



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 “stdio” 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)
Rev 3.3 JT 04/03/02 (changed to incorporate HEPC9)
Rev 3.4 JT 18/02/03 (added troubleshooting tips)

The DSP code in the “stdio” 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 V3.2

Example revision 3.2 made for Server/Loader V3.3

Example revision 3.3 made for HEPC9, Server/Loader V3.4

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. If you have a HEART based board this means that you must create a FIFO connection by adding HEART statements in the network file. For the HEPC8, this means that the HERON DSP module must be in slot 1 of the HEPC8, and the routing jumpers on the module must be set to boot from FIFO 1.

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.

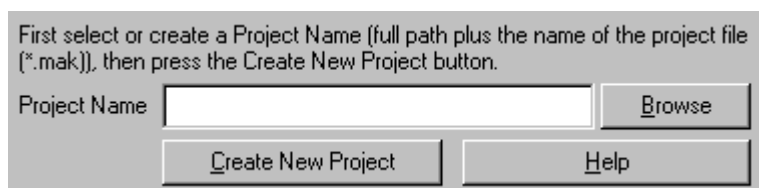
Starting

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

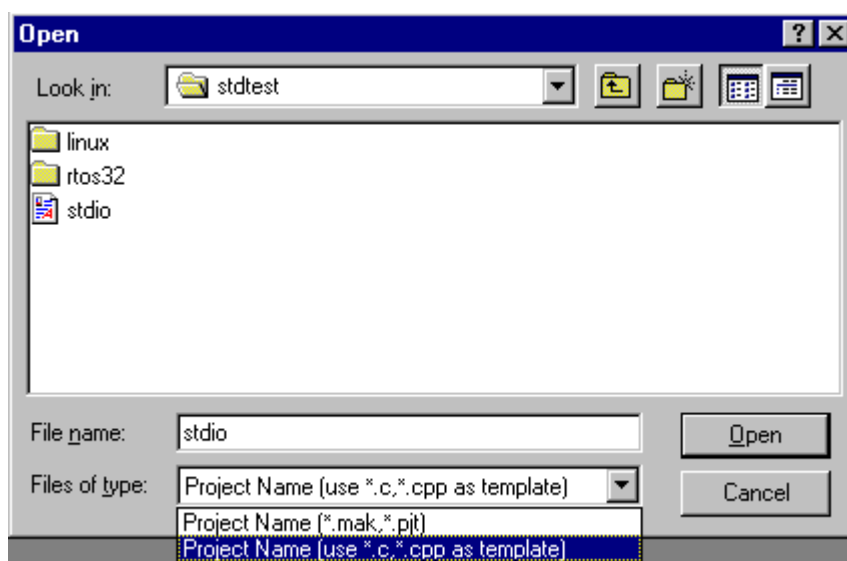
HUNT ENGINEERING provide several Code Composer Plug-in tools that allow you to make your development faster. The first is one that sets up Code Composer ready for your hardware, so you don't need to configure device drivers etc and can be found from the Start→Programs→HUNT ENGINEERING→AutoConfigure CCS. We assume that this is already set up, but the AutoConfigure plugin also copies `cdb` files etc into the correct locations.

When you start with the `stdtest` 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.

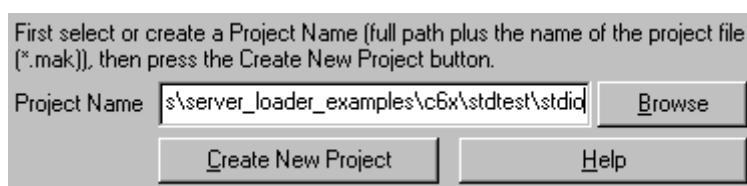


This will guide you through setting up the project, and as long as you choose the name “`stdio`” for the project will incorporate the “`stdio.c`” file. Simply browse to the “`stdtest`” example directory, change the “Files of Type” field to “Project Name (use `*.c`, `*.cpp` as template) and select “`stdio.c`”. Then click the

“Open” button. This will generate a project name that is based on the C file name (“stdio.prj” or “stdio.mak” from “stdio.c”).



When the project name has been created, create the project by clicking “Create New Project”.



Create the project for the Server/Loader; the plug-in creates and includes a file called “stdio_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; build the demo by choosing Project → Rebuild all. There should be no errors, or warnings.

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 “stdio” 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 stdio.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 stdiocfg.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).

Build the demo by choosing Project → re-build all. There should be no errors, or warnings.

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 w32.bat in a DOS-box. 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.

Network file

The “Create new HERON-API Project” plug-in will have created a template network file. This network file will have 1 DSP processor declared. If your carrier board is HEART-based, it will also have 2 FIFO connections defined, using 2 HEART statements. The 2 HEART statements together create a bi-directional link between the host interface and the DSP. This is necessary so that the Server/Loader “stdio” functions can be executed.

If you have a HEART based carrier board, such as the HEPC9, let’s have a quick look at the contents of the network file. We only need to define a DSP module and a host interface:

```
c6 0      heron  ROOT      (0)    00000001  c:\heapi\examples\c6\stdio.out
pcif 0     host1  normal          0x05
```

Make sure that the HERON-ID is correct for your system situation.

The first field defines the module type. The ‘c6’ can be used for HERON1, HERON4, and other HERON processor types. The ‘fpga’ can be used for FPGA and HERON-IO modules. The ‘gdio’ can be used for any HEGDxx module. The ‘ibc’ stands for Inter Board Connector. The ‘pcif’ stands for PC InterFace. Note that the filenames are arbitrary. Although they must be there, the actual values are not used – by HeartConf. If you also use the same network file with the Server/Loader, then it’s probably

useful to have a proper filename – which is then used by the Server/Loader.

Finally, it's time to connect the different modules up. For example, to connect the C6x to the host:

```
heart      host1  0      heron    0      1
heart      heron  0      host1   0      1
```

A 'heart' means: make a one way FIFO connection. Next, specify the 'from' module (here "host1") and 'from' FIFO (here "0"), then the 'to' module (here "heron") and the 'to' FIFO (here "0"). These two connections create a duplex FIFO connection between the host and the C6x. The C6x can reach the host by reading/writing FIFO 0. The host can reach the C6x by reading/writing its FIFO 0.

As another example, let's create a connection between the C6x (fifo 1) and the fpga (fifo 2).

```
heart      fpga1  2      heron    1      t=0
```

This creates a simplex connection from the FPGA module to the C6x. The FPGA can reach the C6x by writing to its FIFO 2. The C6x can read what the FPGA wrote by reading its FIFO 1. As a last example, let the FPGA read the GD output using FPGA FIFO 2:

```
heart      gd12   3      fpga1   3      3
```

As a last remark, the last field of a 'heart' statement is the number of timeslots. You can choose between 1, 2, 3, 4, 5, and 6. HeartConf will automatically allocate an actual timeslot. If you want to allocate a timeslot yourself, you can use the "t=" or "v=" specifiers – as shown above. For more details on how this works, please refer to the Server/Loader manual.

After all this, sorry, but we don't need to use a network file for this demo. It only uses 1 processor but no FIFO's. However, other examples do use FIFO's – for example the GDIO examples, the multiple processor examples, and the host examples. With those examples you may very well want to use this functionality. You can still try it here, if you wish. Creating FIFO connections won't alter the way this example works. Simple click – enable the "Program HEART ..." field, and choose or create a network file. Close the Properties window and do the reset.

Troubleshooting

Depending on HERON module type and size of the code and data in your project, you may find you encounter an error message as follows:-

```
Error: can't allocate .IDRAM$heap ...
```

The reason is that the sum of (software) segment sizes that are to be placed in the IDRAM memory is larger than the IDRAM memory size. Solution is to reduce in size one or more (software) segments, or to move software segments from IDRAM onto external memory (for example SBSRAM, SDRAM).

Projects created with the Create New HERON-API project plug-in are configured to create a 'map' file. In your project directory you can find a *.map file (where * is the name of your project) after you built your project. The 'map' file will show you where the different software segments have been placed. This allows you to see segment sizes and where they were placed.

A snippet from the *.map file of this example would show you:-

```
.bss      0      800000a0      00005590      UNINITIALIZED
          800000a0      00004080      stdio.obj (.bss)
          80004120      0000081c      stio62s.lib : bootload.obj (.bss)
```

The '.bss' segment is all global data found in all software sections of your code and libraries. You can see immediately that the C example file takes quite a chunk of memory: 0x4080 bytes. When we inspect the C file, stdio.c, we can see that this is mostly 'static int Data[4096]'. One way of solving the problem is to dynamically allocate the memory, instead of using a static global array:-

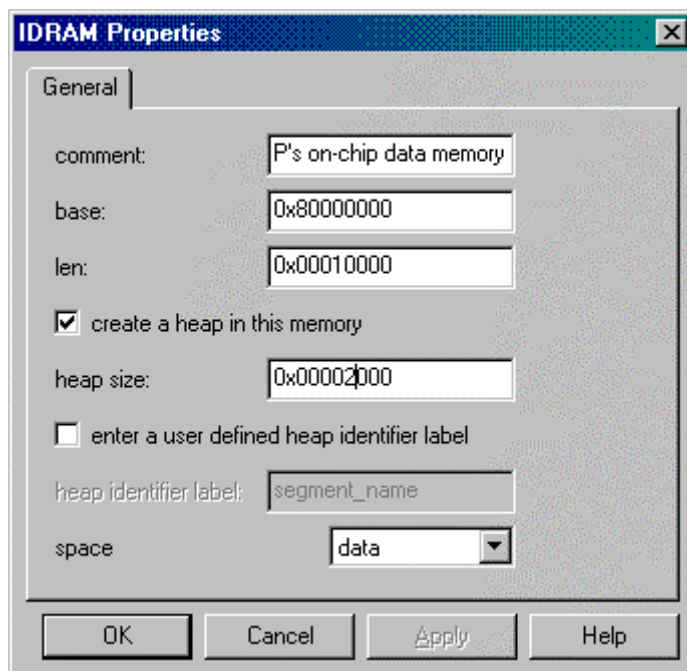
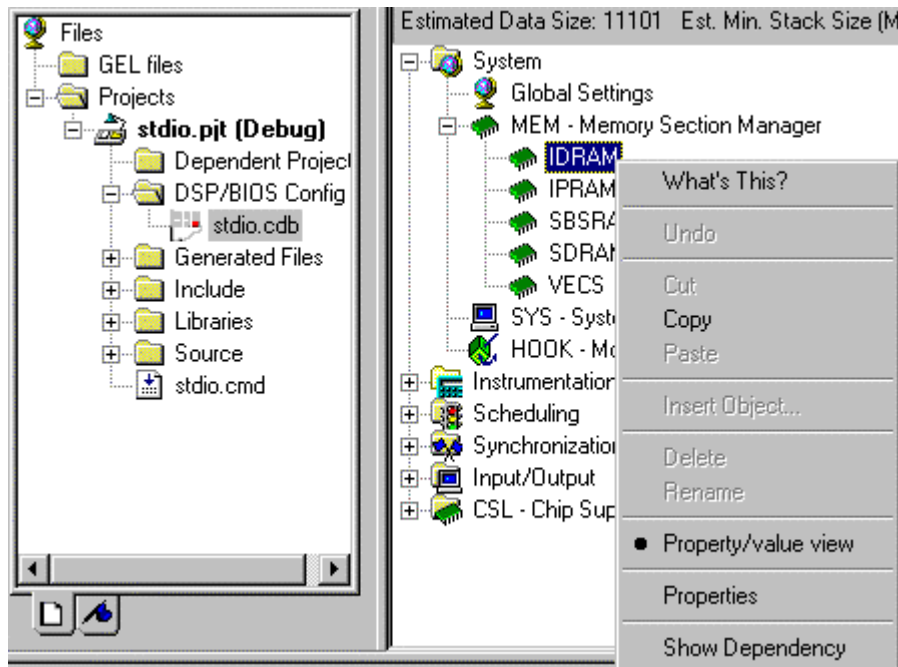
```
Int * Data = NULL;
Data = malloc(4096*sizeof(int));
```

Dynamically allocated memory is allocated from the (user) heap. The user heap is located in external

memory: SDRAM. However, you don't always want to put your data in external memory.

But sometimes you want to have your global data in internal memory. In that case, to fix the problem, instead of dynamically allocating global data, try reducing the size of one or more segments. One of the segments you could reduce is size is the system stack or the system heap (note: *system* heap is used by DSP/BIOS, *user* heap by your program; they are different). DSP/BIOS uses the system heap to dynamically allocate DSP/BIOS objects. For projects that use few DSP/BIOS resources it's safe to reduce the size of the system heap from the default 0x4000 bytes to 0x2000 or even 0x1000 bytes.

To reduce the IDRAM heap size, open your CDB file. Go to 'MEM – Memory Section Manager' and open the properties for IDRAM (right-click on the IDRAM entry). There's an entry 'heap size' which defaults to 0x4000. Change this to 0x2000 or 0x1000. Click OK. Now rebuild.



The heap in IDRAM is used by the 'system' (ie DSP/BIOS). (If you would try to un-tick 'create a heap in this memory' and CCS will warn you that try to remove system heap.)