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.demon.co.uk
URL: http://www.hunteng.co.uk

The “mysl” Server/Loader example.

Rev 3.1 P.Warnes 19-7-00 (changed to reflect CCS 1.2 and additional HUNT CCS plug ins)
Rev 3.2 JT 15/03/01 (changed to incorporate SL plugin’s enhanced capabilities)

The DSP code in the “mysl” example demonstrates the use of the stdio library supplied with the Server/Loader.

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 2.0 made for HERON-API V2.3

Example revision 3.0 made for CCS V1.2

Example revision 3.1 made for Server/Loader 3.2

Example revision 3.2 made for Server/Loader V3.3

Example software

The example that we supply is a C file called stdio.c. It needs to be built using Code Composer Studio and uses the HERON-API software that has been installed on your PC when you did the "install drivers and tools" from your CD.

Hardware setup

The example shows the communication between a HERON module and the host. This means that the HERON module must be connected to the host via a HERON FIFO.

The demo as shipped is for an HEPC8 with the module in HERON slot 1. The module has its default routing jumpers set to 0, so that it boots from the host.

If you are running the demo on a different hardware configuration, you will need to change the network file that describes the connections to the Server/Loader.

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 the Start→Programs→HUNT ENGINEERING→AutoConfigure CCS.

We assume that this is already set up, but the plugin also copies cdb files etc into the correct locations.

When you start with the mysl example, simply copy the source files from the CD into a new directory. Then start Code Composer and choose Tools→HUNT ENGINEERING→Create new Heron-API project. This will guide you through setting the project up, and as long as you choose the name "stdio" for the project will incorporate the "stdio.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. You can now build the demo by choosing Project → Rebuild all. There should be no errors.

Manually Setting up the Project

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).

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

Saving the .cdb file wll 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 democfg.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" from the 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 → re-build all. There should be no errors.

Loading and Running the Example

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 my32.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.

This demo is almost the same as the stdio example for the Server/Loader but it uses the Server/Loader that you have built yourself, allowing you to make additional features.