00000041 00000042
000000013F405120       00000043 00000044 00000045 00000046 00000047 00000048
00000049 0000004a
000000013F405140       0000004b 0000004c 0000004d 0000004e 0000004f 00000050
00000051 00000052
000000013F405160       00000053 00000054 00000055 00000056 00000057 00000058
00000059 0000005a
000000013F405180       0000005b 0000005c 0000005d 0000005e 0000005f 00000060
00000061 00000062
000000013F4051A0       00000063 00000064 00000065 00000066 00000067 00000068
00000069 0000006a
000000013F4051C0       0000006b 0000006c 0000006d 0000006e 0000006f 00000070
00000071 00000072
000000013F4051E0       00000073 00000074 00000075 00000076 00000077 00000078
00000079 0000007a
000000013F405200       0000007b 0000007c 0000007d 0000007e 0000007f 00000080
00000081 00000082
000000013F405220       00000083 00000084 00000085 00000086 00000087 00000088
00000089 0000008a
000000013F405240       0000008b 0000008c 0000008d 0000008e 0000008f 00000090
00000091 00000092
000000013F405260       00000093 00000094 00000095 00000096 00000097 00000098
00000099 0000009a
000000013F405280       0000009b 0000009c 0000009d 0000009e 0000009f 000000a0
000000a1 000000a2
.
.
.
```

5.5. Image Processing Demonstration

The image processing program is modified from the example codes in TI MCSDK. This application shows implementation of an image processing system using a simple multicore framework. This application will run TI image processing kernels (imagelib) on multiple cores to do image processing (eg: edge detection, etc) on an input image.

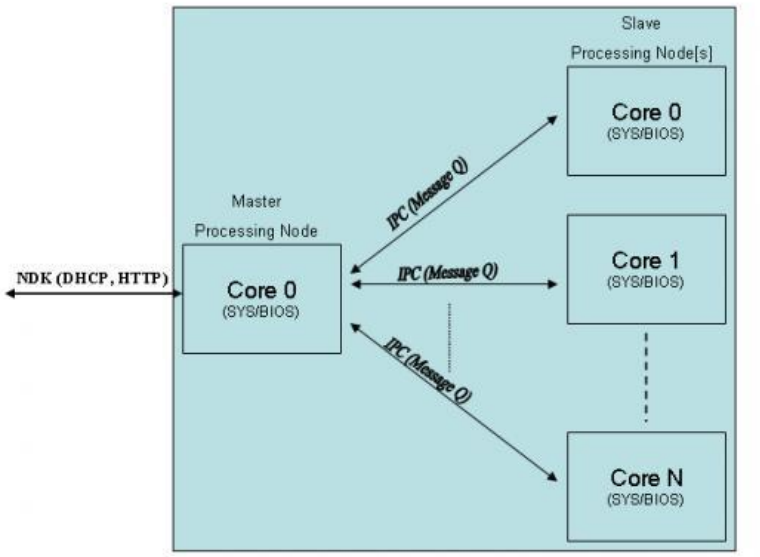

Figure 5-7 Image Processing Application Software Framework

The user input image will be BMP image. The image will be transferred to external memory using NDK (http). The Ethernet port on the bracket of the Lightning board must be connected to an external Ethernet switch (support gigabit rates) before running this example. Each DSP has a fixed IP number that is determined by its order. The pre-given IP addresses are shown below. The user can use a browser to input the BMP image form web page provided by HTTP server.

	IP
DSP 0	192.168.1.101
DSP 1	192.168.1.102
DSP 2	192.168.1.103
DSP 3	192.168.1.104

PC Setting	
IP	192.168.1.100
Subnet Mask	255.255.254.0

5.5.1. Build Instruction

Steps to build image processing program are listed below:

1. Import CCS project
   1.1 Click Project->Import Existing CCS/ CCE Eclipse Project.
   1.2 Select Lightning_PCIE\examples\image_processing\ipc\evmc6678l directory.
2. Select DSPC8681E as active configuration.
3. Clean and rebuild
   3.1 Clean and rebuild the image_processing_evmc6678l_master project.
   3.2 After build is completed, image_processing_evmc6678l_master.out will be created in Lightning_PCIE\examples\image_processing\ipc\evmc6678l\master\DSPC8681E.

5.5.2. Load Image Processing Example

Lightning_PCIE\examples\script\DSPC8681E\image_processing.bat can help users load the DSP binary to setup image processing program on each DSP automatically. To load this example, there are three steps:

1. Initialize DDR3 module (init_1000.bat).
2. Run Image Processing loader script (image_processing.bat).
   2.1 Convert .out to .hex
      2.1.1 image_processing_evmc6678l_slave.out
      2.1.2 image_processing_evmc6678l_master.out
   2.2 Load the image to each DSP.

```
Translating to Intel format...

"C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\image_processing\ipc\e
vmc6678l\slave\Debug\image_processing_evmc6678l_slave.out"
==> .text:_c_int00

"C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\image_processing\ipc\e
vmc6678l\slave\Debug\image_processing_evmc6678l_slave.out"   ==> .text

"C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\image_processing\ipc\e
vmc6678l\slave\Debug\image_processing_evmc6678l_slave.out"   ==> .const

"C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\image_processing\ipc\e
vmc6678l\slave\Debug\image_processing_evmc6678l_slave.out"   ==> .switch

"C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\image_processing\ipc\e
vmc6678l\slave\Debug\image_processing_evmc6678l_slave.out"   ==> .vecs

"C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\image_processing\ipc\e
```

```
vmc6678l\slave\Debug\image_processing_evmc6678l_slave.out"  ==> .cinit
Translating to Intel format...

"C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\image_processing\ipc\e
vmc6678l\master\DSPC8681E\image_processing_evmc6678l_master.out"
==> .text:_c_int00

"C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\image_processing\ipc\e
vmc6678l\master\DSPC8681E\image_processing_evmc6678l_master.out"
==> .text

"C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\image_processing\ipc\e
vmc6678l\master\DSPC8681E\image_processing_evmc6678l_master.out"
==> .const.1

"C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\image_processing\ipc\e
vmc6678l\master\DSPC8681E\image_processing_evmc6678l_master.out"
==> .const.2

"C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\image_processing\ipc\e
vmc6678l\master\DSPC8681E\image_processing_evmc6678l_master.out"
==> .switch.1

"C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\image_processing\ipc\e
vmc6678l\master\DSPC8681E\image_processing_evmc6678l_master.out"
==> .vecs

"C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\image_processing\ipc\e
vmc6678l\master\DSPC8681E\image_processing_evmc6678l_master.out"
==> .switch.2

"C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\image_processing\ipc\e
vmc6678l\master\DSPC8681E\image_processing_evmc6678l_master.out"
==> .cinit
Load HEX image:
C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\..\bin\DSPC8681E\image_
processing_evmc6678l_slave.hex to 0:1, start address 0x0c100000
Load HEX OK
Load HEX image:
C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\..\bin\DSPC8681E\image_
processing_evmc6678l_slave.hex to 0:2, start address 0x0c100000
Load HEX OK
Load HEX image:
C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\..\bin\DSPC8681E\image_
processing_evmc6678l_slave.hex to 0:3, start address 0x0c100000
Load HEX OK
Load HEX image:
C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\..\bin\DSPC8681E\image_
processing_evmc6678l_slave.hex to 0:4, start address 0x0c100000
Load HEX OK
Load HEX image:
C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\..\bin\DSPC8681E\image_
processing_evmc6678l_slave.hex to 0:5, start address 0x0c100000
```

```
Load HEX OK
Load HEX image:
C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\..\bin\DSPC8681E\image_
processing_evmc6678l_slave.hex to 0:6, start address 0x0c100000
Load HEX OK
Load HEX image:
C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\..\bin\DSPC8681E\image_
processing_evmc6678l_slave.hex to 0:7, start address 0x0c100000
Load HEX OK
Load HEX image:
C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\..\bin\DSPC8681E\image_
processing_evmc6678l_slave.hex to 1:1, start address 0x0c100000
Load HEX OK
Load HEX image:
C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\..\bin\DSPC8681E\image_
processing_evmc6678l_slave.hex to 1:2, start address 0x0c100000
Load HEX OK
Load HEX image:
C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\..\bin\DSPC8681E\image_
processing_evmc6678l_slave.hex to 1:3, start address 0x0c100000
Load HEX OK
Load HEX image:
C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\..\bin\DSPC8681E\image_
processing_evmc6678l_slave.hex to 1:4, start address 0x0c100000
Load HEX OK
Load HEX image:
C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\..\bin\DSPC8681E\image_
processing_evmc6678l_slave.hex to 1:5, start address 0x0c100000
Load HEX OK
Load HEX image:
C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\..\bin\DSPC8681E\image_
processing_evmc6678l_slave.hex to 1:6, start address 0x0c100000
Load HEX OK
Load HEX image:
C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\..\bin\DSPC8681E\image_
processing_evmc6678l_slave.hex to 1:7, start address 0x0c100000
Load HEX OK
Load HEX image:
C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\..\bin\DSPC8681E\image_
processing_evmc6678l_slave.hex to 2:1, start address 0x0c100000
Load HEX OK
Load HEX image:
C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\..\bin\DSPC8681E\image_
processing_evmc6678l_slave.hex to 2:2, start address 0x0c100000
Load HEX OK
Load HEX image:
C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\..\bin\DSPC8681E\image_
processing_evmc6678l_slave.hex to 2:3, start address 0x0c100000
Load HEX OK
Load HEX image:
C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\..\bin\DSPC8681E\image_
processing_evmc6678l_slave.hex to 2:4, start address 0x0c100000
Load HEX OK
```

```
Load HEX image:
C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\..\bin\DSPC8681E\image_
processing_evmc6678l_slave.hex to 2:5, start address 0x0c100000
Load HEX OK
Load HEX image:
C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\..\bin\DSPC8681E\image_
processing_evmc6678l_slave.hex to 2:6, start address 0x0c100000
Load HEX OK
Load HEX image:
C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\..\bin\DSPC8681E\image_
processing_evmc6678l_slave.hex to 2:7, start address 0x0c100000
Load HEX OK
Load HEX image:
C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\..\bin\DSPC8681E\image_
processing_evmc6678l_slave.hex to 3:1, start address 0x0c100000
Load HEX OK
Load HEX image:
C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\..\bin\DSPC8681E\image_
processing_evmc6678l_slave.hex to 3:2, start address 0x0c100000
Load HEX OK
Load HEX image:
C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\..\bin\DSPC8681E\image_
processing_evmc6678l_slave.hex to 3:3, start address 0x0c100000
Load HEX OK
Load HEX image:
C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\..\bin\DSPC8681E\image_
processing_evmc6678l_slave.hex to 3:4, start address 0x0c100000
Load HEX OK
Load HEX image:
C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\..\bin\DSPC8681E\image_
processing_evmc6678l_slave.hex to 3:5, start address 0x0c100000
Load HEX OK
Load HEX image:
C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\..\bin\DSPC8681E\image_
processing_evmc6678l_slave.hex to 3:6, start address 0x0c100000
Load HEX OK
Load HEX image:
C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\..\bin\DSPC8681E\image_
processing_evmc6678l_slave.hex to 3:7, start address 0x0c100000
Load HEX OK
Load HEX image:
C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\..\bin\DSPC8681E\image_
processing_evmc6678l_master.hex to 0:0, start address 0x0c000000
Load HEX OK
Load HEX image:
C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\..\bin\DSPC8681E\image_
processing_evmc6678l_master.hex to 1:0, start address 0x0c000000
Load HEX OK
Load HEX image:
C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\..\bin\DSPC8681E\image_
processing_evmc6678l_master.hex to 2:0, start address 0x0c000000
Load HEX OK
Load HEX image:
```

```
C:\Lightning_PCIE\examples\script\DSPC8681E\..\..\..\bin\DSPC8681E\image_
processing_evmc6678l_master.hex to 3:0, start address 0x0c000000
Load HEX OK
```

3. Please refer to the Figure 5-8. Input the BMP image form the internet browser. The URL of DSPs are http://192.168.1.10X, X=1~4. Select the number of core and image path for processing.
4. The output result is shown in Figure 5-9.

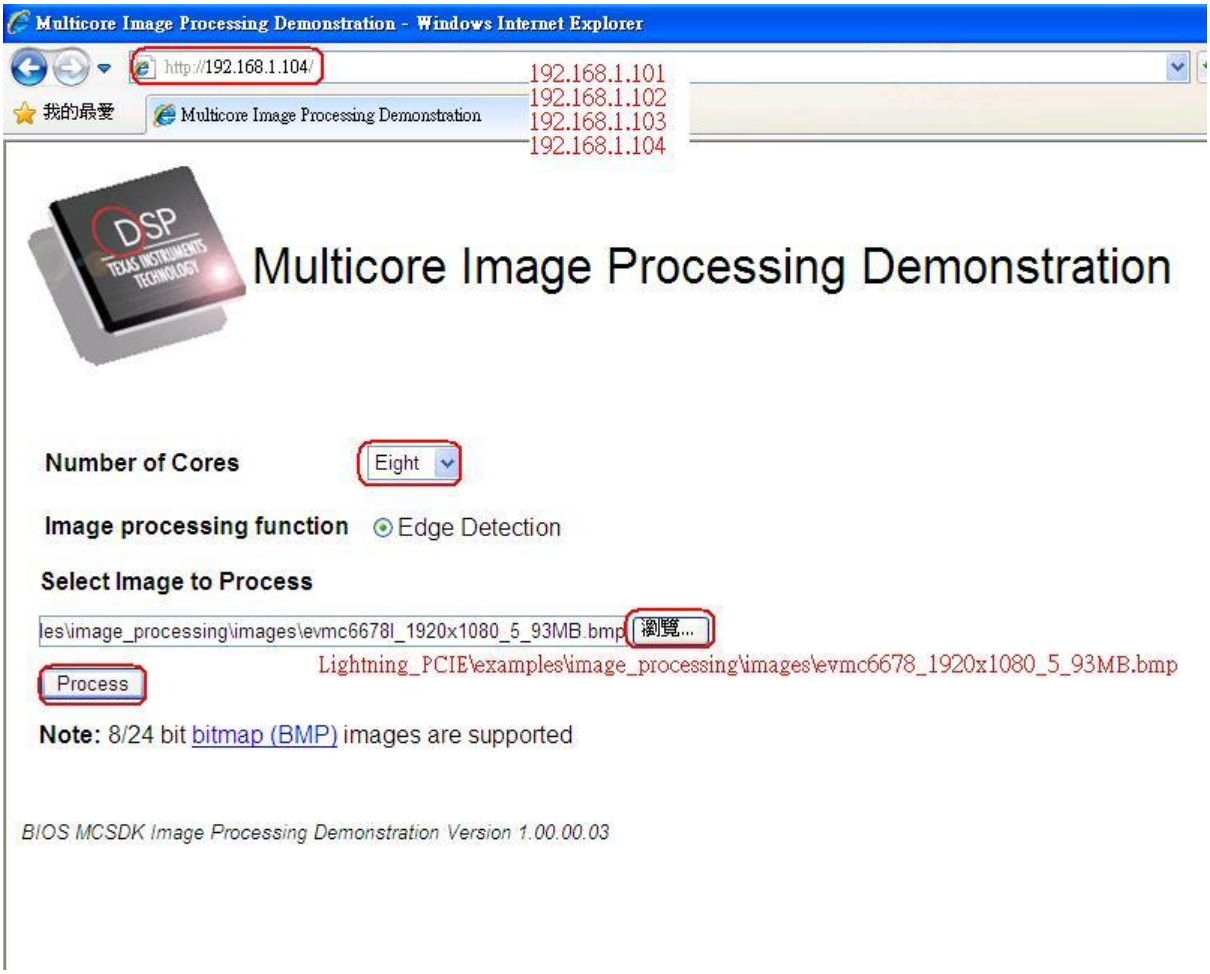

Figure 5-8 Image Processing Input Page

Figure 5-9 Image Processing Output Page