# Keystone Multicore Workshop Demo manual

# Demo 7 MCSDK HUA Demonstration Guide

Wiki URL is <http://processors.wiki.ti.com/index.php/MCSDK_HUA_Demonstration_Guide>

# Overview

The High-Performance DSP Utility Application (HUA) is the Out-of-Box (OOB) demonstration for the Multicore Software Development Kit (MCSDK) which demonstrates, through illustrative code and web pages, how you can interface your own DSP application to the various TI MCSDK software elements including SYS/BIOS, Network Development Kit (NDK), the Chip Support Library (CSL), and Platform Library. The purpose of the demonstration is to illustrates the integration of key components in MCSDK and provide a multicore software development framework on an evaluation module (EVM.)

This document covers various aspects of the demonstration, including a discussion on the requirements, software design, instructions to build and run the application, and troubleshooting steps. Currently, only SYS/BIOS is supported as the embedded OS.

Access to the demo application is done through a PC web browser. The welcome web page provides a starting point with links to more information on TI multicore DSPs and support forums.

In addition at the top of the web page are a number of tabs which implement basic functionality including:

* **Information**: Generates a page displaying a collection of information related to the platform and its operation such as system up time, platform settings, device type, number of cores, core speeds, software element versions, and network stack information. All this information is collected using API calls to the various MCSDK software elements.
* **Statistics**: Generates a page reporting standard Ethernet statistics from the networking stack.
* **Task List**: generates a page reporting the current active SYS/BIOS tasks on the device including information such as Task Priority, Task State, Stack Size Allocated, and Stack Size Used for each task.
* **Benchmarks**: Takes the user to a web page with a list of supported benchmarks that a the user can run on the platform.
* **Diagnostics**: Takes the user to a web page that allows the user to execute a range of platform diagnostics tests.
* **Flash**: Takes the user to a web page that display flash hardware information and allows the user to read and write the flash on the platform.
* **EEPROM**: Takes the user to a web page that allows the user to read the EEPROM.

### Flash

The *Flash* page displays information related to the Flash hardware and allows the user to read and write to the flash. For reading, the user can specify a block to read from flash and then page through the data. For writing the user can either write an arbitrary file (binary blob) or a bootable image. The bootable image option allows you to write an image the EEPROM boot loader can load and execute.

### EEPROM

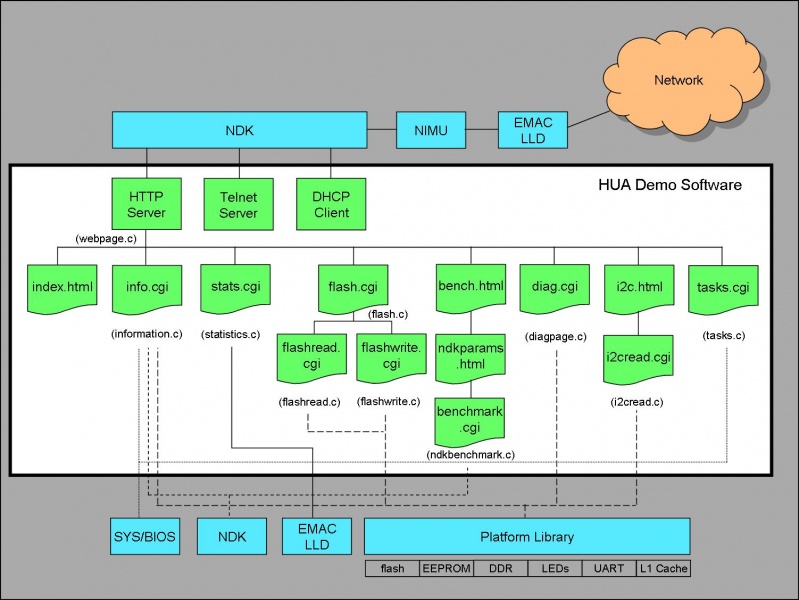
The *EEPROM* page allows a user to read the EEPROM paging through the data in 1K blocks.

# Requirements

The following materials are required to run this demonstration:

* TMS320C6x low cost EVMs [Check MCSDK release notes for supported platforms]
* Power cable
* Ethernet cable
* Windows PC with CCSv5

# Software Design

[](http://processors.wiki.ti.com/index.php/File:HuaArchitecture.jpg)

[http://processors.wiki.ti.com/skins/common/images/magnify-clip.png](http://processors.wiki.ti.com/index.php/File:HuaArchitecture.jpg)

The high level software architecture for the HUA is shown below.

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 EVM (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**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 items that are platform dependent.

# Build Instructions

Please follow the steps below to re-compile the libraries (These steps assume you have installed the MCSDK and all the dependent packages).

* Open CCS->Import Existing... tab and import project from C:\Program Files\Texas Instruments\mcsdk\_2\_00\_00\_xx\demos\hua.
* It should import two projects hua\_evmc6678l and hua\_evmc6670l.
* Right click on each project->Properties to open up the properties window.
* Goto CCS Build->RTSC and check if in other repository have link to <MCSDK INSTALL DIR> (the actual directory).
* The project should build fine.

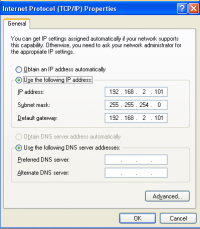
# Run Instructions

The pre-compiled libraries are provided as a part of MCSDK release.

Please follow the procedures below to load images using CCS and run the demo.

Please refer to the hardware setup guide for further setup details.

* Connect the board to a Ethernet hub or PC using Ethernet cable.
* The demo runs in Static IP mode if User Switch 1 is OFF else if it is ON then it runs in DHCP mode. See the Hardware Setup section for the location of User Switch 1.
* If it is configured in static IP mode, the board will come up with IP address 192.168.2.100, GW IP address 192.168.2.101 and subnet mask 255.255.254.0
* If it is configures in DHCP mode, it would send out DHCP request to get the IP address from a DHCP server in the network.
* Connect the debugger and power on the board.
* In CCS window, launch the target configuration file for the board.
* It should open debug perspective and open debug window with all the cores.
* Connect to core 0 and load demos\hua\evmc66xxl\Debug\hua\_evmc66xxl.out.
* Run HUA on core 0, in the CIO console window, the board should print IP address information (for eg: Network Added: If-1:192.168.2.100)
* Open a web browser in the PC connected to the HUB or the board.
* Enter the IP address of the board, it should open up the HUA demo web page.
* Please follow the instructions in the web page to run the demo.

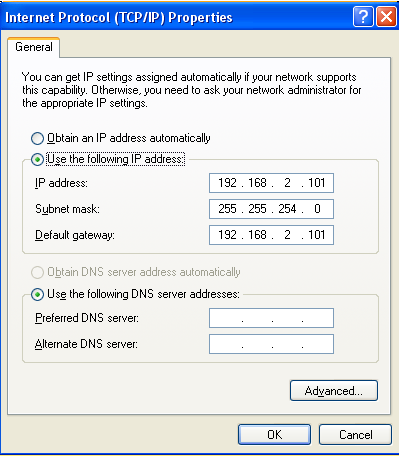
[](http://processors.wiki.ti.com/index.php/File:Wirednwconnection.png)

[http://processors.wiki.ti.com/skins/common/images/magnify-clip.png](http://processors.wiki.ti.com/index.php/File:Wirednwconnection.png)

Note**Note:** If you want to run the demo in static IP address mode, make sure the host PC is in same subnet or can reach the gateway. A sample setup configuration is shown below.

In Windows environment

Set up TCP/IP configuration of ‘Wired Network Connection’ as shown in the following:

[](http://processors.wiki.ti.com/images/1/1f/Wirednwconnection.png)

In Linux environment

Run following command to set the static IP address for the current login session on a typical Linux setup.

**sudo** **ifconfig** eth0 192.168.2.101 netmask 255.255.254.0

# Troubleshooting

## Data verification error when using CCS to load HUA

Check if the EVM GEL is properly configured and run when CCS connects the target. The GEL will initialize the PLL and external memory so that HUA can be loaded and run from external memory.

## The CIO console window does not show the IP address

Check if the EVM is connected to a network with DHCP server running.

## The CIO console window shows the static IP address, but can not ping it

Check if the EVM is connected to a static network, and the PC that is used to ping the EVM has the same subnet address as the EVM does.

|  |  |  |
| --- | --- | --- |
| Very Large FFT Multicore DSP Implementation Demonstration Guide | 1 |  |
|  |  |

**Demo 8 - Very Large FFT Multicore DSP Implementation**

**Demonstration Guide**

**Overview**

This demo software implements single precision floating point very large size FFT on Texas Instruments' latest multicore DSPs including C6678 and C6670. The software requires input data to be placed in the device's external memory. It distributes input data onto different DSP cores. Different DSP cores carry out the actual computations and place the output on the external memory. The software can be configured to use different number of cores to do the actual computation and can computer the FFT of the following sizes

• 16K

• 32K

• 64K

• 128K

• 256K

• 512K

• 1024K

The software can be run on the following EVM and simulators,

* C6678 EVM
* C6678 Functional Simulator
* C6670 EVM
* C6670 Functional Simulator

**Requirements**

The software requires Texas Instruments latest multicore SDK 2.0 (MCSDK 2.0). Particularly it requires the following software components from MCSDK 2.0.

* CCS 5
* DSP/BIOS 6.0
* IPC
* EDMA LLD

**Software Design**

The very large FFT implementation for multicore DSP is designed to achieve maximum performance by distributing the computation task onto multicores and by fully utilizing high performance computational power of DSP.

The basic decimation-in-time approach is used to formulate computing 1-D very large FFT into computing something similar to 2-D FFT computation. For very large N, it can be factored into N = N1\*N2. If its very large 1-D input array is considered as a 2-D N1xN2 array (N1 rows, N2 columns), then the following steps can be taken to compute 1-D very large FFT.

1. Compute N2 FFTs of N1 size in column directions
2. Multiply twiddle factor
3. Store N2 FFTs of N1 size in row directions to form a N2xN1 2-D array
4. Compute N1 FFTs of N2 size in column direction
5. Store data in column direction to form a N2xN1 2-D array

|  |  |  |
| --- | --- | --- |
| Very Large FFT Multicore DSP Implementation Demonstration Guide | 2 |  |
|  |  |

The following paper describes the similar algorithm for multicore FFT implementation.

"High-performance Parallel FFT algorithms for the Hitachi SR8000", Daisuke Takahashi,

Proceedings of The Fourth International Conference/Exhibition on High Performance Computing in the Asia-Pacific Region, 2000, Issue Date: 14-17 May 2000, On page(s): 192 - 199 vol.1

In the actual computation, **N2/NUM\_Of\_CORES\_FOR\_COMPUTE** FFTs of size N1 in step 1. and **N1/NUM\_of\_CORES\_FOR\_COMPUTE** FFTs of size N2 in step 4. are computed on each core. Core0 is used asmaster core and the rest of the cores are used as slave cores. IPC software is used for inter processor communications. In addition to the FFT computations listed above, the core0 (master core) is also responsible for synchronizing all the cores.

The sequence of the main processings for the software thread on the master core (core0) and all the slave cores for computing an entire large size FFT is summarized as follows,

**Software thread on core0**

* FFT computation starts
* Core0 sends a command to all the slave cores informing each core to be in IDLE state
* Core0 waits for all the slave cores in IDLE state
* Core0 sends a command to all the slave cores informing each to start 1st iteration of processing
* Core0 starts its 1st iteration of processing
  1. Core0 fetches **N2/NUM\_Of\_CORES\_FOR\_COMPUTE** columns of its assigned data into L2 SRAM
  2. core0 computes **N2/NUM\_Of\_CORES\_FOR\_COMPUTE** FFTs of N1 size
  3. Multiply twiddle factors of each output
  4. Core0 stores **N2/NUM\_Of\_CORES\_FOR\_COMPUTE** FFTs of N1 size in row direction into an N2xN1 array in external memory (DDR3)
* Core0 waits for all the cores to complete their 1st iteration processing
* Core0 sends a command to all the slave cores to start 2nd iteration of processing
* Core0 starts its 2nd iteration of processing
  1. Core0 fetches **N1/NUM\_Of\_CORES\_FOR\_COMPUTE** columns of its assigned data into L2 SRAM
  2. core0 computes **N1/NUM\_Of\_CORES\_FOR\_COMPUTE** FFTs of N2 size
  3. Core0 stores **N1/NUM\_Of\_CORES\_FOR\_COMPUTE** FFTs of N1 size in row direction into an N2xN1 array in external memory (DDR3)
* Core0 waits for all the cores to complete their 2nd iteration processing
* FFT computation ends

**Software thread on slave cores**

* Each slave core waits for the command from master core (core0)
* Each slave core starts 1st iteration processing when receiving command from Core0 for starting 1st iteration processing
  1. Each slave core fetches **N2/NUM\_Of\_CORES\_FOR\_COMPUTE** columns of its assigned data into L2 SRAM
  2. Each core compute **N2/NUM\_Of\_CORES\_FOR\_COMPUTE** FFTs of N1 size
  3. Multiply twiddle factors of each output
  4. Each slave core stores **N2/NUM\_Of\_CORES\_FOR\_COMPUTE** FFTs of N1 size in row direction into an N2xN1 array in external memory (DDR3)
* Each slave core sends a message to core0 informing the completion of 1st iteration processing

|  |  |  |
| --- | --- | --- |
| Very Large FFT Multicore DSP Implementation Demonstration Guide | 3 |  |
|  |  |

* Each slave core waits for the command from master core (core0)
* Each slave core starts 2nd iteration processing when receiving command from Core0 for starting 2nd iteration processing
  1. Each slave core fetches **N1/NUM\_Of\_CORES\_FOR\_COMPUTE** columns of its assigned data into L2 SRAM
  2. Each slave core computes **N1/NUM\_Of\_CORES\_FOR\_COMPUTE** FFTs of N2 size
  3. Each slave core stores **N1/NUM\_Of\_CORES\_FOR\_COMPUTE** FFTs of N2 size in column direction into an N1xN2 array in external memory (DDR3)
* Each slave core sends a message to core0 informing the completion of 2nd iteration processing

On each core, depending on the sizes of N1 and N2, the total number of FFTs on each core,

**N1/NUM\_Of\_CORES\_FOR\_COMPUTE** and/or **N2/NUM\_Of\_CORES\_FOR\_COMPUTE**, are divided intoseveral smaller blocks in order to accomendate the limited available internal memory (L2 SRAM) on the device and each block size is 8 FFT.

In the actulal implementation, each block of data are prefetched by DMA from external memory into L2 SRAM and the FFT results are writen back to external memory by DDR. 16 DMA channels in total of EDMA instant0 are used for fethcing data. Two DMA channels are used by each core to transfer input and output samples between external memory (DDR3) and internal memory (L2 SRAM).

The following lists the memory utilization of the software for computing size N=N1\*N2 FFT

**External Memory (DDR3)**

* input buffer: 1 complex single precision floating point arrays of size N
* Output buffer: 1 complex single precision floating point arrays of size N
* Temorary buffer: 1 complex single precision floating point arrays of size N

**L2 SRAM**

* 2 complex single precision floating point arrays of 16K size each
* 1 complex single precision floating point arrays of 8K size each
* 2 complex single precision floating point arrays of 1K size each
* 2 complex single precision floating point arrays of N2 size each (twiddle factors)
* 1 complex single precision of floating point array of size N1 (twiddle factors)

**Build Instructions**

The very large FFT demo software comes with pre-created project for C6678 or C6670 EVM. The following lists the steps to compile and build the project,

* Install the project in a directory c:\xx\vlfft. In the configuration file add delete older depository and add a depository to the same directory where the install is (ends with vlfft directory) and the ccs\_base depository
* Make sure that you can see the platform custome.vlfft.evmc6678l.core0
* To compile for C6678 EVM, open the file **vlfftconfig.h** under **\demo\vlfft\vlfftInc** and set the constant

**EIGHT\_CORE\_DEVICE** to 1 and **FOUR\_CORE\_DEVICE** to 0.

* To compile for C6670 EVM, open the file **vlfftconfig.h** under **\demo\vlfft\vlfftInc** and set the constant

**EIGHT\_CORE\_DEVICE** to 0 and **FOUR\_CORE\_DEVICE** to 1.

* To configure the size of the FFT, open the file **vlfftconfig.h** under **\demo\vlfft\vlfftInc** and set one of the following constant definitions to 1 and the rest to zero

|  |  |  |
| --- | --- | --- |
| Very Large FFT Multicore DSP Implementation Demonstration Guide | 4 |  |
|  |  |

VLFFT\_16K

VLFFT\_32K

VLFFT\_64K

VLFFT\_128K

VLFFT\_256K

VLFFT\_512K

VLFFT\_1024K

* To configure the number of DSP cores to compute, open the file **vlfftconfig.h** under **...\demo\vlfft\vlfftInc** and change the constant definition **NUM\_CORES\_FOR\_FFT\_COMPUTE** to one of the following numbers
  + 4-core device: 1, 2, 4
  + 8-core device: 1, 2, 4, 8
* Set either Debug or Release active in ccs5
  + For Debug mode: the following 4 lines from line 92 - line 95 in file **vlfft\_evmc6678l.cfg** under

**..\demos\vlfft\evmc6678l** or **vlfft\_evmc6670l.cfg** under **..\demos\vlfft\evmc6670l** should be disabled orcommented out.

var MessageQ = xdc.module('ti.sdo.ipc.MessageQ');

var Notify = xdc.module('ti.sdo.ipc.Notify');

Notify.SetupProxy = xdc.module('ti.sdo.ipc.family.c647x.NotifyCircSetup');

MessageQ.SetupTransportProxy = xdc.module('ti.sdo.ipc.transports.TransportShmNotifySetup');

* For Release mode: the following 4 lines from line 92 - line 95 in file **vlfft\_evmc6678l.cfg** under

**..\demos\vlfft\evmc6678l** or **vlfft\_evmc6670l.cfg** under **..\demos\vlfft\evmc6670l** should be enabled.

var MessageQ = xdc.module('ti.sdo.ipc.MessageQ');

var Notify = xdc.module('ti.sdo.ipc.Notify');

Notify.SetupProxy = xdc.module('ti.sdo.ipc.family.c647x.NotifyCircSetup');

MessageQ.SetupTransportProxy = xdc.module('ti.sdo.ipc.transports.TransportShmNotifySetup');

• Build the project using the Build Project option under Build in ccs5

**Run Instructions**

* To run the code on C6678 functional simulator, load **vlfft\_evmc6678l.out** from either **\vlfft\evmc6678l\\Debug** or **\vlfft\evmc6678l\Release** directory onto all the cores on the device. This is true regardless the number of cores is configured to compute the FFT. Run the code on all the cores.
* To run the code on C6678 EVM, initialize the PLL and DDR3 of the EVM using right GEL files. load **vlfft\_evmc6678l.out** from either **\vlfft\evmc6678l\Debug** or **\vlfft\evmc6678l\Release** directory onto all thecores on the device. This is true regardless the number of cores is configured to compute the FFT. Run the code on all the cores.

|  |  |  |
| --- | --- | --- |
| Article Sources and Contributors | 5 |  |
|  |  |
| **Article Sources and Contributors** |  |  |
| **Very Large FFT Multicore DSP Implementation Demonstration Guide** *Source*: http://ap-fpdsp-swapps.dal.design.ti.com/index.php?oldid=103203 | *Contributors*: A0214579 |  |