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 HERON_IO5V example3

Rev 1.1 T. Hollis 29-04-05

The HERON-IO5V module is a module that has an FPGA and 2 channels of fast A/D along with 2 channels of fast D/A.

Most users will use the FPGA to provide either a custom I/O capability or a processing resource that uses the FPGA for that processing. In that case the Example3 project can form a starting point for the development of that.

Other users may want to simply use the HERON-IO5V as an I/O module in which case the example3 bit stream for HERON-IO5V is provided and the functionality of that bit stream is described in this document.

History

Example revision 1.0 08-04-04 First written for HERON-IO5V documentation only

Example revision 1.1 29-04-05 Removed reference to specific ISE versions

HUNT ENGINEERING is a trading style of HUNT ENGINEERING (U.K.) Ltd, Co Reg No 3333633
Directors P.Warnes & N.J.Warnes. Reg'd office 34 & 38 North St, Bridgwater, Somerset TA6 3YD. VAT Reg'd No GB 515 8449 31

What the bitstream does

The example3 project for the HERON-IO5V is supplied on the HUNT ENGINEERING CD, along with the bit stream that can be loaded directly onto the HERON-IO5V.

If you make changes to the project and re-build it you can change the functionality to be whatever you want, but if you use the supplied bit stream you need to know what it is doing:-

The standard clock soldered to "User Osc1" of the HERON-IO5V is 100MHz.

On the HERON-IO5V there is a choice of external clock source using the AC coupled clock input, or a clock source from within the FPGA for the A/Ds or D/A's.

The HERON FIFO clocks must be driven at between 60MHz and 100MHz for the HEPC9.

2v1500fg676.hcb   HERON-IO5V fitted to HEART based carrier e.g HEPC9 (100Mhz FIFO clocks)
and using AC clock input for A/D clock and D/A clock.

2v3000fg676.hcb   HERON-IO5V fitted to HEART based carrier e.g HEPC9 (100Mhz FIFO clocks)
and using AC clock input for A/D clock and D/A clock.

For an HEPC8 the FIFO clocks must be less than 60MHz.

2v1500fg676_pc8.hcb   HERON-IO5V fitted to HEPC8 (50Mhz FIFO clocks)
and using AC clock input for A/D clock and D/A clock.

2v3000fg676_pc8.hcb   HERON-IO5V fitted to HEPC8 (50Mhz FIFO clocks)
and using AC clock input for A/D clock and D/A clock.

When the bitstream is loaded onto the HERON-IO5V, the FIFO connections can be selected by sending an HSB "user message". The registers are address 0 = the fifo number that the FPGA reads from, and address 1 = the fifo number that the FPGA writes to. These are normally (but not necessarily) the same.

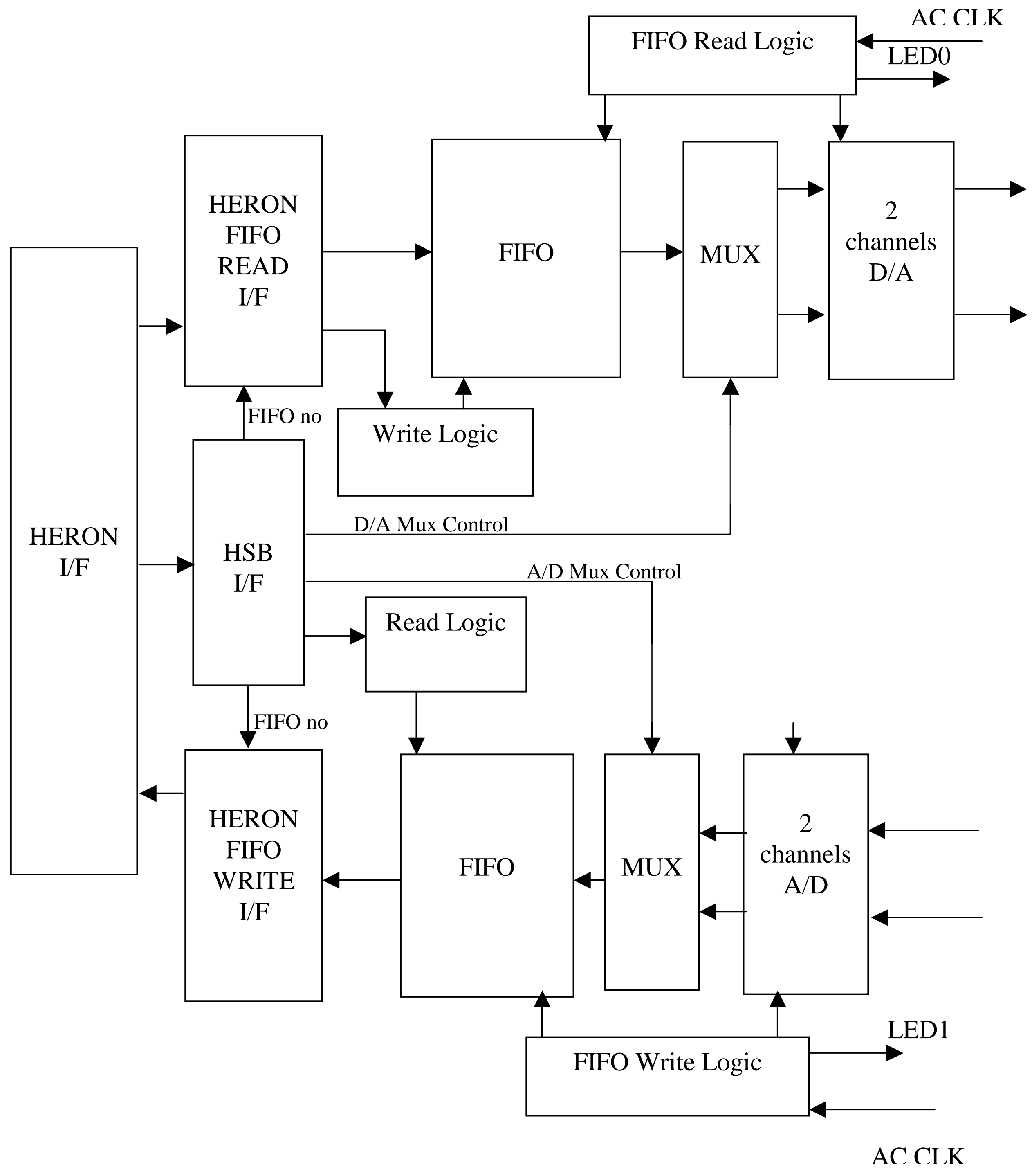
The bitstream defaults to use 1 channel of A/D and 1 channels of D/A. These can be overridden by using an HSB "user message". At address 2, the bottom bit of the register selects one A/D channel (set to 0) or 2 A/D channels (set to 1). At address 3, the bottom bit of the register selects one D/A channel (set to 0) or 2 D/A channels (set to 1).

The selected number of channels use FIFOs that are embedded in the FPGA to buffer data between the FIFOs and analogue components.

If the A/D FIFO internal to the FPGA becomes full, then A/D data will be lost. In that case the Example3 bitstream lights user LED0 for about 1/10th of a second, and on the next sample written to the data stream there is a bit set to show a data loss has occurred. This is enough to make it obviously visible. It will remain lit if data continues to be lost, but will go out if after the 1/10th of a second data is no longer being lost. This provides a simple visual indication if the A/Ds are being clocked faster than the data is being consumed.

If the D/A FIFO internal to the FPGA becomes empty, then the D/A cannot be updated with new data. In that case the Example3 bitstream maintains the previous value for the DACs and lights user LED1 for about 1/10th of a second. This is enough to make it obviously visible. It will remain lit if data continues to be unavailable, but will go out if after the 1/10th of a second data becomes available again. This provides a simple visual indication if the D/As are being clocked faster than the data is being provided.

FUNCTIONAL BLOCK DIAGRAM



A/Ds

As discussed above the A/D clock is generated using the external AC clock input. That is, the frequency of sampling is the same as that input.

Using an external clock allows a user of the example3 IP to set their sample rate as required by their own system needs. This rate should not exceed the rate at which the consumer can read it. This rate will be different for a DSP an FPGA or a Host machine.

If the A/Ds are being clocked faster than the data is being consumed, the user LED0 lights to indicate that data is being lost.

Example 3 provides an input FIFO inside the FPGA that has 511 locations. Depending on the number of channels selected the data in the FIFO has the format:-

One channel

Bit 31	Set to 0
Bit 30	FIFO has overflowed – this is first new data
Bit 29	Ch A THU – set if top half of range is used
Bit 28	CHA OTR – set if overrange
Bit 27..16	Ch A data – signed 12 bit (sample n+1)
Bit 15	Set to 0
Bit 14	FIFO has overflowed – this is first new data
Bit 13	Ch A THU – set if top half of range is used
Bit 12	Ch A OTR – set if overrange
Bits 11..0	Ch A data – signed 12 bit (sample n)

Two channnels

Bit 31	Set to 0
Bit 30	FIFO has overflowed – this is first new data
Bit 29	CHB OTR – set if overrange
Bit 28	Ch B data – signed 12 bit (sample n)
Bit 27..16	Ch B THU – set if top half of range is used
Bit 15	Set to 0
Bit 14	FIFO has overflowed – this is first new data
Bit 13	Ch A THU – set if top half of range is used
Bit 12	Ch A OTR – set if overrange
Bits 11..0	Ch A data – signed 12 bit (sample n)

Data is always written into the FIFO at the rate of the input AC clock, and transferred from the internal FIFO to the HERON FIFO whenever there is space in the HERON FIFO, and data in the internal FIFO.

This makes Example3 an ideal candidate for "Data Acquisition" applications, where the internal FIFO can absorb the A/D data if the consumer interrupts it reading for short periods. An example of that is when a DSP is a consumer of the data, and the I/O interrupts are not serviced because of some other event on the DSP.

D/As

The D/A clock is generated using the AC clock input. That is, the frequency of sampling is the same as that input, but the phase difference between the input and the D/A update clock is not known.

Using an external clock allows a user of the example3 IP to set their sample rate as required by their own system needs. This rate should not exceed the rate at which the producer can write it. This rate will be different for a DSP an FPGA or a Host machine.

If the D/As are being clocked faster than the data is being provided, the user LED1 lights. The D/A is simply not updated, and maintains it's last output value. This ensures that no output glitches can occur.

Example 3 provides an output FIFO inside the FPGA that has 511 locations. Depending on the number of channels selected the data in the FIFO has the format:-

One channel

Bit 31-16	Channel A data (Sample n+1)
Bit 15-0	Channel A data (Sample n)

Two channels

Bit 31-16	Channel B data (Sample n)
Bit 15-0	Channel A data (Sample n)

Data is always transferred from HERON FIFO into the internal FIFO whenever there is space in the internal FIFO, and data in the HERON FIFO. For each AC clock rising edge, data is taken from the internal FIFO and strobed into the D/As if there is data available in the internal FIFO.

This makes Example3 an ideal candidate for "Signal output" applications, where the internal FIFO can buffer the D/A data if the producer, produces the data in bursts.

This makes the normal use of the example bitstream as follows :-

- load bitstream for example3
- Send HSB message to configure the input and output FIFOs
- Send HSB message to configure the number of channels
- Loop outputting D/A data and reading A/D data.

Which is what the examples provided show.

Where are the bitstream and examples?

The bit streams for this example can be found on the HUNT ENGINEERING CD under \fpga\io5v\data_streaming(ex3). The name of the .hcb file reflects the FPGA part number, as explained earlier in this document.

The DSP and host examples can be found in the dsp and host subdirectories.

An easier way to navigate to the correct directory is to select the "Files" link next to the "Data Streaming" link under the IP sections of the CD browser.

The source files for the FPGA example can be found in the \src subdirectory. The sources in the \io5v\common directory are also required. There is a project for ISE in the 'ISE' subdirectory.

What example do I use?

We have provided two pieces of software as examples to use with the example3 FPGA bit streams.

One runs on your Host PC, and requires a FIFO connection between the Host PC and the HERON-IO5V.

The other runs on a HERON-C6000 module and requires a FIFO connection between the DSP module and the HERON-IO5V.

If you do not have a DSP module then you will need to use the Host based example program, but if you have a DSP module then you can choose which example type you want to use. Choose whichever example is closest to your needs.

The following sections describe how to use the Host based and DSP based examples. Choose which one of these you want to follow and simply skip the other section.

C6000 DSP Example software

HEART carrier like HEPC9

If you have a HEART based module carrier like HEPC9, you can run the DSP based example code with your HERON-IO5V and C6000 modules fitted to any slots. Then you need to configure the HEART connections between the modules using HeartConf, or alternatively use default routing jumpers to make the connection.

If you use HeartConf, then a network file that connects the two modules as follows

```
# For HUNT ENGINEERING's Device Driver API use:
# BD API        Board_type      Board_Id        Device_Id
#----------------------------------------------------------------------
# Using API
BD API HEP9A 0 0
#
# Nodes description
# ND  BD_nb  ND_NAME  ND_Type  CC-id HERON-ID  filename(s)
#-------------------------------------------------------------------
  c6   0      dspmodule  ROOT     (0)   00000001  mydspprog.out
  fpga 0      heronio5   normal         00000002  nofile

  ibc  0      ibc1       normal          0x06
  pcif 0      NodeC      normal          0x05

#--------------------------------------------------------------------
#       from:slot  fifo  to:slot   fifo   timeslots
#--------------------------------------------------------------------

heart      dspmodule  2      heronio5   3     1
heart      heronio5   3      dspmodule  2     1
```

could be used to make a connection from FIFO#02 of the DSP module to FIFO#3 of the HERON-IO5V. Only one timeslot has been allocated in HEART in each direction, but more could be allocated if necessary. Of course you need to modify the HERON-IDs to correctly show which slots your modules are fitted to.

DSP example : What it does

We supply a c source file for a C6000 based HERON module. Before you can use it you need to make a new project that matches the C6000 module type that you have.

Once the DSP program is compiled and loaded onto the DSP it will be used to gather the data captured by the HERON-IO5V A/Ds into the DSP memory.

The program is called 'example3.c' and it is located in the 'dsp' sub-directory of this example. This example can be used both as a confidence check and a starting point for your development.

After you have loaded the correct bit stream into your HERON-IO5V module, the "DONE" LED should be switched off showing that the configuration was successful. Also The USER LED4 should flash about once per second, showing that the clocks are properly running in the FPGA. If the DONE LED is off, but the LED is not flashing it may be because the Delay Locked Loop used in the clock circuit of the FPGA needs to be reset. You can do this using the utility under "programs → HUNT ENGINEERING → API board RESET" if you want, but such a reset is better made using the HUNT ENGINEERING Reset Plug in for Code Composer Studio.

If you have a HEART based carrier (like an HEPC9) you will need to set the FIFO connections that will be used in your system. You will do this using the Heartconf program, which can be called from the HUNT ENGINEERING reset plug in for Code Composer Studio. This is probably the best way as it will re-configure your connections every time you reset the system. If you are not already comfortable doing this you should review the HEART movies and documentation again.

The program will send HSB messages to set the FIFOs that the HERON-IO5V FIFO will use according to the defines made in your program. The program will then calculate a Sine and Cosine pattern that is used for the DAC data. This will be continually output by the DSP to the DACs of the HERON-IO5V. The program will then start to capture blocks of data from the HERON-IO5V A/Ds into the DSP memory. Then the DSP will "unpack" this data into a separate array for each channel, with the correct sign extension for further processing of that data. It is not recommended that the data be displayed using Code Composer's Graph routines as this needs a breakpoint to force the graphs to update. As example3 continuously streams data to the DSP, the breakpoint will stop the DSP reading the data and some will be lost.

DSP example: setting it up

If you have an HEPC9 the FIFO connections are determined by the settings that you made.

The example that we supply is a C file called example3.c. It needs to be changed to reflect your actual needs, and then built using Code Composer Studio.

The example is a HERON-API project that can be set up using the 'Create new HERON-API project' plug-in. To do this, choose "Tools→HUNT ENGINEERING→Create new HERON-API Project" from inside Code Composer Studio. This will guide you through setting up the project and as long as you choose the name "example3" for the project it will incorporate the example3.c source file.

This project will incorporate the correct HERON-API library for your module and Module carrier combination.

You must review some settings at the top of the source file, and change them to reflect your system setup.

Firstly the lines

```
#define INFIFONO  2       /* FIFO through which IO5V Data is received */
#define OUTFIFONO  2       /* FIFO through which IO5V Data is sent */
```

are used to define which FIFO numbers the DSP will use.

Then the lines

```
#define DOINPUT   /* comment out for no inputs */
#define DOOUTPUT  /* comment out for no outputs*/
```

can be used to select if input, output or both will be run.

Then the lines

```
#define NO_IP_CHANNELS 2
#define NO_OP_CHANNELS 2
```

can be used to set how many channels will be used.

By setting the lines

```
#define FPGA_TO_DSP 3
#define DSP_TO_FPGA 3
```

you can set up which FIFO numbers the HERON- IO5V will use.

Finally

```
#define FPGA_SLOT              2
```

must be set to show which module slot the HERON- IO5V is plugged into. This is used in the HSB address.

Then you can build the example using Code Composer Studio.

DSP Example: using it

To run the example, load the program onto the DSP using the File→Load command of Code Composer. At this stage it is advisable to open the HUNT ENGINEERING Reset Plug in, and set the option to “halt processors, reload them and then run to main”. If you have a HEART based carrier and you will use HeartConf, select this also in the reset plug in. Then use this reset to reset the system, and reload it, which empties the FIFOs, and configures HEART if you need to.

But before you can run the example, you must have remembered to load the bit-stream onto the HERON- IO5V. What bit-stream to choose is shown above. If you don’t know how to load a bit-stream onto the HERON- IO5V, please review example1 once more. Verify that after the loading process the “Done” LED goes off, and now the system has been reset (using the plug in) the USER LED4 should be flashing.

It doesn’t really matter in what order the DSP and the HERON- IO5V is loaded. You may just as well first load the bit-stream and only then load the DSP. Finally, do a “Debug->Run” in Code Composer Studio to start the example.

The example3 starts by configuring the FIFOs and number of channels using the HSB. These are programmed according to the settings you configured near the top of the program.

The program then enters a loop that sends a buffer to the D/As, and uses double buffers to request buffers from the A/Ds. The A/D buffers are unpacked using a function from the HEL_Unpack library to split the data up from the two channels. Note the use of the Se version of the unpack function so that the sign bit is extended making the data be the C type “signed short”.

The array “errors” is used to perform an ”OR” of the top bits – allowing us to detect if the THU, OTR or FIFO overflow bits have been set anywhere at all in the buffer.

NOTE: stopping the program with Code Composer Breakpoints will cause A/D data to be lost and the D/A data to stop being output. When this situation occurs, it may be necessary to re-start the program to recover.

Also using printf or long processing functions will reduce the rate at which the I/Os can be clocked.

The DAC outputs can be viewed on an oscilloscope.

DSP example: Changing and Building it

In the sections above you have already changed and built the DSP code a number of times. It uses DSP/BIOS and HERON-API.

DSP/BIOS

DSP/BIOS is the multi-threading environment provided as part of the Code Composer Studio development environment. It also provides 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 Studio 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.

Performance expectations

When being used with a DSP, like the HERON4, the usual problem with an I/O stream is that the processor interrupt latency can sometimes be extended by several interrupt sources happening at the same time. The FIFOs provided in the FPGA when using the example3 bit stream can be used to "even out" those problems and increase the overall sampling rate that can be achieved without losing data. These extended interrupt latencies will still occur though, and cause uncertainties in the latency of the system. This means that, for most of the time, the system latency will be one value, but when an extended interrupt service time occurs, the data will be buffered in the FIFOs. When the system returns to "normal" function, the latency in the processing will be longer than before, and will gradually reduce to the value before as the processing and I/O recovers from this event.

When using I/P only, and a dedicated DMA, data can be read by the HERON4 at the equivalent of 50Mhz for 2 channels, or 100Mhz for one channel. That is 200Mbytes/sec. This requires three time slots in the HEART ring.

When using O/P only, and a dedicated DMA, data can be written by the HERON4 at the equivalent of 50Mhz for 2 channels, or 100Mhz for one channel. That is 200Mbytes/sec. This requires three time slots in the HEART ring.

When using both I/P and O/P and dedicated DMAs, data can be transferred by the HERON4 at the equivalent of 17Mhz for 2 channels in and 2 channels out (136Mbytes/second), and 31Mhz for one channel in and one channel out (124Mbytes/second).

All of the above measurements are made with the example3 program supplied, which is to say, transferring only data to and from the internal memory of the DSP. Any use of external memory will reduce these performances as can be demonstrated by the example3a and example3b programs.

It is worth noting that the minimum ADC sample frequency is 40MHz, there is no minimum sample frequency for the DAC's.

Example3a

Example3a is an example of using only the A/Ds to grab data, and store it to the external DRAM buffer of a DSP module. It is a modified version of the example3.c that simply reads the A/D inputs until the DRAM is filled.

It is written as a Server/Loader application as it uses the stdio functions of the Server/Loader to write the results to a file when the buffer is full. In order to be able to malloc virtually the whole of the 16 Mbytes of SDRAM it is necessary to change the cdb file so that all of the SDRAM can be used for heap. Simply change the "heap size" setting to be 0x01000000.

This example can run at 35MHz with a single channel, i.e. 70Mbytes/second, but to achieve this will require 2 time slots in the HEART ring. Each time slot has a capacity of 66.67Mbytes/second. This reduced rate relative to example3 is because the data is being written directly to the SDRAM of the processor, using the same memory bus as the FIFO accesses.

Example3b

Example3b is an example of using only the D/As and playing a long patter that is stored in external SDRAM. In this example an 8Mbyte pattern is generated and then sent to the DACs continuously.

In order to be able to malloc the 8 Mbytes of SDRAM it is necessary to change the cdb file so that all of the SDRAM can be used for heap. Simply change the "heap size" setting to be 0x01000000.

This example can run at 35Mhz with a single channel, i.e. 70Mbytes/second but to achieve this will require 2 time slots in the HEART ring. Each time slot has a capacity of 66.67Mbytes/second. This

reduced rate relative to example3 is because the data is being written directly to the SDRAM of the processor, using the same memory bus as the FIFO accesses.

Host based Example software

HEART carrier like HEPC9

If you have a HEART based module carrier like HEPC9, you can run the Host based example code with your HERON-IO5V module fitted to any slot. Then you need to configure the HEART connections between the module and the Host using HeartConf. This is done from inside the hegraph program for you using the network file that is in the same directory as the host example software.

Host example : What it does

The example that we supply is a C file that can be added into a Microsoft Visual C++ console mode project along with the hendrv.lib (host API library).

This example will stream data to and from the HERON-IO5V over the PCI FIFOs. It is not very useful to do this because the operating system on the HOST will spend long periods not servicing interrupts. This means that the example can run at rates as high as a few Mhz, but will lose data every few seconds.

Host example : Using it

First the correct example2 bit-stream for your module should be programmed into the FPGA using the standard Windows FPGA programmer found under "Programs → HUNT ENGINEERNG → Program HERON FPGA". What bit-stream to choose is discussed in the above sections and depends on the type of HERON-IO5V you have and the type of module carrier you have.

If you don't know how to load a bit-stream onto the HERON-IO5V, please review example1 once more. Verify that after the loading process the "Done" LED goes off. Finally, execute the hostex3 program.

Host example: Changing and Building it

The file hostex3.c can be copied from the CD. It needs to have its attributes changed so that it is not read only.

Make a console mode project, and add the C file. Add also the hendrv.lib (usually in c:\heapi).

Change the project settings so that under "C/C++" "pre-processor" settings $(HEAPI_DIR) is added to the include path.

FPGA example code

You should understand the HUNT ENGINEERING VHDL support for HERON modules before looking at this section. If you do not then please review example1 again (the getting started example for FPGA modules).

This section only discusses therefore the things that are unique to example3.

Example3 has some options in the user_ap3.vhdl file that allow you to select some options for the FIFO clocking.

Just like in Example1 (the getting started example) you can select if the 100Mhz is divided by 2 to provide the FIFO clock frequency. This allows example3 to be able to clock the FIFOs at 50Mhz (suitable for HEPC8) or 100Mhz (suitable for HEPC9). You must also remember to set the HIGH_FCLK_G option to show if this clock is higher or lower than 60Mhz.

For example3 the correct options for an HEPC9 are :-

DIV2_FCLK	FCLK_G_DOMAIN	HIGH_FCLK_G	HIGH_FCLK_RD	HIGH_FCLK_WR
False	True	True	n/a	n/a

So both input and output FIFOs will be clocked at 100Mhz.

Example3 also has some options that need to be set for the A/D clocking.

For example3 using the AC clock (CLKIN) the correct options are :-

SCLK_G_DOMAIN	INTERNAL_SCLK_A
True	False

There are also options for clocking the DAC and in Example3 these should be set:-

INTERNAL_SCLK_DAC	DAC_PLL	DAC FREQUENCY
False	False	100

This sets the DAC's to be clocked by the external AC clock (CLKIN). In the default mode the DAC's come out of power on reset in a basic DAC configuration with the PLL disabled. With the PLL disabled a particular DAC FREQUENCY is expected as described in the 'DAC Clocks' section of 'Setting up the Configuration Package' in the HERON_IO5V User Manual. The example has been built with a DAC FREQUENCY setting of 100, ie 100MHz. In practice this allows the DAC sample clocks up to 100MHz to be used from the AC clock input.

There are FIFOs used in the example, that are taken from the Core Generator. If you want to understand using Core Generator cores in a design then review the "DSP with FPGA" tutorial.

You need to consider the timing constraints that are defined in the .ucf file for your design. Actually if you use a time specification that is more strict than needed there is no problem, so the standard .ucf file have the clock frequencies specified. If the project builds (as example3 does) with this specification it is still guaranteed to work at lower clock speeds. If you add new clock nets into your design then you need to add new timing constraints into your design.