



HUNT ENGINEERING
Chestnut Court, Burton Row,
Brent Knoll, Somerset, TA9 4BP, UK
Tel: (+44) (0)1278 760188,
Fax: (+44) (0)1278 760199,
Email: sales@hunteng.co.uk
<http://www.hunteng.co.uk>
<http://www.hunt-dsp.com>



The “remote” Server/Loader example.

Rev 1.0 JT 15/11/03 (new with SL 4.08)

This example will work with HEART boards that support Inter-Board Connectors, such as the HEPC9. It assumes 2 DSP processors on 2 different HEART boards that are connected via Inter-Board Connectors. One of the boards is “remote”, i.e. it will not be accessed via its PCI interface. One DSP send a short message to the other DSP, which verifies the message and prints on the screen whether it was successful or not. The network file shows how HEART connections can be made from node to node even if the connection will in reality go over an Inter-Board connection. The HERON-API is used to send the message between the processors and to the host program over the HERON FIFO.

The use of HERON-API means that the example is easily changed to use any HERON C6000 module. HERON-API uses DSP/BIOS internally so must be built using Code Composer Studio.

This document describes how to make the project and build the DSP application.

History

Example revision 1.0 made for Server/Loader V4.08

Example software

The example that we supply consists of two C files for the DSPs called module1.c and module2.c. They need to be built using Code Composer Studio and use the HERON-API software that has been installed on your PC when you did the “install drivers and tools” from your CD.

Hardware setup

This example will only work with HEART boards that support Inter-Board Connectors, such as the HEPC9. The example shows the communication between two HERON modules on different HEART boards via Inter-Board Connectors. One HERON module is connected to the host, and there is a connection between the two HERON modules.

The demo as shipped is for an HEPC9 with a HERON module in slot 1 of an HEPC9 with the board-switch set to 0, and a HERON module in slot 1 of an HEPC9 with the board-switch set to 1, connected by two EM2's using channel 0 on both EM2's. One module has a FIFO connection with the host so that module can write (printf) to the screen.

If you are running the demo on a different hardware configuration, you will need to change the #defines in the DSP source code to reflect the connections that you have, and also the network file that describes the connections to the Server/Loader.

Remote

The network file declares one board to be “remote”. This means that the Server/Loader will not access that board via its PCI interface. Instead, the Server/Loader will try to access to board via another board that is connected to it via Inter-Board Connectors. In this example, HEPC9 1 is remote, and the Server/Loader will try to access HEPC9 1 via HEPC9 0.

Using “remote” boards is useful if boards (one board or multiple boards) are embedded, in another PC, or simply outside of the current PC. However, you can also use “remote” even if the board is actually in the current PC. The “remote” keyword just specifies “don't use this board's PCI interface”, so it doesn't matter where the board is, as long as it has an Inter-Board Connection to it.

DSP/BIOS

DSP/BIOS is the multi-threading environment provided as part of the Code Composer development Environment. It also provided services for configuring processor features such as hardware interrupts and timers.

As it is included in Code Composer Studio, along with the Compile tools for the C6000, all users of HERON hardware will be able to use it.

This example is configured and built using Code Composer and DSP/BIOS.

HERON API

HERON_API is the hardware independence layer that we provide to access HERON FIFOs and other features of the HERON modules. It allows the DMA engines of the processor to be used when transferring to and from the FIFOS without knowledge of the FIFO hardware, or the DMA engines.

Starting

We assume that a user of this example has previously installed Code Composer and followed the confidence checks. They should also be familiar with using Code Composer.

Configuring the example

HUNT ENGINEERING provide several Code Composer Plug-in tools that allow you to make your development faster. The first is one that sets up Code Composer ready for your hardware, so you don't need to configure device drivers etc and can be found from Start→Programs→HUNT ENGINEERING→AutoConfigure CCS.

If both boards are in the same PC, you can configure Code Composer Studio to debug both boards at the same time. Open the Confidence Checks program (Start→Programs→HUNT ENGINEERING→Confidence checks). Verify that in GroupTools→Boards in Group both of your HEPC9's are selected. Then select Code Composer→AutoConfigure Group. This will now configure Code Composer Studio to debug all boards as selected in GroupTools→Boards in Group. From the Parallel Debug Manager you can open a debug window for any module on any board in the Group.

If the remote board is truly remote (i.e. not in the current PC) you cannot use AutoConfigure, you will need to configure Code Composer Studio manually. This is further explained in the "host api – windows (install and user)" manual.

When you start with the 'remote' example, simply copy the source files from the CD into a new directory. Then start Code Composer and you will see the Parallel Debug Manager appear. Start a debug window for the first HERON module (Open→HEP9A 0/CPU_1) and one for the second HERON module (Open→HEP9A 1/CPU_1). You need to create a new project for both processors.

Now create the project for the **first HERON** module. Choose "Tools→HUNT ENGINEERING→Create new Heron-API project". This will guide you through setting up the project. As long as you choose the name "module1" for the project it will incorporate the module1.c file. Create the project for the Server/Loader; the plug-in creates and includes a file called "module1_stub.c" to the project. This file 'links' the Server/Loader DSP library with the HERON-API library. The CDB files as delivered with the HERON-API installation have a task 'TSK0' with entry point '_maintask'. Therefore, the project is ready to be compiled.

The default .cdb file will actually place all code into external memory, and switch on the program cache. This is a good general purpose setting, but might need to be changed for your actual application.

You can now build the demo by choosing Project→Rebuild all. There should be no errors or warnings.

Create the project for the **second HERON** module. Choose "Tools→HUNT ENGINEERING→Create new Heron-API project". This will guide you through setting up the project. As long as you choose the name "module2" for the project it will incorporate the module2.c file. Create the project for the Server/Loader; the plug-in creates and includes a file called "module2_stub.c" to the project. This file 'links' the Server/Loader DSP library with the HERON-API library. The CDB files as delivered with the HERON-API installation have a task 'TSK0' with entry point '_maintask'. Therefore, the project is ready to be compiled.

You can now build the demo by choosing Project→Rebuild all. There should be no errors or warnings.

Setting up the Project Manually

For your information (or if there is some problem) here is how to set up the project yourself:

Make sure that you have copied all of the .cdb files from the directory %HEAPI_DIR%\heron_api\cmd into the directory C6000\bios\include under the directory where your Code Composer Studio installation is (usually c:\ti).

The First HERON module.

In Code Composer, select 'Project →new' and choose the path for your project. The name must be module1 for this demo.

Select 'File → New → DSP/BIOS Config' and choose the correct .cdb file for your hardware. This

will have a name that uses your HERON module number and possibly an option that is available for that module.

In the DSP/BIOS config tool, right click on Global properties, and check that the CLKOUT property is set to the frequency of your processor module. This is used by DSP/BIOS to calculate the correct settings for the timer period.

This .cdb file has some items set up which are for HERON-API. DO NOT CHANGE THESE!

For this example you need to set up a task that is called TSK0. Under its properties set its function to be “_maintask”.

Use ‘File → Save’ to save the cdb file to the project directory as module1.cdb.

Saving the .cdb file will generate a .cmd file, but that file will not place the sections heronapi_code and heronapi_data. For this reason there is a .cmd file supplied by us, in the directory %HEAPI_DIR%\heron_api\cmd that will be called by your heron module number and have _slbios.cmd at the end, i.e. heronx_slbios.cmd. You need to copy this to your project directory. This cmd file also adds the stio62s.lib to the project. It is done here as it must be in the project before the rts6201.lib which also defines the standard I/O functions.

Now add the source file to the project and the .cdb, and also the heronx_slbios.cmd. Edit the .cmd file that you have inserted and change the .cmd file that it includes to replace the ***** by the name of your .cdb file, i.e. change *****cfg.cmd to be module1cfg.cmd.

Because Code Composer Studio does not support the use of environmental variables in the library path you also need to change the line that has %HESL_DIR% to have the actual path name of where you installed the Server/Loader.

Add the HERON_API library “herons.lib” (in directory %HEAPI_DIR%\heron_api\lib) to the project.

Go to Project Options and add %HEAPI_DIR%\heron_api\inc and %HESL_DIR%\inc to the include path.

Select -o3 optimisation from the compiler optimisation menu.

The default .cdb file will actually place all code into external memory, and switch on the program cache. This is a good general purpose setting, but might need to be changed for your actual application.

Next, you need to add the file “stub.c” to the project (Project→Add Files to Project, select “stub.c” that resides in the example directory that you created). Edit this file and make sure that the proper heron file is included (e.g. if you have a HERON1 module, ensure that “stub.c” has “#include <heron1.h>”, if you have a HERON4 module, ensure that “stub.c” has “#include <heron4.h>”, and so on).

You can now build the demo by choosing Project→Rebuild all. There should be no errors or warnings.

The second HERON module.

Repeat the process above to build the DSP code for the second module, but use the name module2 in place of module1.

Running the Demo

The DSP application is now ready for use in the example, by running the demo. You can do this after you have built the host side program, by running the w32.bat. The DSP JTAG chain must be released before you can successfully run the demo. When you use Code Composer Studio to build the project it will leave the processor halted unless you choose Debug→Run Free before you exit. You can achieve the same thing after you have exited Code Composer Studio by running the API JTAG reset function for each board. Better, use GroupTools→Group JTAG reset from the Confidence checks program to JTAG reset all boards in the Group.