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

HUNT ENGINEERING

HERON-FPGA7

HERON Module with Virtex II 3,6 or 8M gate FPGA, and 256Mbytes of SDRAM

USER MANUAL

Hardware Rev A
Document Rev C
R.Williams 06/05/05

TECHNICAL SUPPORT

Technical support for HUNT ENGINEERING products should first be obtained from the comprehensive Support section http://www.hunteng.co.uk/support/index.htm on the HUNT ENGINEERING web site. This includes FAQs, latest product, software and documentation updates etc. Or contact your local supplier - if you are unsure of details please refer to http://www.hunteng.co.uk for the list of current re-sellers.

HUNT ENGINEERING technical support can be contacted by emailing support@hunteng.demon.co.uk, calling the direct support telephone number +44 (0)1278 760775, or by calling the general number +44 (0)1278 760188 and choosing the technical support option.

TABLE OF CONTENTS

Introduction

The HERON module is a module defined by HUNT ENGINEERING to address the needs of our customers for real-time DSP systems. The HERON module is defined both mechanically and electrically by a separate HERON module specification that is available from the HUNT ENGINEERING CD, via the user manuals section from the CD browser, or online from http://www.hunteng.co.uk and going to the application notes section in the user area.

The HERON module specification also defines the features that a HERON module carrier must provide. HERON stands for Hunt Engineering ResOurce Node, which tries to make it clear that the module is not for a particular processor, or I/O task, but is intended to be a module definition that allows "nodes" in a system to be interconnected and controlled whatever their function. In this respect it is not like the TIM-40 specification which was specific to the 'C4x DSP.

As the HERON-FPGA7 was developed, HUNT ENGINEERING have already developed HERON modules carriers like the HEPC9, HERON processor modules (that carry various other members of the TMS320C6000 family of DSP processors from TI), and HERON-IO modules. In addition to these modules, the HERON specification is a super-set of the pre-existing HUNT ENGINEERING GDIO module, so the GDIO modules from our 'C4x product range can also be used in HERON systems.

The HERON module connects to the carrier board through several standard interfaces.

- The first is a FIFO input interface, and a FIFO output interface. This is to be used for the main inter-node communications. (It is usually also used for connection to the HOST computer if any).
- The second is an asynchronous interface that allows registers etc to be configured from a HERON module. This is intended for configuring communication systems, or perhaps to control some function specific peripherals on the carrier board.
- The third is a JTAG (IEEE 1149.1) interface for running processor debug tools.
- The fourth is the HERON Serial Bus, used for configuration messages.
- The last is the general control such as reset, power etc.

HUNT ENGINEERING defined the HERON modules in conjunction with HEART – the Hunt Engineering Architecture using Ring Technology. This is a common architecture that we will adopt for our HERON carriers that provides good real time features such as low latency and high bandwidth, along with software reconfigurability of the communication system, multicast, multiple board support etc., etc.

However, it is not a requirement of a HERON module carrier that it implements such features. In fact our customers could develop their own module carrier and add our HERON modules to it. Conversely our customers could develop application specific HERON modules themselves and add them to our systems.

The HERON-FPGA7 is HERON module that can be used for hardware signal processing or for flexible I/O.

The HERON-FPGA7 provides a Virtex II FPGA (Xilinx part number xc2v*000-*ff1152) in a choice of 3M, 6M and 8M gates. For more part information refer to the Xilinx web site at http://www.xilinx.com .

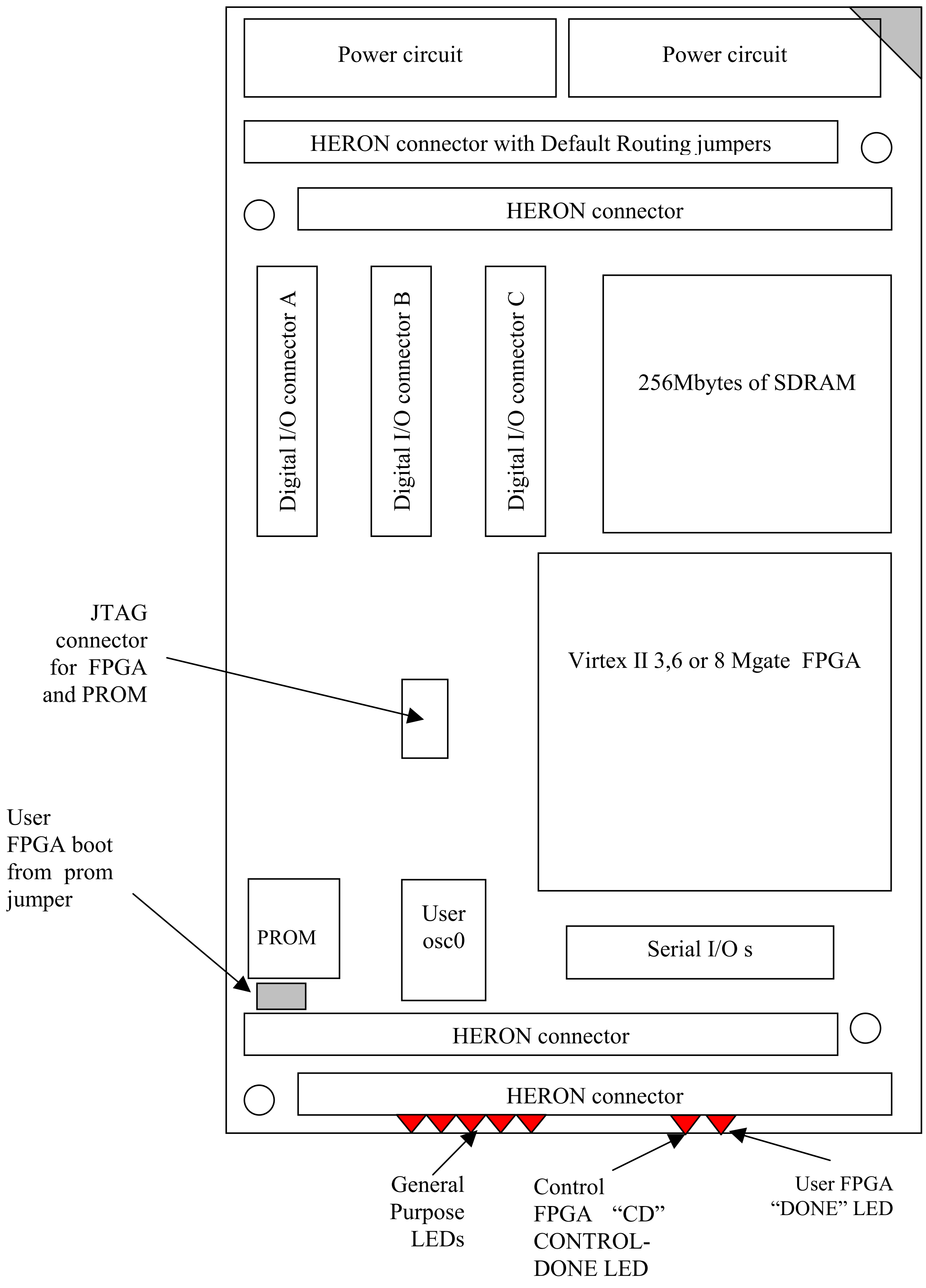
The HERON-FPGA7 connects all of the HERON module signals, except JTAG, to the FPGA, allowing flexible use of the module's resources.

The HERON-FPGA7 provides 256Mbytes of SDRAM, directly connected to the pins of the Xilinx FPGA. This memory can be used as general purpose memory by the FPGA to perform whatever storage function is required by the users design.

The HERON-FPGA7 also provides some of the FPGA I/Os connected to connectors on the module. This allows the FPGA to be configured for a variety of I/Os. There are also configurable components that provide RS232 and RS485.

The HERON-FPGA7 uses the HERON module's serial bus to download configuration bit-streams into the FPGA, allowing the user to configure it with standard functions provided by HUNT ENGINEERING or functions that they have developed themselves using the Xilinx development tools. It is also possible for the module to configure the FPGA from a PROM, but at the time of launching the module suitable PROMS were not yet produced by Xilinx. This is intended to simplify the deployment of systems after the FPGA functions have been fully developed. As soon as suitable PROMS are available from Xilinx, they will be included on new modules, and can be retro-fitted to existing modules upon request.

Physical Location of Items on the HERON-FPGA7



The HERON-FPGA7 is a module that plugs into a HERON module carrier.

The HERON-FPGA7 should be fitted to the carrier card along with any other modules that your system has and their retaining nuts fitted (see a later section of this manual for details).

The Default routing jumpers must be set correctly for the system. For most systems the FPGA based modules will not require any default routing jumpers to be fitted. This will allow the FPGA to access whatever FIFO is needs to, and will rely on the FIFO being connected to the right place by the carrier configuration. (See a later section for details on default routing jumpers).

The HERON-FPGA7, following reset, will enter a state where it can be interrogated and programmed using the HERON module's serial bus. It is addressed according to the Carrier number and the slot number of the HERON slot that it is fitted to.

The FPGA configuration data will have been generated using the Xilinx development tools. HUNT ENGINEERING provide examples for the HERON-FPGA modules in the correct format for use with the Xilinx ISE software. HUNT ENGINEERING also provides software for the Host PC that will allow the output files from the Foundation software to be loaded onto a HERON-FPGA module.

Follow the "Starting your FPGA development" tutorial from the HUNT ENGINEERING CD, and then the general FPGA examples found in the same place on the CD.

It could be possible to use the HERON-FPGA7 as an I/O or memory module using one of the example bit streams. In this case it is not necessary to be concerned how to program the FPGA – simply load the example bit stream and use it.

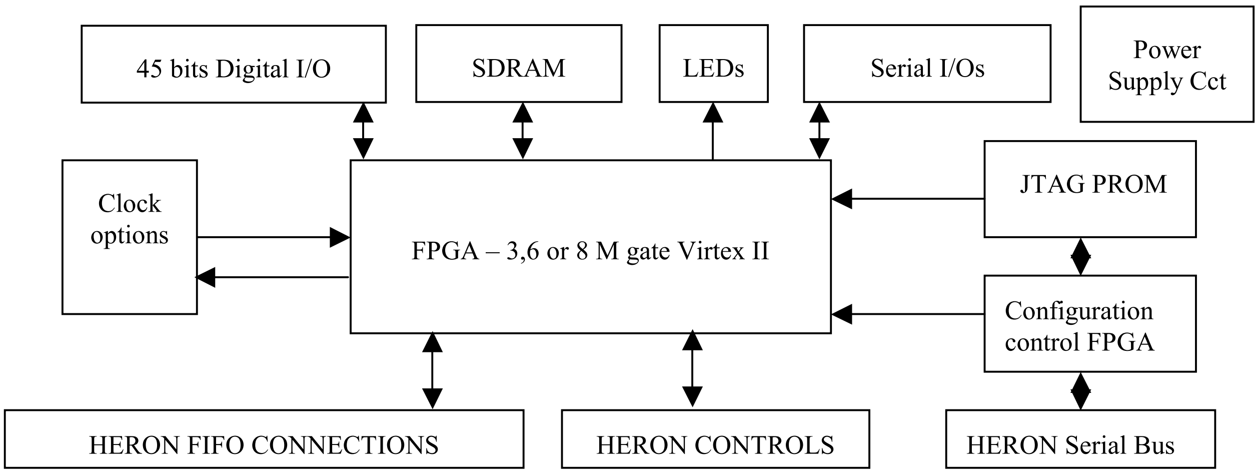
Standard Intellectual Property (IP)

HUNT ENGINEERING provides examples for the HERON-FPGA7 that perform different functions. It is possible to use these standard configurations directly if they fit your needs.

It could be possible to request a new standard example from HUNT ENGINEERING, which could avoid the need to purchase and learn how to use the FPGA development tools. Depending on the complexity of your request HUNT ENGINEERING may choose not to offer it, or to charge for it.

New IP for the HERON-FPGA7 will be posted on the HUNT ENGINEERING web site in the user area whenever it becomes available. HERON-FPGA7 users can then take advantage of that IP free of charge.

This section describes the features of the HERON-FPGA7 and why they are provided.



Serial Configuration of the User FPGA

The HERON-FPGA7 usually has the configuration of the user FPGA downloaded using the HERON module's serial configuration bus. This allows the use of "standard" configurations as supplied on the HUNT ENGINEERING CD, or of user defined configurations without the need to return the module to the factory.

It is imagined that as the "standard" set of functions grows, that they be made available to users of HERON-FPGA modules via the HUNT ENGINEERING web site or CD update requests. Also HUNT ENGINEERING will have the possibility to provide semi-custom configurations for a charge via email.

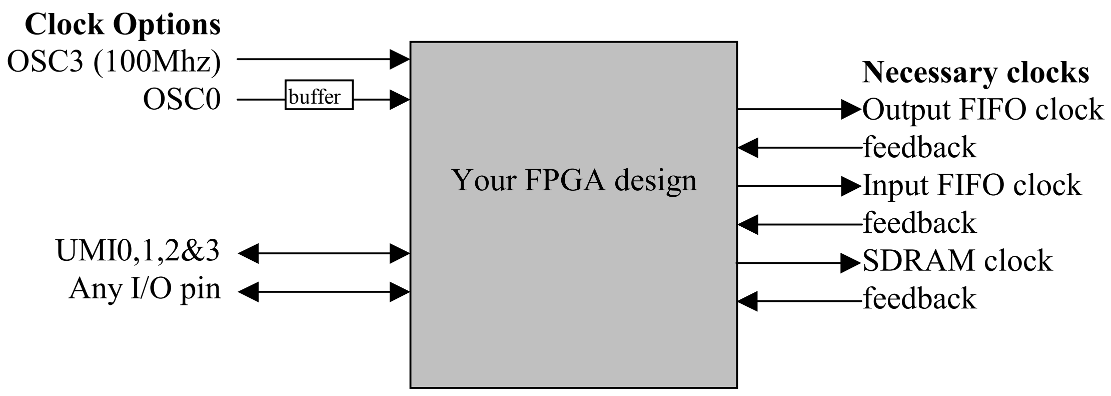
USER FPGA boot ROM

As an alternative to the serial configuration download, it is possible to configure the FPGA from a Flash based configuration PROM. This feature will only be available after Xilinx make the correct PROMS available to be fitted to the HERON-FPGA7 However, if a system is being deployed with a host machine such as a PC, it might be preferable to continue to use the serial configuration method, as this will make in field upgrades and bug fixes simpler to deploy.

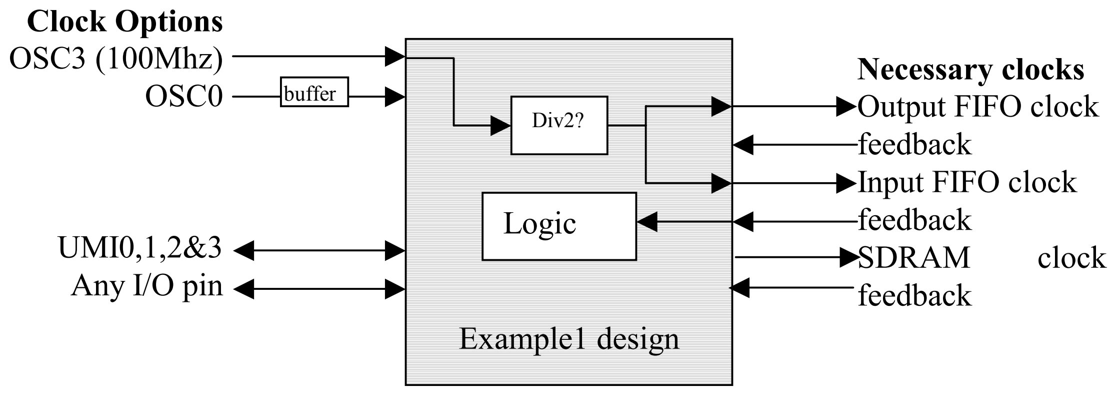
The PROM fitted to the FPGA7 is Flash based, and can be programmed (and re-programmed) using a Xilinx JTAG cable (such as Xilinx Parallel cable 3 or 4). An option in the JTAG download software can cause the FPGA to be configured on completion of the PROM being programmed.

Clocking of the FPGA

The Xilinx FPGA used on the HERON-FPGA7 does not have a single clock input, but rather it can use any one of its pins to provide a clock input. This means you can have many sources of clocks, each of which can be used inside your FPGA design. You can even divide these clocks using flip flops, or even multiply using digital clock manager components.



The simplest way to manage your FPGA design is to use just one clock throughout your design. However the FPGA must drive both of the HERON FIFO clocks, at a frequency that is suitable for your module carrier (see the documentation for your module carrier for details of its restrictions). The FPGA must also drive the SDRAM clock, at a frequency suitable for the SDRAM (133MHz). The FPGA may also need to use clocks for the digital I/Os. The frequency for these might be limited by the equipment that it is connected to, or by the needs of your signal processing. The needs for these clocks can only be determined by looking carefully at the needs of your system.



If these clocks cannot be the same, then the next best situation is to have one clock derived from the other. In that way the relationship between the clock edges will be known.

The most difficult case for your FPGA design is to have many clocks from different sources that are all used in the same design. Then you must carefully manage signals that cross from one clock "domain" to another. This can be handled by FIFOS, or by multiple registering to prevent metastability problems. Refer to texts on digital design to understand these issues.

The HERON-FPGA7 provides a highly flexible set of choices for the clocking of the FPGA.

The HERON-FPGA7 has a socket for a user oscillator. It has a further location for a surface mount oscillator.

Default shipping state is to have a 100Mhz oscillator fitted to the surface mount site – driving 100Mhz on UserOsc3. This is a standard commercial oscillator module, that is +-100ppm accuracy. If you require higher accuracy clocks then you should use one of the other clock sources.

Any of the digital I/Os on the digital I/O connectors or the UMI pins on the module connector could be used as a clock input or output.

HERON FIFOS

The HERON module can access up to 6 input FIFOs and up to another 6 output FIFOs. Actually it is most likely that a carrier board will not implement all 12 FIFO interfaces. Each FIFO interface is the same as the others, using common clocks and data busses.

The input and output interfaces are separate though, allowing data to be read and written at the same time by a module like the HERON-FPGA7.

While it is possible to read one FIFO and write another FIFO at the same time, the use of shared pins means no more than one can be written or read at the same time.

For each interface (input or output) there is a FIFO clock that must be a constant frequency, and running constantly. There may be some minimum and maximum frequency requirements for a particular Module Carrier card that the designer of the FPGA contents must be sure to comply with. This is because the FIFO clocks are generated by the FPGA, probably based on one of the clock inputs to the part.

Each FIFO interface has a separate "enable" signal that is used to indicate which FIFO is accessed using the clock edge.

Input FIFOs

The input FIFOs use a common data bus that is driven onto the HERON module. It is important to ensure that no more than one of the FIFOs are read at the same time, but more importantly that no more than one has its output enable selected.

By properly asserting the "read enable" and "output enable" signals relative to the clock the FPGA can access the FIFO of its choice at a rate up to one 32 bit word per clock cycle.

For the timing of those signals refer to the HERON module specification.

Each input FIFO interface provides Flags that indicate the state of the FIFO. An empty flag shows that there is no data to be read, an almost empty flag shows that there are at least 4 words left. While the almost flag is not asserted accesses can be made on every clock, but after it is asserted, it is better to make one access only, then check the empty flag on the next clock, before deciding if another access is possible.

Output FIFOs

The output FIFOs use a common data bus that is driven by the HERON module. It is important to ensure that no more than one of the FIFOs is written at the same time – unless that is required by your system.

By properly asserting the "write enable" signals relative to the clock, the user FPGA can access the FIFO of its choice at a rate up to one 32-bit word per clock cycle.

For the timing of those signals refer to the HERON module specification.

Each output FIFO interface provides Flags that indicate the state of the FIFO. A full flag shows that there is no room left to write, an almost full flag shows that there are at least 4 words of space left. While the almost flag is not asserted accesses can be made on every clock, but after it is asserted, it is better to make one access only, then check the full flag on the next clock, before deciding if another access is possible.

FIFO clocks

The FIFO clocks are provided by the user FPGA, but are buffered externally using an LVT245 buffer that is able to provide the drive current required on these signals. To enable circuitry internal to the FPGA to be designed to use the actual clock that is applied to the FIFO, the buffered FIFO clock signals are connected to the remaining GCLK inputs. This allows DLLs to be used to provide a clock internal to the FPGA that has the same phase as that applied to the FIFOs on the carrier board.

SDRAM

The HERON-FPGA7 provides 256Mbytes of SDRAM directly connected to the I/O pins of the Virtex-II FPGA. The SDRAM is organised as a 32-bit wide memory bank of 64M locations.

In order to use the SDRAM correctly, an appropriate SDRAM controller function is required inside the FPGA. This controller must correctly initialise the SDRAM, and provide a constant refresh mechanism.

An appropriate SDRAM controller is provided in the VHDL support from HUNT ENGINEERING. This SDRAM controller can be found in the Hardware Interface Layer and must be used by all designs wishing to use the SDRAM.

Digital I/O

The HERON-FPGA7 connects 45 of the FPGAs I/O pins to connectors. This allows them to be configured as digital Inputs and Outputs as chosen by the user's FPGA program.

For the HERON-FPGA7, Vcco is always connected to 3.3V. Therefore, all FPGA I/O pins used with the Digital I/O connectors must be selected to use an IO standard that is compatible with a 3.3V Vcco.

Module and Carrier ID

The HERON specification assigns pins on the HERON module that give a HERON module access to the carrier ID of the carrier that it is plugged into, and a unique HERON slot identifier.

These IDs are used by the configuration FPGA so that the module is addressed on Heron Serial Bus (HSB) using this information. These signals are also connected to the User FPGA so a user program can use them if required.

General Purpose LEDs

There are some LEDs on the HERON-FPGA7 that are connected via a buffer to some of the FPGA I/O pins. There are five such LEDs, which can be used by the FPGA program to indicate various states of operation.

Done LEDs

There are two Done LEDs, labelled “DONE” and “CD”. They are illuminated if the relevant FPGA is not configured.

LED “DONE” is connected to the user Virtex-II FPGA.

LED “CD” is connected to the Control FPGA.

This means that the “CD” LED should flash at power on, and then go out showing that the control FPGA is ready to accept a configuration stream for the User FPGA.

After downloading a bitstream to the user User FPGA LED “DONE” should also go out.

Serial Ports

There is a connector on the HERON-FPGA7 that provides the opportunity to use a variety of serial port configurations. There is a MAX3160 “protocol converter” chip provided external to the FPGA, which can provide: -

1 channel 4 wire RS232	With RTS and CTS
2 channels of 2 wire RS232	Without RTS and CTS
1 channel of 4 wire RS485	One pair in each direction
2 channels of 2 wire RS485	Two bi-directional pairs

The components are purely voltage level converters, and any UART logic must be implemented in the FPGA.

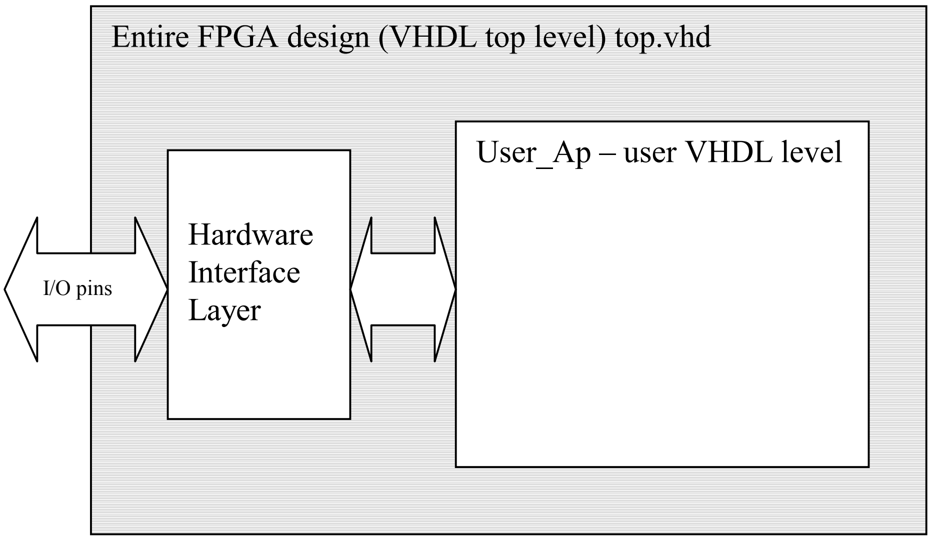
Getting Started on your FPGA Design

HUNT ENGINEERING provide a comprehensive VHDL support package for the HERON-FPGA7.

This package consists of a VHDL "top level", with corresponding user constraints file, VHDL sources and simulation files for the Hardware Interface Layer, and User VHDL files as part of the examples.

The Hardware Interface Layer correctly interfaces with the Module hardware, while the top level (top.vhd) defines all inputs and outputs from the FPGA on your module. Users should not edit these files unless a special digital I/O format is required – see the later section "Digital I/O from the FPGA".

The file user_ap.vhd is where you will make your design for the FPGA, using the simplified interfaces provided by the Hardware Interface Layer.


Entire FPGA design (VHDL top level) top.vhd
User_Ap – user VHDL level
Hardware Interface Layer
I/O pins

Organisation of VHDL support for FPGA modules

After synthesising your design, you will use the Place and Route tools from Xilinx (either as part of your ISE package, of from the Xilinx Alliance tools). These tools will use the User Constraints File (.ucf) to correctly define the correct pins and timing parameters. You will need to minimally edit this file to have the timing constraints that you need, but the file provided means you do not need to enter the pin constraints at all.

It is expected that every user will start by following the Getting Started example, Example1, which is supplied on the HUNT ENGINEERING CD. By working through the Getting Started example you will be able to see how the User FPGA is configured, how a simple example can be built, and a new bitstream generated.

In this way, you can use Example1 to check your understanding of how the module works, and you can also use the example as a sanity check that your hardware is functioning correctly.

Working through Example 1

All HERON modules that have FPGAs have an "Example1" provided for them. It is a simple example that connects data from the input FIFO interface to the output FIFO interface, and also exercises the HSB interface.

This example is fully described in the "Starting your FPGA development" tutorial on the HUNT ENGINEERING CD.

The tutorial works through running the example and then modifying and re-building it. The tutorial assumes that you are using the latest version of the ISE Foundation tool-set from Xilinx. If you are using a different version of ISE Foundation, you will simply need to convert the project as described in the application note provided by HUNT ENGINEERING titled 'Using Different Versions of ISE'. There is also an application note on the HUNT ENGINEERING CD that describes using design flows that are different from ISE, titled 'Using VHDL tools other than ISE'.

The example is quite simple but demonstrates the use of the interfaces found in the Hardware Interface Layer supplied with the module. The example is supplied in two ways. Firstly, there is a ready-to-load bitstream, supplied in the Hunt Compressed Bitstream file format, or '.hcb' format. This is the file format used by the HUNT ENGINEERING configuration tool. Secondly, there is an example1 project supplied for ISE, enabling the design to be rebuilt and a new bitstream downloaded.

The bitstream file is provided to allow you to load the example1 onto the hardware without having to re-build it. This is a useful confidence check to see if any problems you are experiencing are due to changes you have made, or the way you have built the design. If the bitstream from the CD fails to behave then the problem is more fundamental.

To make things easier, we have created the proper ISE project files for the examples.

Using these projects will allow you to run the complete design flow, from RTL-VHDL source files to the proper bitstream, ready to download on your Heron FPGA board.

No special skills are required to do this.

However, if you want to write your own code and start designing your own application, you must make sure that you have acquired the proper level of expertise in:

* VHDL language,
* Digital Design
* Xilinx FPGAs
* ISE environment and design flow

Proper training courses exist which can help you acquire quickly the required skills and techniques. Search locally for courses in your local language.

Preparing ISE

Before beginning work with Example1 you will need to make sure ISE is properly installed. In addition, you should ensure that you have downloaded the latest service pack from the Xilinx website for the version of ISE you are using.

Copying the examples from the HUNT ENGINEERING CD

On the HUNT ENGINEERING CD, under the directory "fpga" you can find directories for each module type. In the case of the HERON-FPGA7 the correct directory is "fpga7v1".

There are two ways that you can copy the files from the CD.

1) The directory tree with the VHDL sources, bit-streams etc can be copied directly from the CD to the directory of your choice. In this case there is no need to copy the .zip file, but the files will be copied to your hard drive with the same read only attribute that they have on the CD. In this case all files in the example directories need to be changed to have read/write permissions ('Example1', camera examples and SDRAM example directories). It is a good idea to leave the permissions of the 'Common' directory set to read only to prevent the accidental modification of these files.
2) To make the process more convenient we have provided the zip file, which is a zipped image of the same tree you can see on the CD. If you "unzip" this archive to a directory of your choice, you will have the file permissions already set correctly.

Opening the Example1 Project

Let us start with Example1. In the tree that you have just copied from the CD, open the `Example1` sub-directory. You should see some further sub-directories there:

* `ISE` holds the ISE project files.
* `Src` holds the application-specific Source files.
* `Sim` holds the simulation scripts for ModelSim.
* `Leo_Syn` holds the synthesis scripts for Leonardo Spectrum and Synplify users.
  You may ignore this directory in this chapter.

Open the Xilinx ISE Project Navigator. If a project pops up (from a previous run), then close it. Use `File → Open project`.

Select `Example1\ISE\XXXXX.ise` and click on the "`Open`" button.

After some internal processing, the "Sources window" of the Project Navigator will display the internal hierarchy of the Example1 project.

If you are encountering errors at this stage, you should verify that:

The example files have been correctly copied onto your hard disk, and especially the `\Common and Example1\Src` directories.

The correct version of ISE has been successfully installed. Be sure to have installed XST VHDL synthesis and the support for the Virtex2 family.

The Project's functional parameters

Double click on "`user_ap1`" in the Sources window. This opens the VHDL colour-coded text editor so that you can see the part of the project where you can enter your own design.

The first code that you will see at the beginning of this file is a VHDL Package named "`config`" which is used to configure the design files according to the application's requirements. See the next section of this manual for a description of these items.

Below the package section, you will see the User_Ap1's VHDL code.

This is where you will insert your own code when you make your own design.

We provide a system which is built in such a way that the user should not need to edit any other file than User_Ap (and the entities that this module instantiates).

In particular, the user should NOT modify the HE_* files, even when creating new designs for the FPGA.

Setting up the Configuration Package

At the top of the file USER_APx.VHD (where x indicates the example number) there are settings that you can change to affect your design (in this case the example). The idea is that settings that are often changed are found here.

1. Divide External Clock by 2

The example uses the 100Mhz oscillator that is fitted to Osc3 of the module. It generates the FIFO clock either directly from this 100Mhz, or divides it by 2 to generate a 50Mhz FIFO clock. Unfortunately the HEPC8 module carrier cannot support a clock as high as 100Mhz, and the HEPC9 carrier cannot support a clock as low as 50Mhz.

Set this parameter to "True" if you want to divide the external clock by two and use this as your main Clock.

If you are using an HEPC8 carrier board, set DIV2_FCLK to "True".

If you are using an HEPC9 carrier board, set DIV2_FCLK to "False".

2. FIFO Clocks

You must decide whether you will have a single common clock for driving the input and output FIFOs. Normally a design is simpler if the same clock is used for input and output FIFOs, but the module design allows you to use different frequencies or phases if that is more convenient for the design of your system. Whether you use a common clock or separate clocks will affect your design, but it also affects the use of clocks in the Hardware Interface Layer.

Set FCLK_G_DOMAIN to True if you have the same clock driving both FIFOs. This is the default option for the Examples. If you are unsure, select this choice.

Then, you must know whether your clocks are running slower than 60 MHz or not. This is the frequency that *you* connect to the SRC_FCLK_G in your design.

Set HIGH_FCLK_G to True if your global clock is running at 60 MHz or above. In this case an HF DLL will be used in the FIFO clocks to ensure the proper timing.

Set HIGH_FCLK_G to False otherwise. In this case the HF DLL does not work, and an LF DLL is necessary.

Set FCLK_G_DOMAIN to False is you have a different clock driving each Fifo. This option should be reserved to advanced users familiar with the management of multiple clock domains systems.

Then, you must know whether each of your clocks are running slower than 60 MHz or not. These are the frequencies that *you* connect to the SRC_FCLK_RD and SRC_FCLK_WR in your user_ap.

Set HIGH_FCLK_RD to True if your Input Fifo clock is running at 60 MHz or above, so that the HF DLL will be used for the input FIFO clock.

Set HIGH_FCLK_RD to False otherwise, so a LF DLL will be used for the input FIFO clock.

Set HIGH_FCLK_WR to True if your Output Fifo clock is running at 60 MHz or above, so that the HF DLL will be used for the output FIFO clock.

Set HIGH_FCLK_WR to False otherwise, so that a LF DLL will be used for the output FIFO clock.

The Table below summarises the available choices:

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

In the case of example1, the correct choices are:

For the HEPC8

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

For the HEPC9

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

User Timing Constraints

As with all FPGA designs it is necessary to apply some timing constraints to the design to ensure that the tools generate a design that will operate at the frequency that you require. Example1 has these defined in the '.ucf' file.

Although you can use the configuration package to set the frequency of the FIFO clocks to be 50Mhz, you can still use the stricter timing constraints needed when those clocks run at 100Mhz. So in the case of example1 you do not need to change any of the timing constraints. When you make changes to the design however you may find that you introduce more clock nets that need to be added to the ucf file. In some cases you may find that the tools are unable to achieve your desired clock frequency and then (if you are using an HEPC8) you should change the constraints to reflect you true needs.

For more details on Timing Constraints please refer to the Xilinx tools documentation.

Creating the Bitstream for Example1

Once the project has been opened as described above:

1. In the "Sources in project" window (Project Navigator), highlight (*single*-click on) the entity 'top' ("..\..\Common\top.vhd").
   This is extremely important! Otherwise, nothing will work!

2. Double-click on the "Generate Programming File" item located in the "Processes for Current Source".

   This will trigger the following activity :

   Complete synthesis, using all the project's source files.
   Since warnings are generated at this stage, you should see a yellow exclamation mark appear besides the "Synthesize" item in the Processes window.

   Complete Implementation:
   - Translation
   - Mapping
   - Placing
   - Routing

3. Creation of the bitstream.
   Note that this stage does run a DRC check, which can potentially detect anomalies created by the Place and route phase.

   When the processing ends, the proper bitstream file, with extension "`.rbt`" can be found in the project directory. This file MUST be called top.rbt. If it is not then you have synthesised a small part of your design because you did not properly highlight top.vhd in step1.

   In the "Pad Report" verify a few pins from the busses, like:
   LED(0) = AJ19, LED(1) = AJ18, LED(2) = AH19, LED(3) = AH18, etc... To do this you need to open "implement design" in the processes window, then open "Place and Route". Then double click on the pad report to open it
   If you see different assignments, STOP HERE, and verify the UCF file selected for the project.

You can download this file on your FPGA board and see how it works. See a later section of this manual.

Note that the user_ap level includes a very large counter that divides the main system clock and drives the LED #4. It is then obvious to see if the part has been properly programmed and downloaded: the LED should flash. The hardware will probably require a reset after configuration before the LED starts to flash.

If the LED does not flash, we recommend that you shut down the PC or reprogram the device using a "safe" bitstream. Otherwise, some electrical conflicts may happen (see below).

Possible causes for the device failure to operate are:

1. Wrong (or no) UCF file.
   This happens (for example) if you select the XST-version of the UCF with a Leonardo Spectrum (or Synplify) synthesis. The pin assignment for all the vectors (busses) will be ignored, and these pins will be distributed in a quasi-random fashion!

2. Wrong parameters in the CONFIG package.
3. Design Error.

If nothing seems obvious, rerun the confidence tests, then return to the original example 1.

Simulating the Complete Design

To generate the bitstream as above, you did not need to do any simulation. However, if you start modifying the provided examples and add your own code, verification can soon became an important issue.

If you need to simulate your design, you will need to install a VHDL simulator such as ModelSim (available in Xilinx Edition, Personal Edition, or Special Edition).

The example projects provided on the HUNT ENGINEERING CD include simulation files that provide a starting point for simulating your own design. If you wish to work through the simulation examples provide, please read the document 'Simulating HERON FPGA Designs'.

Making your own FPGA Design

The actual contents of the User FPGA on the HERON-FPGA7 are generated by the user. While making this development requires some knowledge of Digital Design techniques, it is made quite simple by the development environment that you use.

We recommend the Foundation ISE series software available from Xilinx, because that is what we use at HUNT ENGINEERING, and any examples and libraries we provide are tested in that environment. However there are other tools available from third parties that can also be used. The use of VHDL sources for our Hardware Interface Layer and examples means that virtually any FPGA design tool could be used. Any development tool will eventually use the Xilinx Place and Route tool, where the user constraints file that we supply will ensure that the design is correctly routed for the module. There are application notes on the HUNT ENGINEERING CD that describe how other tools might be used.

The best way to learn how to use your development tools is to follow any tutorials provided with them, or to take up a training course run by their vendors.

ALL NEW PROJECTS SHOULD START FROM ONE OF THE PROVIDED EXAMPLES – that way all of the correct settings are made, and files included. Your design should take place entirely within the User-Ap level, except in the case of needing to change the I/O formats of the Digital I/Os in which case it is necessary to minimally edit the top.vhd file – see a later section in this manual for details.

It is assumed that you are able to use your tools and follow the simplest of Digital Design techniques. HUNT ENGINEERING cannot support you in these things, but are happy to field questions specific to the hardware such as "how could I trigger my A/D from a DSP timer?" or "How can I use the FIFO interface component to do….?".

User_Ap Interface

This section describes the Interface between the User_Ap central module (or *entity* using VHDL terminology) and the external Interface hardware. This is the part where you connect your FPGA design to the resources of the module.

In other words:

1. The Clocks system
2. How your application can communicate with the external world: Digital I/Os, HSB interface, FIFOs and SDRAM.

You need to understand this interface in order to properly connect your processing logic.

The complete FPGA project consists of a Top level in which many sub-modules (*entities*) are placed (*instantiated*) and interconnected. One of these modules is User_AP: ***your*** module.

The top-level and the other modules make the system work, but you do not have to understand nor modify them in any way.

Let us see all of the Inputs/Outputs (*Ports*) of your User_Ap module:

Note that the names used for these ports are effectively "reserved" even if the user does not connect to that signal. This means the user must be careful not to re-use the same name for a signal that should not connect to these ports.

Port	Direction in User_Ap	Description
GENERAL		
RESET	In	Asynchronous module reset (active high)
CONFIG	In	System config signal (active low)
ADDR_FLAGSEL	In	Module input to select the mode of some module pins – see HERON specification
BOOTEN	Out	Module output not normally used
UMI_EN[0:3]	Out	Uncommitted Module Interconnect enables, FPGA output driven when low
UMI_IN[0:3]	In	Uncommitted Module Interconnects in
UMI_OUT[0:3]	Out	Uncommitted Module Interconnects out
MID[0:3]	In	Module ID of this module slot
CID[0:3]	In	Carrier ID of this carrier
UDPRES	Out	Optional reset to system. Drive to '1' if not used.
LED[0:4]	Out	5 x LEDs, can be used for any purpose
CLOCK SOURCES		
OSC0	In	External Clock from OSC0 oscillator
OSC3	In	External Clock from OSC3 oscillator (buffered)
SERIAL I/OS		
T1IN	Out	Data driven to RS232/485 chip
T2IN	Out	Data driven to RS232/485 chip
R1OUT	In	Data input from RS232/485 chip
R2OUT	In	Data input from RS232/485 chip
RS485_232	Out	Control signal driven to RS232/485 chip
HDPLX	Out	Control signal driven to RS232/485 chip
FAST	Out	Control signal driven to RS232/485 chip
FIFO CLOCK INTERFACE		
FCLK_RD	In	Read FIFO Clock to be used in this module (only when FCLK_G_DOMAIN = FALSE)

SRC_FCLK_RD	Out	Input FIFO Clock source for the top level (only when FCLK_G_DOMAIN = FALSE)
FCLK_WR	In	Output FIFO Clock to be used in this module (only when FCLK_G_DOMAIN = FALSE)
SRC_FCLK_WR	Out	Output FIFO clock source for the top level (only when FCLK_G_DOMAIN = FALSE)
FCLK_G	In	Common FIFO clock to be used in this module (only when FCLK_G_DOMAIN = TRUE)
SRC_FCLK_G	Out	Common FIFO clock source for the top level (only when FCLK_G_DOMAIN = TRUE)
INPUT FIFOs		
INFIFO_READ_REQ[5:0]	Out	Input FIFO Read Request
INFIFO_DVALID[5:0]	In	Input FIFO Data Valid
INFIFO_SINGLE[5:0]	In	Input FIFO Single Word Available
INFIFO_BURST[5:0]	In	Input FIFO Burst Possible
INFIFO0_D [31:0]	In	Input FIFO 0 Data
INFIFO1_D [31:0]	In	Input FIFO 1 Data
INFIFO2_D [31:0]	In	Input FIFO 2 Data
INFIFO3_D [31:0]	In	Input FIFO 3 Data
INFIFO4_D [31:0]	In	Input FIFO 4 Data
INFIFO5_D [31:0]	In	Input FIFO 5 Data
OUTPUT FIFOs		
OUTFIFO_READY[5:0]	In	Output FIFO Ready for Data
OUTFIFO_WRITE[5:0]	Out	Output FIFO Write Control
OUTFIFO_D [31:0]	Out	Data written into Output FIFO
HE_USER I/F		
MSG_CLK	Out	Clock to HE-USER interface logic.
MSG _DIN [7:0]	In	Data received from HSB
MSG _ADDR [7:0]	In	"address" received from the HSB
MSG _WEN	In	Write access from the HSB
MSG _REN	In	Read access from the HSB
MSG _DONE	In	Message was successfully transmitted (used when initiating HSB messages)
MSG _COUNT	In	Counter enable when initiating HSB messages
MSG _CLEAR	In	Asynchronous clear for address counter when

		initiating HSB messages
MSG _READY	Out	to acknowledge an access from the HSB
MSG _SEND	Out	Message send command (used when initiating HSB messages)
MSG _CE	Out	to control speed operation
MSG _DOUT [7:0]	Out	Data to be sent to HSB
MSG _SEND_ID	Out	Indicates when a byte should be replaced by Own ID (used when initiating HSB messages)
MSG _LAST_BYTE	Out	To indicate when the last byte to be sent is presented when initiating HSB messages
DIGITAL I/O		
CONN_A_EN[0:7]	Out	Digital I/O enables for connector A, FPGA output pin driven when low
CONN_A_IN[0:7]	In	Digital I/O in for connector A
CONN_A_OUT[0:7]	Out	Digital I/O out for connector A
CONN_B_EN[0:7]	Out	Digital I/O enables for connector B, FPGA output pin driven when low
CONN_B_IN[0:7]	In	Digital I/O in for connector B
CONN_B_OUT[0:7]	Out	Digital I/O out for connector B
CONN_C_EN[0:7]	Out	Digital I/O enables for connector C, FPGA output pin driven when low
CONN_C_IN[0:7]	In	Digital I/O in for connector C
CONN_C_OUT[0:7]	Out	Digital I/O out for connector C
SDRAM I/F		
SDRAM_WR_DATA[31:0]	Out	SDRAM Write Port Data
SDRAM_WR_ADDR[25:0]	Out	SDRAM Write Port Address
SDRAM_WR_AEN	Out	SDRAM Write Port Address enable
SDRAM_WR_READY	Out	SDRAM Write Port data-ready
SDRAM_WR_BURST	Out	SDRAM Write Port burst-ready
SDRAM_WR_DONE	In	SDRAM Write Port write access completed
SDRAM_RD_DATA[31:0]	In	SDRAM Read Port Data
SDRAM_RD_ADDR[25:0]	Out	SDRAM Read Port Address
SDRAM_RD_AEN	Out	SDRAM Read Port Address enable
SDRAM_RD_READY	Out	SDRAM Read Port data-ready
SDRAM_RD_BURST	Out	SDRAM Read Port burst-ready
SDRAM_RD_DONE	In	SDRAM Read Port read access completed

SDRAM_RD_DVAL	In	SDRAM Read Port data valid

Hardware Interface Layer

All of the signals listed above are connected between the 'User_Ap' interface and the pins of the FPGA via the 'Hardware Interface Layer'. The Hardware Interface Layer includes logic that correctly interfaces many different functional parts of the FPGA, from HERON-FIFO interfaces, SDRAM interface, to clock inputs and outputs, to digital I/O and serial I/O.

For a complete description of the Hardware Interface Layer (HIL), please read the document 'Using the Hardware Interface Layer in your FPGA Design'.

Important!

There are many signals that are connected between the FPGA on the HERON-FPGA7 and the HERON module connectors. Most of these signals will only be used by advanced users of the HERON-FPGA7. The FPGA pinning of these signals is defined in the UCF file, and the top.vhd has these signals commented out. Users that want to use these signals will need to uncomment them in the top.vhd and add the correct ports to the user_ap.vhd.

The FPGA sets any I/O pins of the device that are not listed in the design to have a 50-150K pull down. Most of the HERON module signals are pulled to their inactive state by 10K resistors so this 50K will have no effect. However the UDPRES signal does not, and setting this signal low will cause your whole board to be reset. Thus it is important that the UDPRES pin is driven high by the FPGA if it is not being used.

It is also advised to do the same with the LED pins to prevent them becoming illuminated erroneously.

Other Examples

There are some other examples (source and bitstream files) provided for the HERON-FPGA7 on the HUNT ENGINEERING CD. Follow "Getting Started" and then "Starting with FPGA". Choose "General FPGA Examples" and click on the directory for the fpga7. The examples are in the directories Memory_Test(ex2), SDRAM_FIFO(ex3) etc.

These examples have 'pdf' documents that describe their function.

The second example, 'Memory_Test' is useful for users who wish to validate that the SDRAM memory fitted to their module is good.

The third example, 'SDRAM_FIFO' is provided for users who need the SDRAM to behave like a big FIFO. In fact, in this example the SDRAM is used to create two independent 128Mbyte FIFOs.

Please note that as with the examples any "user" work should be done in the user_ap section i.e. do not put your own logic into the hardware interface layer, but simply include them into your own design. This enables your design to be protected from hardware specific details like pin-out, and also allows you to benefit from any new versions that HUNT Engineering might make available without having to re-work your part of the design.

How to Make a New Design

For any new design that you make, it is important that you start from the examples provided on the HUNT ENGINEERING CD.

When making a new design for the HERON-FPGA7 by starting from one of the examples you will already have a project that is correctly set up to use the supplied Hardware Interface Layer. The project will already include the correct settings and user constraints.

In fact, in all situations you should start development from one of the examples on the HUNT ENGINEERING CD, even if you intend to develop the FPGA in a way that is completely different to any of the examples.

In the case where you are to make a new design that does not match a standard example, you should start development from Example1 and add your own logic in place of the existing Example1 VHDL. By doing this, you will automatically inherit the proper ISE settings, user constraints and project structure.

When your are creating a new design from one of the standard CD examples you will need to be sure that the version of ISE design tools you are using matches the version of ISE for which the example projects were created. If you are using a different version of ISE then you must work through the HUNT ENGINEERING application note 'Using Different Versions of ISE'.

In developing new VHDL, there are proper training courses that exist to help you quickly acquire the required skills and techniques. Search locally for suitable training on these subjects. You may also consider sub-contracting part or whole of the new FPGA design.

Inserting your own Logic

When making a new design, you will create and insert your own logic inside the USER_AP module.

From here you can interface to the HERON FIFOs, the SDRAM, the HSB and the general purpose digital I/Os.

When these interfaces are simple, you may code the proper logic directly in the USER_AP module.
For more complex interfaces, you may code separate entities in separate source files, and instantiate these entities within USER_AP, as was done in the Examples.

Important: the first thing to edit in the user_ap.vhd file is the package section where generic parameters are set to match your configuration and your design.

Important: The HE_USER interface cannot be left entirely unconnected. If you have a design that does not use the features of this interface, you must be certain to connect the following. The Clock of the HE_USER must be running. The inputs MSG_SEND, MSG_SEND_ID, MSG_LAST_BYTE and MSG_CS must be connected to 0. The MSG_READY must be connected to '1'.

Top-level Fine Tuning (using other special IO pins)

The top level defines all of the I/O pins from the FPGA. Some of them are not used in the examples, but have buffers instantiated in the top.vhd. Some of those pins can have alternative signal formats that require a different Xilinx primitive to be instantiated for the buffers.

Refer to the hardware details section of this manual to learn which pins are suitable for which use.

If the buffers that are already instantiated in top.vhd (usually LVTTL) are suitable for your needs then there is no need to modify top.vhd, you can simply use the signals that are connected as ports to the user_ap file.

If you need different buffer types then it is necessary to edit top.vhd.

The method to do that is:

Make a copy of the original TOP.vhd file (from /Common) and work on this copy.

Each I/O pin has a buffer type instantiated in top.vhd.

Edit the instantiation to use the proper Xilinx Buffer primitive. You may sometimes have to insert attributes in the UCF file to qualify the IO.

Modify the User_Ap entity to make these signals visible.

Add the signals in the User_Ap instantiation port map.

User Timing Constraints

As with all FPGA designs it is necessary to apply some timing constraints to the design to ensure that the tools generate a design that will operate at the frequency that you require. These will be defined in the .ucf file.

The .ucf file provided as a template has some timing constraints already, but when you make changes to the design you may find that you introduce more clock nets that need to be added to the ucf file.

For more details on Timing Constraints please refer to the Xilinx tools documentation.

Hints for FPGA Designs

Having said that we cannot support you in making your FPGA design, we always try to make your development easy to get started, so this section outlines some things that you need to think about.

The FPGAs are basically synchronous devices, that is they register data as it passes through the device – making a processing pipeline. It is possible to apply asynchronous logic to signals but the FPGA concept assumes that logic is between registers in the pipeline.

This pipeline gives rise to two things that you need to consider. One is the maximum clock frequency that that pipeline can operate at, and the other is the number of pipeline stages in the design.

As with any component in your FPGA design, components from the HERON-FPGA7 HIL operate synchronously. That is, any control or data signal that you connect to the library component must be generated from logic that uses the same clock signal that is connected to the library component. Similarly, logic that is connected to outputs of the library component will need to be clocked by the same clock signal.

For a conventional circuit design, you would normally need to consider the signal delays from the output of one synchronous element to the input of the next element. By adding up a 'clock to output' delay from the output of the first element, adding routing delays and the 'setup to clock' delay for the input of the second element you would have a timing figure to match against the clock period. If the calculated figure is found to be too large, the circuit must be slowed down, or logic must be simplified to reduce the calculated value to one that fits the requirements.

When creating a design using the Xilinx development tools however, you only need to add a timing specification to the clock net that is used to clock both elements. This specification, which may be supplied in units of time or units of frequency will be automatically used by the tool to check that the circuit will run at the specified speed.

This leaves you free to focus on the functionality of the signals, while the Xilinx implementation tools work on achieving your specified time constraints. If at the end of your implementation the tools tell you that your timing constraints have been achieved, then the combination of all setup, hold and routing delays are such that your design will operate at the frequency you defined.

What this means for your design is that when you place a library component you need to consider whether signals are set high or low correctly on each clock edge (note, all library components are positive/rising edge clocked). What you do not need to worry about is the timing issues of each signal beyond having applied a time constraint on to the clock net that is applied to those components and the connected logic.

Use of Clocks

Because of the assumption that the design is a pipeline, the development tools will allow you to enter a specification for the clocks used in the design. This allows you to specify the frequency that you need the resulting design to operate at.

Simple designs will use only one clock, and all parts of the design will use that clock. It is usual for every "part" of the design to use the rising edge of this clock. This makes it very simple for the development tools to determine the maximum possible frequency that design could be used at. Adding too much combinatorial logic between pipeline stages will reduce the maximum possible clock rate. This gives rise to a hint – If your design will not run fast enough, add some more pipelining to areas where lots of combinatorial logic is used.

Typically development tools will give a report stating the maximum clock rate that can be used in a particular design, and will probably raise errors if that is slower than the specification that you have provided for the clock used in the design.

More complicated designs would use several clock nets, which may be related in frequency or phase, or may be completely independent. In such a design you must be careful when outputs from logic using one clock are passed to logic using a different clock. It is often useful to add a FIFO, which allows input and output clocks to be completely independent.

Possible Sources of Clocks

As you can see from the section above, an FPGA design may require one or more clock frequencies to achieve the job it needs to do. How *you* implement *your* design governs the number and frequencies of the clocks you need. The design of the module has been made to give you as much flexibility as possible, but ultimately it is up to you which will be used.

On the HERON-FPGA7 there are two possible Crystal oscillator modules, and all of the digital I/Os. Any of these can be used as clocks in the FPGA design. One technique for example could be to use one of the UMI pins as a clock input, which can be driven by the timer of a DSP module, or possibly another FPGA module driving a clock onto that UMI. This type of use though is system specific, and we cannot supply a generic example for that. The examples that we provide for the module assume a clock is fitted to the UserOSC0 socket, and use that as the only clock in the design.

Flow Control

Because the processing speed of the FPGA will almost never be the same as every other component in your system it will be necessary to use some flow control in your design. The most general way to implement this is to use Clock enables to enable the processing only when it is possible for data to flow through the "system". Otherwise some type of data storage (like FIFOs) must be implemented to ensure that data is never lost or generated erroneously.

When data is read from the HERON Input FIFOs there are FIFO flags to indicate when there is data to be read. Reading from the FIFO when the flags indicate that there is no data to read will result in false data being fed through your system. Thus your design must either a) only assert the read signal when the Empty Flag (EF) is not asserted, or b) use the EF as a clock enable for the logic in the design, thus preventing the invalid data caused by reading an empty FIFO, from being propagated through the design. The actual method used will depend on the needs of the design.

When data is written to the HERON Output FIFOs there are flags to indicate when it is possible to write new data. Writing to the FIFO when the flags indicate there is no room will result in data being lost from your system. Thus your design must not assert the Write enable signal when the Full Flag (FF) is asserted.

Pipeline Length or "latency"

The latency of your design will be determined by the length of the pipeline used in your FPGA design. The simplest way to determine this is to "count" the Flip-Flops in your data path, but whichever tools you are using might provide a more elegant way. For example the "Core Generator" will state the pipeline steps used with each core, and will even let you specify a maximum in some cases.

I/O from the FPGA

In addition to the FIFO, SDRAM and HSB interfaces of the HERON-FPGA7, there are some General purpose Digital I/O connectors and some options for serial interfaces. The use of these interfaces will be specific to a particular design, so they have not been included in the examples supplied. The pins to use are defined in the top.vhd, and the locations are already defined in the .ucf files. The choice of buffer type and the time specs of those interfaces must be taken care of by the designer.

DSP with your FPGA

The FPGA can be used to perform powerful Digital Signal Processing. It is beyond the scope of this manual and indeed not part of the normal business of HUNT ENGINEERING to teach you how to do this. There is however a simple way to build Signal Processing systems for the Xilinx FPGAs.

Xilinx supply as part of their tool-set something called a “Core Generator”. This provides a simple way of generating filters, FFTs and other “standard” signal processing elements, using a simple graphical interface. It results in a “block” that can be included in your design and connected like any other component. Typically it will have a clock input that will be subject to the pipeline speed constraints, and clock enables to allow flow control.

Using the Core Generator you can quickly build up signal processing systems, but for more complex systems you should take a course on Signal Processing theory, and perhaps attend a Xilinx course on DSP using FPGAs.

Software

The software for use with your FPGA will consist of several parts:

FPGA Development tool

The application contents of the FPGA will be generated using FPGA development tools. Xilinx ISE is the recommended tool, along with ModelSim if you require simulation.

It is possible to use alternative FPGA synthesis tools such as Leonardo Spectrum, or Synplicity, but ultimately the Place and Route stage will be performed by the same tool. Users of ISE4 and ISE5 have the Place and Route tool included, but users of the other tools require the Xilinx Alliance tool.

The FPGA design can be entered using VHDL.

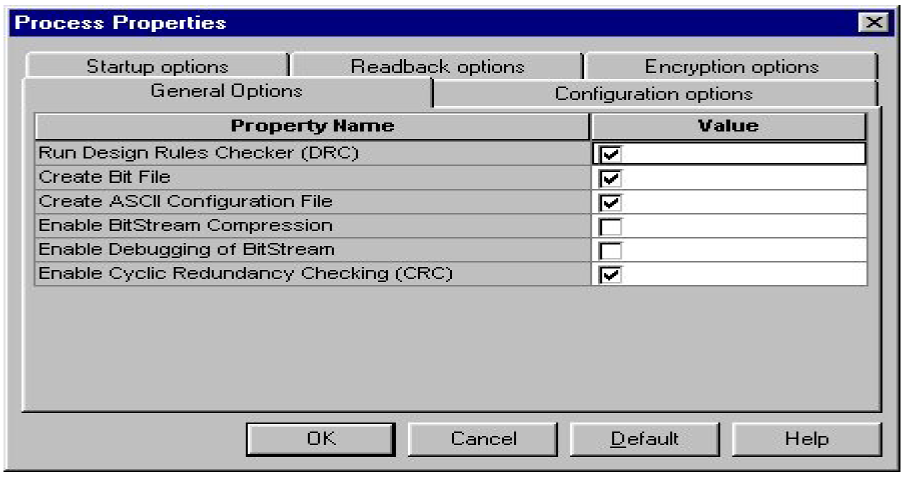
Design Files for the FPGA

The FPGA design can be downloaded onto the FPGA7 in two ways. Via the Configuration serial bus which requires a *.rbt file or *.hcb file, or via the Flash PROM on the JTAG chain which requires a *.mcs file.

Generating Design Files

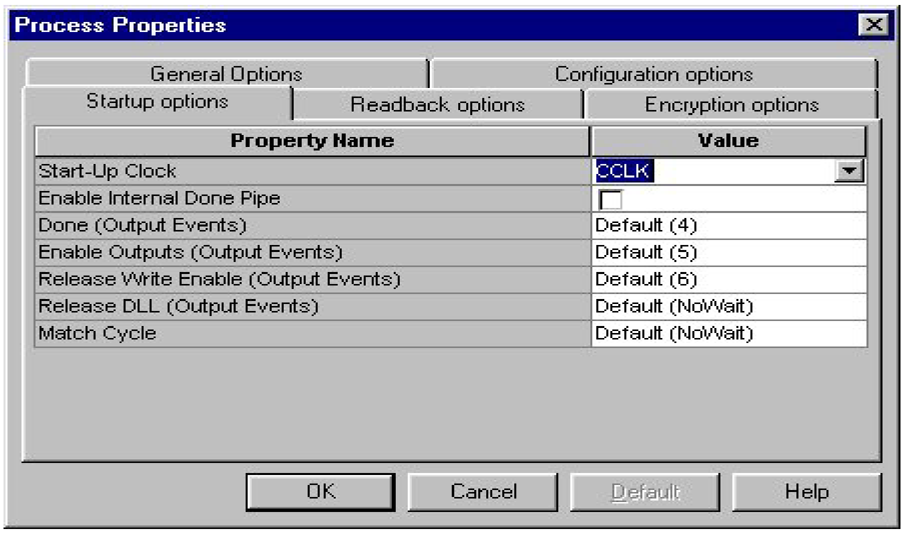
Files for HERON Utility (*.rbt)

The sections in this document titled "Creating a Project" and "Inserting your own logic" lead up to the generation of a *.rbt file which can be downloaded via the Configuration serial bus using the HERON Utility. Before generating the *.rbt file right click on "generate Program File" in the "processes for Current Source" window in ISE, select "Properties" on the menu and then select "General Options". Check that "create ASCII Configuration File " has been selected.



Property Name	Value
Run Design Rules Checker (DRC)	✓
Create Bit File	✓
Create ASCII Configuration File	✓
Enable BitStream Compression	☐
Enable Debugging of BitStream	☐
Enable Cyclic Redundancy Checking (CRC)	✓

Also from "Process Properties" select the "Start-up Options" and check that CCLK has been selected for the "Start-Up Clock". This is the default used in the example projects provided.



Property Name	Value
Start-Up Clock	CCLK
Enable Internal Done Pipe	☐
Done (Output Events)	Default (4)
Enable Outputs (Output Events)	Default (5)
Release Write Enable (Output Events)	Default (6)
Release DLL (Output Events)	Default (NoWait)
Match Cycle	Default (NoWait)

Files for PROMs (*.mcs)

The Flash PROM on the FPGA7 can be programmed, and reprogrammed via the JTAG chain using '*.mcs' files. These files are generated by the Xilinx "PROM File Formatter" after the '*.rbt' file has been generated.

Please read the document "Using iMPACT with FPGA modules" for a detailed description of how to create the correct '.mcs' file for your design.

HERON_FPGA Configuration Tool

HUNT ENGINEERING provides a tool to allow you to load the FPGAs in your system with .rbt files that you create, or copy from the CD.

For details about using this tool refer to the "Started your FPGA development" tutorial on the HUNT ENGINEERING CD.

The windows tool actually calls a program with command line parameters set according to your choices.

The program is HRN_FPGA.exe which will have been installed on your DSP machine in the directory %HEAPI_DIR%\utils.

For help using that program directly type hrm_fpga –h in a DOS box.

HUNT ENGINEERING HOST-API

The HOST-API provides a consistent software interface to all HERON Module carriers, from a number of operating systems.

While the FPGA development tools can only be run under Windows on a PC (some can be obtained in Unix versions for a workstation). It is possible to deploy your system from a number of different Host operating systems. In these cases the HOST-API and the FPGA loader tool can allow you to **use** your system, even if you cannot make your FPGA development there.

Refer to the tutorials, documentation and examples for the HOST-API on the HUNT ENGINEERING CD.

HUNT ENGINEERING HERON-API

If you have C6000 DSPs in your system, you can use HERON-API to communicate with the FPGA modules via the HERON-FIFOs. Refer to the tutorials, documentation and examples for the HERON-API on the HUNT ENGINEERING CD.

HERON Module Type

The HERON-FPGA7 module implements all four of the HERON connectors, which means it is a 32-bit module that can access all twelve of the possible HERON FIFOs.

For a complete description of the HERON interfaces, signal definitions and connector types and pin outs, refer to the separate HERON specification document. This can be found on the HUNT ENGINEERING CD, accessed through the documentation viewer, or from the HUNT ENGINEERING web site at http://www.hunteng.co.uk.

The HERON-FPGA7 does not have a processor so does not assert the "Module has processor" pin as defined in the HERON specification.

The HERON-FPGA7 does not support JTAG so does not assert the "Module has JTAG" pin as defined in the HERON specification.

The HERON-FPGA7 has a serial bus so asserts the "Module has serial bus" pin as defined in the HERON specification.

The HERON-FPGA7 has a 32 bit interface so asserts the "32/16" pin low.

Hardware Reset

Before the HERON-FPGA7 can be used, it must be reset. This reset initialises the Heron Serial Bus circuitry into a state where it can be used. Depending on the way that the user FPGA was last configured, it may also reset some functions in the user FPGA.

This reset DOES NOT cause the user FPGA to require re-configuration.

This signal is driven by the HERON module Carrier and must NOT be left unconnected, as this will cause the HERON-FPGA7 to behave erratically. It must also NEVER be driven by the user FPGA on the HERON-FPGA7.

Software Reset (via Serial Bus)

The Serial configuration bus has a reset command that is executed at the beginning of a bit stream download. This must never be confused with the system hardware reset provided on the HERON pin – it is not the same thing. The Serial bus reset simply resets the internal configuration of the FPGA but will NOT perform a hardware reset.

It cannot affect the HERON carrier board FIFOs, or any other module in the system.

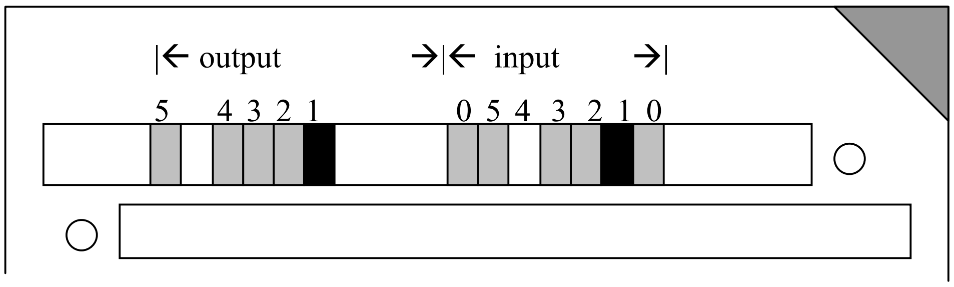
Config

There is a system wide Config signal that is open collector and hence requires a pull up to be provided by the motherboard. The HERON-FPGA7 can drive this signal active (low) or inactive if required, or can use it as an input to disable data transfer during a DSP booting phase.

If the FPGA program does nothing with this signal the signal will be pulled high by the carrier board.

Default Routing Jumpers

The default routing jumpers are provided by HERON modules. These are the longer pins on the topmost HERON connector of the module. These pins protrude above the HERON module when it is fitted to the module carrier, to which jumper links can be fitted.



These jumpers can be used to select the default routing of FIFO 0 on the Carrier card.

The exact use of these jumpers is defined by the HERON module carrier, so you should reference the user manual for the Carrier card you are using, but the numbering of these "default routing" jumpers is defined in the HERON spec, and shown above.

e.g. the jumpers as fitted in the diagram show that the default routing for both the input FIFO #0 and the output FIFO #0 is selected as 1.

HERON processor modules accept their boot stream from FIFO 0, which is why these jumpers are provided for FIFO #0 only.

The HERON-FPGA5 does not need to boot from FIFO 0, so the Default Input FIFO 0 can be set or not set as required by the user.

Physical Dimensions of the Module

A size 1 HERON module is 4.0 inches by 2.5 inches overall.

The 5mm limit on component height under the module is not violated by the HERON-FPGA7.

The maximum height of the HERON-FPGA7 above the module including mating connectors and cables is 6.5mm.

This means that the assembly of a HERON module carrier and the HERON-FPGA7 is less than the 20mm single slot spacing of PCI, cPCI and VME.

Power Requirements of the HERON-FPGA7

The HERON-FPGA7 only uses power from the 5V HERON supply. The 3.3V and 1.5V required by the FPGA are generated on board from this +5V.

The maximum rating of the HERON-FPGA7 is determined by the number of gates and the actual FPGA program loaded.

The other logic on the HERON-FPGA7 has a maximum of 150mA at 5v.

The Switch mode circuits on the HERON-FPGA7 can supply 5.0A at 1.5V and 3.3A at 3.3V. These circuits are at least 80% efficient.

FPGA Power consumption/Dissipation

The power consumption of an FPGA is governed by the number of edge transitions per second. This means it depends on not only on the configuration loaded into it and the clock frequency but also on the data being processed.

The flexibility of a Xilinx FPGA means that determining the possible power consumption is not a trivial task, an estimate can be obtained by using the Xilinx 'XPower' software package. It is still difficult to get an accurate measure because you need to describe the real data values and timings to be able to estimate correctly.

The FPGA package on the HERON-FPGA7 is an FF1152 which can dissipate 4.5 watts maximum with an ambient temperature of 50 deg C.

The HERON-FPGA7 power supplies for the FPGA are capable of delivering:-

1.5Volt	@	5Amps	7.5Watts
3.3Volts	@	3.3Amps	10.89Watts
		Total	18.39Watts

This is well above the bare package maximum power dissipation. This means that the first limit on power consumption of the HERON-FPGA7 is determined by the FPGA package.

It is always a good idea to have some airflow past the package, and normally a Fan fitted to the Case of the PC is sufficient to provide this.

The dissipation limit of the package can be increased by fitting a heatsink and possibly a fan. Depending on the performance of this heatsink your FPGA design could then reach the limit of the power supply circuits on the module. In the unlikely event that this second limit is reached it will be necessary to modify the module to use external power supplies for your system.

FIFOs

The HERON FIFO connections are shown in the table of FPGA pin-out. The timing of the signals can be found in the HERON module specification which is on the HUNT ENGINEERING website and CD.

The design implemented in the user FPGA MUST drive a constant clock onto the FIFO clock pins. The clock driven by the FPGA on "O/P FIFO clock" and "I/P FIFO clock" is buffered with an LVT245 buffer that has enough current drive for the carrier board clock signal. The actual phase of the clock on the FIFO will be the same as the inputs on the GCLK pins, so DLLs can be used to use that same clock to drive logic in the FPGA.

There may be a minimum and maximum frequency imposed by the module carrier that the module is fitted to.

However it is not necessary to look up any of this information as we supply the HIL components that will take care of the FIFO accessing for you.

SDRAM

The SDRAM on the HERON-FPGA7 is organised as a 32-bit wide memory bank, of 64M locations. In total, the memory is 256Mbytes.

The SDRAM requires a specific sequence of events to be completed after power up before the memory can be accessed. The SDRAM also requires a constant memory refresh command.

However, the SDRAM must always be used via the SDRAM controller in the Hardware Interface Layer. This controller will automatically take care of the power up control sequence and issuing of memory refresh commands.

All you will to do in your design is simply read and write data through the read and write ports of the SDRAM controller in order to read and write the external SDRAM memory.

User FPGA Clocking

As has been said elsewhere in this manual, the clocking of the FPGA can be a complex issue. The FPGA does not have such a thing as a clock pin, but rather can use an I/O pin as a clock, for almost any part of the FPGA design.

Your design will be simpler if a single clock is used, or even if there are several clocks used, but they are derived from each other. However the FPGA must drive the input and output FIFO clocks and the SDRAM clock. In each case the logic that interfaces to each must be clocked from the same clock as the interface.

When the HERON-FPGA7 is shipped there is a 100Mhz oscillator fitted to User OSC3.

The SDRAM interface should always be run at 133MHz. The SDRAM controller will automatically generate 133MHz from the OSC3 input at 100MHz. This is done using a Digital Clock Manager (DCM) to frequency synthesize 133MHz.

Different carrier boards have different requirements for the FIFO clocks. If they can all be the same then this is the simplest case, and a single clock input to the FPGA is needed.

If the rates of those clocks need to be different but can be derived from each other then the design can still be kept quite simple, but sometimes it will be necessary to have several completely independent clocks presented to the FPGA.

To allow a large amount of flexibility, the HERON-FPGA7 offers two Oscillator modules. There are also other possibilities like the Digital I/O signals or the UMI pins of the HERON module.

Clocks inputs can be used directly in your FPGA design, but Xilinx provide Digital Clock Managers (DCMs). These can be used for a number of purposes such as clock multipliers, or to align the phase of an internal clock with that of a clock signal on an I/O pin of the device. This second way is used by the Hardware Interface Layer to guarantee data access times on some of the interfaces.

GCLK0P is driven from the buffered Output FIFO clock, allowing a DCM to be used to synchronise the internal logic to that clock. This is used by the Hardware Interface Layer to manage the FIFO clocking.

GCLK6P is driven from the buffered Input FIFO clock, allowing a DCM to be used to synchronise the internal logic to that clock. This is used by the Hardware Interface Layer to manage the FIFO clocking.

GCLK7P is driven from the SDRAM clock provided to the external SDRAM devices, allowing a DCM to be used to synchronise the internal logic to that clock. This is used by the Hardware Interface Layer to manage the SDRAM clocking.

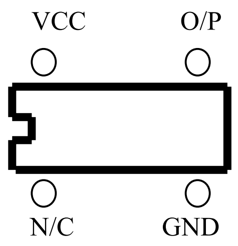
GCLK1S and GCLK5P are driven by OSC0 and OSC3 respectively. This allows the correct frequency to be driven from an oscillator, and the DCM to be used to distribute the clock with known phase.

GCLK1P and GCLK0S are driven by the CONN_A0 and CONN_A1 signals respectively. These signals are the first two I/O pins on Digital I/O Connector A. This allows the correct frequency to be supplied as two separate clocks, or one differential clock input and a DCM to be used to distribute the clock, or clocks, with known phase.

User Oscillators

The OSC0 socket on the HERON-FPGA7 accepts a plastic bodied TTL oscillator such as the SG531 type from Seiko Epson. The package is a 0.3" 8 pin DIL type body but with only 4 pins.

The socket has four pins as shown



This device is available in TTL and 3.3V versions. 5V versions can be used as the output is buffered by a 5V tolerant buffer before it is connected to the FPGA pin. The socket provides +5v supply.

OSC3 is a surface mount location that is populated at build time. It is powered from 3.3V and the output is NOT buffered before connection to the FPGA. OSC3 is fitted with a commercial grade (_+100ppm) 100Mhz oscillator at the factory.

The above part number is just an example of the oscillator type that can be fitted, and the exact specification of the oscillator should be chosen carefully for the application it will be used for. For example the tolerance, jitter and temperature dependence **might** be important considerations for some applications.

Digital I/O Connectors

The Digital I/O connectors provide the possibility to have digital I/Os connected directly to the User FPGA device.

I/O Characteristics

The characteristics of the I/O are governed by what is programmed into the FPGA. However only certain formats are possible with each voltage level on the Vcco pins of an I/O bank.

On the HERON-FPGA7 the Vcco is hardwired to 3.3V. This precludes the setting of any IO standard in the FPGA design that requires a Vcco other than 3.3V.

NOTE VIRTEX II I/Os are not 5v tolerant!

Using Digitally Controlled Impedance (DCI)

The Virtex II architecture allows the use of DCI to control the impedance of certain I/O pins. Each bank of the FPGA uses a pair of resistors connected to the VRN and VRP. FPGA that the FPGA uses to set the impedance of multiple drivers and receivers.

To use DCI the appropriate buffer must be placed in the FPGA design.

On the HERON-FPGA7 there are 50R 1% resistors connected to the VRN and VRP pins of bank 1. This means that the I/Os on Digital I/O connectors A, B and C can use DCI without changes to the board.

"DIGITAL I/O n" Connector Type

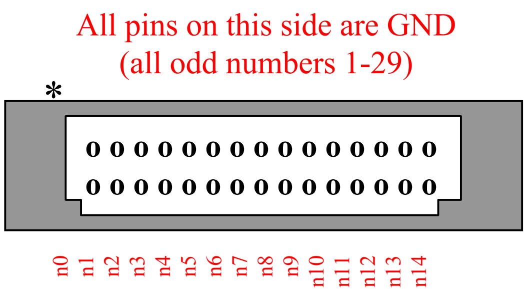
The Digital I/O connectors are surface mount 1.25mm pitch connectors. They are arranged as 15 pins in each of 2 rows. It is supplied by Hirose and its part number is DF13-30DP-1.25V(50). This connector has polarisation against incorrect insertion and mechanical retention of the mating half.

The mating connector is also supplied by Hirose and has part number DF13-30DS-1.25C which requires crimp contacts part number DF13-2630SCFA. These crimps are only available from Hirose in large quantities and require special tooling. Usually if you have explained at the time of ordering how you will be using your HERON-FPGA7 module there will be cabling supplied that suits your needs.

If your requirements change then HUNT ENGINEERING will be able to supply assemblies or component parts to meet your needs but a charge will apply.

"DIGITAL I/O n" Connector Pin-out

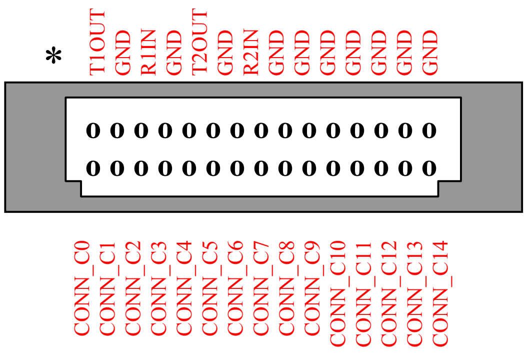
The connector sits against the top surface of the PCB, facing upwards away from the board. Connectors A and B are as shown in the diagram below. Connector C also includes RS232/RS485 signals as detailed in the following section.



Where n is the Connector letter A or B.

Connector C Pin-out

Connector C has 15 general purpose digital I/Os just like Connectors A & B. It also includes signals for implementing RS232 or RS485. The connector sits against the top surface of the PCB, facing upwards away from the board.



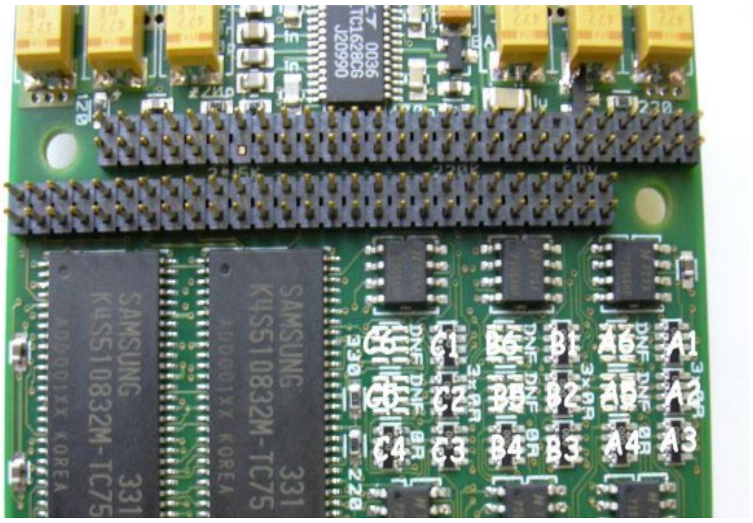
Differential Pairs

The HERON-FPGA7 uses a Virtex II device that supports differential signalling formats. The allocation of pins on the I/O connectors have been carefully chosen so that in most cases adjacent pins on an I/O connector form the positive and negative halves of a differential pair. This makes it possible to use up to 21 differential pairs.

Resistor Packs

These resistor packs are fitted on the HERON-FPGA7, and allow a series resistor in each of the I/O lines of the Digital I/O connectors and/or a differential terminating resistor between signal pairs. The standard build of the HERON-FPGA7 is to fit 0R for the serial resistor packs, making the Digital I/O NOT 5V tolerant. 100R (or other available value) can be requested if necessary for a particular application. The differential resistor packs are not fitted as standard.

The resistor packs are labelled in the photograph, and the table gives the relationship to the Digital I/O connector and signals.



FPGA7 Digital I/O Resistor Packs

SERIES RESISTORS					
Resistor Pack	Digital I/O		Resistor Pack	Digital I/O	
	Conn	Signals		Conn	Signals
A1	A	A0,A1,A2,A3	B1	B	B0,B1,B2,B3
A2	A	A4,A5,A6,A7	B2	B	B4,B5,B6,B7
A3	A	A8,A9,A10,A11	B3	B	B8,B9,B10,B11
A4	A	A12,A13,A14,--	B4	B	B12,B13,B14,--
C1	C	C0,C1,C2,C3			
C2	C	C4,C5,C6,C7			
C3	C	C8,C9,C10,C11			
C4	C	C12,C13,C14,--			

DIFFERENTIAL RESISTORS					
Resistor Pack	Digital I/O		Resistor Pack	Digital I/O	
	Conn	Signals		Conn	Signals
A6	A	A0→1,A2→3 A4→5,A6→7	B6	B	B0→1,B2→3 B4→5,B6→7
A5	A	A8→9,A10→11 A12→13,--	B5	B	B8→9,B10→11 B12→13,--
C6	C	C0→1,C2→3 C4→5,C6→7			
C5	C	C8→9,C10→11 C12→13,--			

Voltage Levels

The HERON-FPGA7 has the option of having series Resistors fitted in all I/O lines. The FPGA side of these resistors is connected to the over-voltage protection, which for the FPGA7 is set at 3.3V. Fitting 100R series resistors allows the module to accept 5V or 3.3V signals without damaging the FPGA.

The standard build of the HERON-FPGA7 is to fit 0R for these resistor packs, making the Digital I/O NOT 5V tolerant. This allows the Digital I/O connectors to support any of the voltage formats provided by the Virtex II.

100R (or other available value) can be requested if necessary for a particular application, but this precludes the use of some of the other I/O standards supported by the Virtex II.

Differential Termination

The HERON-FPGA7 has the option of having parallel Resistors fitted between differential

I/O pairs. This allows differential standards such as LVDS to be connected as inputs to the FPGA.

Where LVDS is to be used, 100R parallel resistors are required by the Virtex-II FPGA to correctly terminate an input signal. For LVDS output no termination is required.

The standard build of the HERON-FPGA7 is to not fit these resistor packs, leaving all digital I/O signals to be independent. If you wish to use parallel termination resistors please indicate this to HUNT ENGINEERING when you place your order.

ESD Protection

All of the Digital I/Os are protected against Electro Static Discharge and over voltage. The devices used are Harris SP723 parts.

This protects the inputs to IEC1004-2 level 4, and provides over voltage limiting to the range 0 to +5V.

Serial I/Os

Provided on Connector C are connections to one MAX3160 part. This part performs the necessary signal level generation for implementing RS232 or RS485. The connector also provides fifteen general purpose I/Os as discussed in the previous sections, which include a diagram of Connector C.

To use the RS232 or RS485 I/Os you will need to place an appropriate Universal Asynchronous Receiver/Transmitter (UART) in your FPGA design. All that the MAX3160 parts do is to provide the appropriate voltage levels for the signals driven on the connector.

Examples of an RS232 UART are provided on the HUNT ENGINEERING CD and website. These examples can be used as a starting point for implementing RS232 with the HERON-FPGA7.

The signals T1OUT, T2OUT, R1IN and R2IN are the RS232/485 signals from the MAX3160.

There are two optional resistors that may fitted if requested for termination of RS485 signals. At built time, these sites are left empty as standard, but if requested, may be fitted with 120 Ohm resistors. With these resistors fitted there will be 120 Ohms between T1OUT and T2OUT and between R1IN and R2IN. When using the MAX3160 for RS485 this will provide 120 Ohm termination for the transmit pair, and 120 Ohm termination for the receive pair.

Use of the MAX3160

Of course the best way to determine how to configure the MAX3160 is to look at the data sheet provided by the manufacturer Maxim Integrated Products, found at:

www.maxim-ic.com

There is one MAX3160 component fitted to the HERON-FPGA7, powered from 3.3V, with the RS232/485 signals connected to Connector C as described above. The logic side of the parts is connected directly to pins of the FPGA.

The MAX3160 has 2 digital inputs T1IN and T2IN, and 2 digital outputs R1OUT and R2OUT.

There are also three control signals also connected to the FPGA, FAST, RS485/RS232 and HDPLX.

RS232

Set FAST = high, RS485/RS232 = low and HDPLX = low.

Now the signal T1IN is driven by the FPGA, and the RS232 version of this signal appears on the T1OUT pin of the connector.

The signal T2IN is driven by the FPGA, and the RS232 version of this signal appears on the T2OUT pin of the connector.

The signal R1OUT is driven by the MAX3160, according to the RS232 version on the R1IN pin of the connector.

The signal R2OUT is driven by the MAX3160, according to the RS232 version on the R2IN pin of the connector.

Then a UART circuit configured into the FPGA can use this as either a TX/RX + CTS/RTS RS232 channel using flow control, or alternatively 2 independent TX/RX RS232 channels without flow control.

The MAX3160 can support baud rates up to 1Mbit/second in RS232 mode. If the FAST pin is set to logic low it limits the slew rate of the signals giving better immunity to errors but then the baud rate is limited to 250Kbps.

Full Duplex R485/RS422

Set FAST = high, RS485/RS232 = high and HDPLX = low.

Now the signal T1IN is driven by the FPGA, and the inverted RS422/RS485 version of this signal appears on the T1OUT pin of the connector, the non-inverted version of this signal appears on the T2OUT pin of the connector.

The signal T2IN is driven by the FPGA, and is used as a driver enable signals that is driven high to enable the outputs T1OUT and T2OUT.

The signal R1OUT is not used.

The signal R2OUT is driven by the MAX3160, according to the signal formed by the RS485/RS422 pair on the pins R1IN (inverting) and R2IN (non-inverting) pins of the connector.

Then a UART circuit configured into the FPGA can use this as a TX/RX signal pair that are driven differentially according to RS422/RS485.

The MAX3160 can support baud rates up to 10Mbit/second in R485 mode. If the FAST pin is set to logic low it limits the slew rate of the signals giving better immunity to errors but then the baud rate is limited to 250Kbps.

Half Duplex R485/RS422

Set FAST = high, RS485/RS232 = high and HDPLX = high.

Now the signal T1IN is driven by the FPGA, and the inverted RS422/RS485 version of this signal appears on the T1OUT pin of the connector, the non-inverted version of this signal appears on the T2OUT pin of the connector.

The signal T2IN is driven by the FPGA, and is used as a driver enable signals that is driven high to enable the outputs T1OUT and T2OUT.

The signal R1OUT is not used.

The signal R2OUT is driven by the MAX3160, according to the signal formed by the RS485/RS422 pair on the pins T1OUT (inverting) and T2OUT (non-inverting) pins of the connector.

Then a UART circuit configured into the FPGA can use this as a TX/RX signal pair that are driven differentially according to RS422/RS485 on a single pair formed by T1OUT and T2OUT. The direction of Transmit/Receive is now controlled using the Driver enable connected to T2IN. This type of RS485 connection is often used for multi-drop applications where all devices except one default to input, and only “respond” when polled by the “master”.

The MAX3160 can support baud rates up to 10Mbit/second in R485 mode. If the FAST pin is set to logic low it limits the slew rate of the signals giving better immunity to errors but then the baud rate is limited to 250Kbps.

ESD protection

For the RS232 and RS485 signals, the MAX3160 provides protection against wiring faults and the use of slew rate limiting is provided to minimise radiated noise. Users should ensure that cabling used in a system enables compliance with EMC directives.

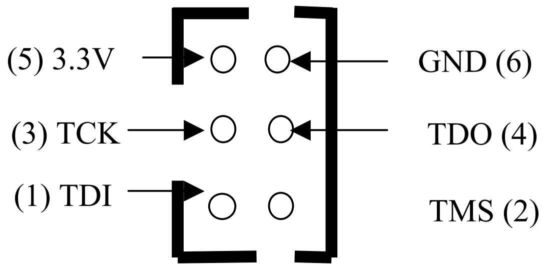
Using the JTAG Programmable Configuration PROM

The module has been built with a JTAG programmable FLASH based PROM for the user FPGA, so that your FPGA design can be programmed into this PROM. Then the user FPGA can be configured with your FPGA design on power up, and re-configuration can be forced using the send bitstream command over HSB with a zero length bitstream.

When the FPGA configuration file is downloaded to the PROM via JTAG it is possible to select "Load FPGA" on the "Program Options" then when the configuration has been downloaded to the PROM the FPGA is automatically configured. This is a useful option in systems that do not have a serial configuration bus. More information on this can be found in the "Downloading Files via JTAG" section of this document.

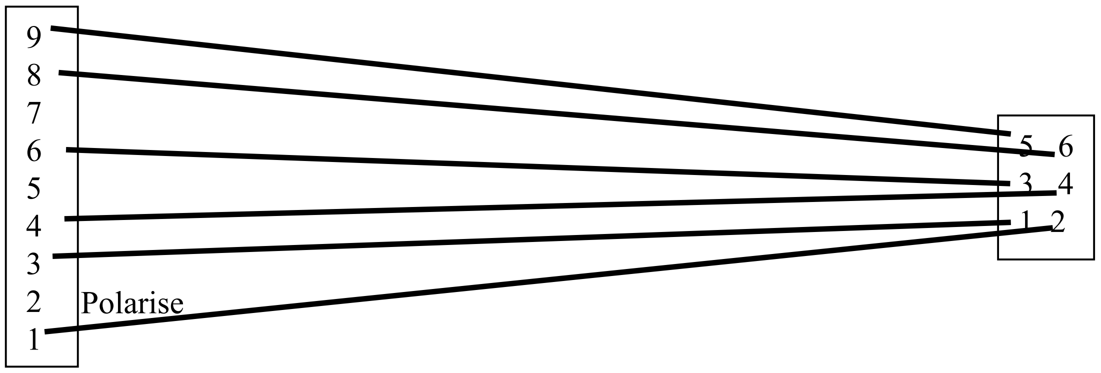
The module will have a JTAG connector fitted that is a Hirose 2mm 3x2 connector of part number DF11-6DP-DSA(01). The mating half that is required for cabling is also a Hirose part, the housing is DF11-6DS-2C and crimp part number DF11-2428CSA.

The pin-out is:



Top view of connector.

The cable supplied is as follows:



Which can be fitted to a Xilinx JTAG cable (such as Xilinx parallel cable 3 or 4) and used to connect to the PROM on the module.

Boot from PROM Jumper

The Boot from PROM (BFPROM) jumper is used to select if the FPGA boots from the ROM or via HSB.

For normal operation (HSB) the jumper should NOT be fitted.

Uncommitted Module Interconnects

There are some "Uncommitted Interconnect" signals defined by the HERON specification, which are simply connected to all modules.

These are intended to connect control signals between modules, for example a processor module can (via software) drive one of these signals with one of is timer outputs. Then if an I/O module can accept its clock input from one of these signals, it is possible to implement a system with a programmable clock. There will be other uses for these signals that are module design dependent.

The HERON-FPGA7 connects these signals to FPGA I/O pins allowing the user configuration to use these if required.

General Purpose LEDs

There are some general purpose LEDS on the HERON-FPGA7, which are driven by buffers that are external to the FPGA.

The LEDs labelled 0 to 4 are driven by the FPGA pins LED0 to LED4 respectively. The LED will illuminate when the FPGA drives the pin low.

Other HERON module signals

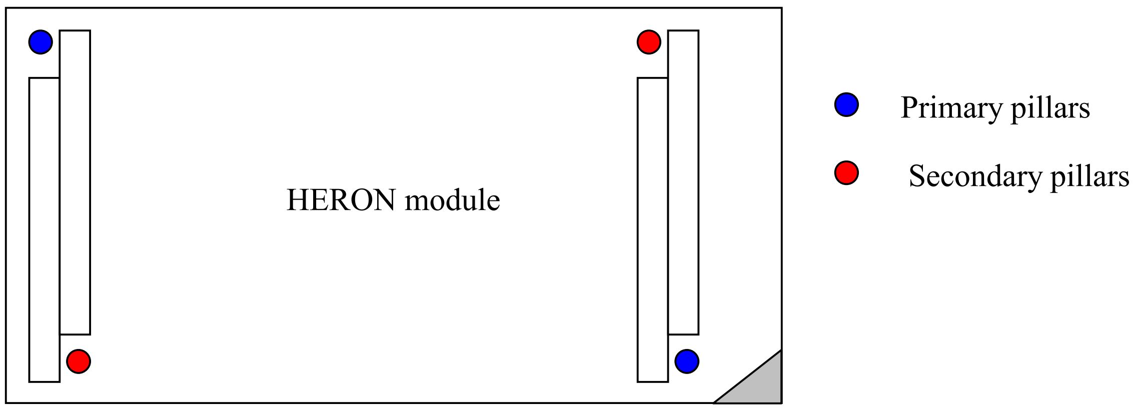
There are many signals that are connected between the FPGA on the HERON-FPGA7 and the HERON module connectors. Most of these signals will only be used by advanced users of the HERON-FPGA7. The FPGA pinning of these signals is shown in the appendix of this manual, and their use in a system is described in the HERON module specification found on the HUNT ENGINEERING CD and Web Site.

The FPGA sets any I/O pins of the device that are not listed in the design to have a 50-150K pull down. Most of the HERON module signals are pulled to their inactive state by 10K resistors so this 50K will have no effect. However the UDPRES signal does not, and setting this signal low will cause your whole board to be reset. Thus it is important that the UDPRES pin is driven high by the FPGA if it is not being used.

It is also advised to do the same with the LED pins to prevent them becoming illuminated erroneously.

Fitting HERON modules to your carrier is very simple. Ensure that the module carrier does NOT have power applied when fitting modules, and normal anti-static precautions should be followed at all times.

Each HERON slot has four positions for fixing pillars

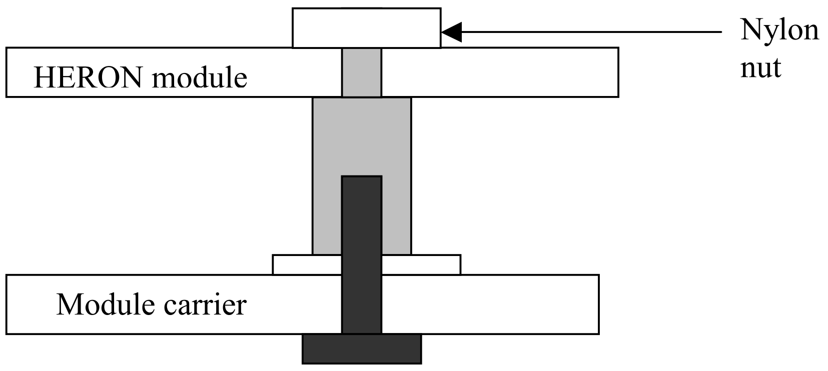


The Carrier card will probably only have spacing pillars fitted to the primary location for each HERON slot. The pillars for the secondary locations will be supplied as an accessory. The reason for this is that the legacy GDIO modules cannot be fitted if the secondary pillars are in place.

The HERON modules are asymmetric about their connectors, so if a module is fitted entirely the wrong way round, the module does not line up with the markings on the carrier card. In particular, notice the triangles on the silk screen of the HERON modules and the HERON slots of the carrier card. These should be overlaid when the module is fitted.

The HERON connectors are polarised, preventing incorrect insertion. So if more than a gentle force is needed to push the module home, check to make sure that it is correctly aligned. Take care not to apply excessive pressure to the centre of the module as this could stress the module's PCB unnecessarily.

Normally the primary fixings will be enough to retain the modules, simply fit the nylon bolts supplied in the accessory kit to the top thread of each mounting pillar.



If the environment demands, the secondary fixing pillars can be fitted to modules that allow their use.

Achievable System Throughput

In a HERON system there are many factors that can affect the achievable system throughput. It must be remembered at all times that the part of the system that has the lowest limit on bandwidth will govern the throughput of the system.

FIFO Throughput

The HERON-FPGA7 can access the HERON carriers FIFOs in 32-bit mode. It can (with the right contents) transfer one 32-bit word in and another out in the same clock cycle.

For example running at a FIFO clock speed of 50Mhz, the HERON-FPGA7 can transfer 200Mbytes/sec in at the same time as transferring 200Mbytes/sec out.

The use of faster clock speeds for the FIFOs will of course result in higher data rates.

SDRAM Throughput

The SDRAM memory interface on the HERON-FPGA7 is organised as 32-bit wide memory and is designed to operate at 133MHz. The SDRAM used is capable of transferring data on consecutive clock cycles. This means the absolute maximum data rate possible is 532 Mbytes/second.

There are two main factors that will reduce this bandwidth. These factors will vary from application to application.

Firstly, the SDRAM uses one shared data bus for read accesses and write accesses. This means at any one time only a read burst or a write burst can be in progress. For applications that are both reading and writing concurrently the total bandwidth of 532 Mbytes/sec will be shared between the read and write.

Secondly, the SDRAM cannot instantaneously present read data or accept write data. There are a small number of clock cycles either side of a data burst that are required to open the SDRAM memory row and then close the memory row. The smaller the burst, the more impact this row open and close time will have.

Troubleshooting

The following sections attempt to cover all likely problems. Please check through this section before contacting technical support.

Hardware

If the Hardware has been installed according to the Instructions there is very little that can be wrong.

- Has the "DONE" LED gone out – if not then the FPGA is not configured
- Perhaps you do not have a FIFO clock being driven from your FPGA Design
- Not driving the UDPRES signal high in your FPGA Design will result in unpredictable behaviour.

Software

As long as the software has been installed using the installation program supplied on the HUNT ENGINEERING CD, there should be little problem with the software installation.

If you have problems then return to one of the example programs supplied with the system.

CE Marking

HUNT ENGINEERING have performed testing on its products to ensure that it is possible to comply with the European CE marking directives. The HERON-FPGA7 cannot be CE marked as it is a component in a system, but as long as the following recommendations are followed, a system containing the HERON-FPGA7 could be CE marked.

The immense flexibility of the HUNT ENGINEERING product range means that individual systems should be marked in accordance with the directives after assembly.

1.The host computer or housing in which the HERON-FPGA7 is installed is properly assembled with EMC and LVD in mind and ideally should itself carry the CE mark.

2. Any cabling between boards or peripherals is either entirely inside the case of the host computer, or has been assembled and tested in accordance with the directives.

The HERON-FPGA7 digital I/Os ARE protected against Static discharge, so if the cabling does exit the case, there is suitable protection already fitted.

HUNT ENGINEERING are able to perform system integration in accordance with these directives if you are unsure of how to achieve compliance yourself.

Technical Support

Technical support for HUNT ENGINEERING products should first be obtained from the comprehensive Support section http://www.hunteng.co.uk/support/index.htm on the HUNT ENGINEERING web site. This includes FAQs, latest product, software and documentation updates etc. Or contact your local supplier - if you are unsure of details please refer to http://www.hunteng.co.uk for the list of current re-sellers.

HUNT ENGINEERING technical support can be contacted by emailing support@hunteng.co.uk, calling the direct support telephone number +44 (0)1278 760775, or by calling the general number +44 (0)1278 760188 and choosing the technical support option.

Appendix 1 – HERON Serial Bus Commands

Module Address

The HERON-FPGA7 is configured to respond to Heron Serial Bus (HSB) commands addressed to it using the combination of the Board number and slot number that the module is fitted to. In this way multiple HERON-FPGA modules can be uniquely addressed in the same system. The HSB address is a 7 bit address that is formed by the bottom three bits of the slot number (slots 1 to 4 are valid – 001,010,011, 100) with the 4 bits from the board number switch forming the top 4 bits of the seven.

e.g. on board number 1 slot 2 the address would be (board number<<3) || slot[2.0] which is 0x06.

The HERON-FPGA7 can respond to three different types of serial bus commands:

Module Enquiry

The HERON-FPGA7 can receive a message requesting its module type:

Master to FPGA module

module type query (01)-->address of requestor

It will then send a reply as follows:

FPGA module to "original master"

module query response(02)-->module address (from)-->module type (02)

-->family number(07)-->option-->String byte 0 -->String byte1….String byte26

The string is the string that Xilinx put into their bitstream files. It is always 27 bytes long, but can actually be null terminated before that. e.g. a Virtex II 3Mgate part would return 2v3000ff1152-4.

FPGA Configuration

The Configuration transaction will be:

Master to FPGA module

Configure (03)-->address of requestor--> first config byte-->

2nd config byte......last config byte

After which the FPGA module will reply:-

FPGA module to original master

configuration success (05)/configuration fail(06)-->module address

User I/O

Any further use of the HSB will be defined by the bit-stream supplied to the module.

The actual use of these messages cannot be defined here, but the format of them must be:

> Master to FPGA module
>
> user write (08) -->address of requestor -->register address byte -->value byte
>
> -->optional value byte-->optional value byte......

In this way single or multiple bytes can be written, starting from the address given.

Nothing is returned from a write request.

This will result in the 8 bit address being written into the application FPGA using an address strobe, then one or more data bytes being written to the application FPGA using a data strobe, and qualified by a write signal. It is therefore the responsibility of the application FPGA to support auto incrementing addresses if required by its function.

For a read request

> Master to FPGA module
>
> user read (09) -->address of requestor -->register address byte -->length byte

In this way single or multiple bytes can be requested, starting from the address given.

The reply will be

> FPGA module to original master
>
> user read response(10)-->module address-->data byte-->optional data byte

This will result in the 8 bit address being written into the application FPGA using an address strobe, then one or more data bytes being read from the application FPGA using a data strobe, and qualified by the absence of write signal.

Appendix 2 – FPGA Pinout for Development Tools

The following is the pin-out of the FPGA, so that the signals can be connected in the Xilinx development tools.

Data Out FIFO:

Signal name	FPGA pin	Description
DO0	AL11	HERON FIFO Data bit output
DO1	AG14	HERON FIFO Data bit output
DO2	AN8	HERON FIFO Data bit output
DO3	AJ12	HERON FIFO Data bit output
DO4	AH12	HERON FIFO Data bit output
DO5	AP6	HERON FIFO Data bit output
DO6	AK11	HERON FIFO Data bit output
DO7	AF13	HERON FIFO Data bit output
DO8	AL9	HERON FIFO Data bit output
DO9	AN6	HERON FIFO Data bit output
DO10	AG12	HERON FIFO Data bit output
DO11	AP4	HERON FIFO Data bit output
DO12	AL8	HERON FIFO Data bit output
DO13	AE13	HERON FIFO Data bit output
DO14	AN4	HERON FIFO Data bit output
DO15	AH10	HERON FIFO Data bit output
DO16	AM11	HERON FIFO Data bit output
DO17	AG13	HERON FIFO Data bit output
DO18	AP9	HERON FIFO Data bit output
DO19	AJ11	HERON FIFO Data bit output
DO20	AH13	HERON FIFO Data bit output
DO21	AP7	HERON FIFO Data bit output
DO22	AK10	HERON FIFO Data bit output
DO23	AF12	HERON FIFO Data bit output
DO24	AL10	HERON FIFO Data bit output
DO25	AN7	HERON FIFO Data bit output
DO26	AG11	HERON FIFO Data bit output
DO27	AP5	HERON FIFO Data bit output
DO28	AM9	HERON FIFO Data bit output
DO29	AE12	HERON FIFO Data bit output
DO30	AN5	HERON FIFO Data bit output
DO31	AH9	HERON FIFO Data bit output
FCLK0	AM17	O/P FIFO Clock output to input of buffer. Use to drive correct frequency.
DOCLK/GCLKx	AK17	O/P FIFO Clock output of buffer - use with DLL for internal logic
DOF0CONT0	AJ14	O/P FIFO #0 Write enable (active high) output
DOF0CONT1	AM12	O/P FIFO #0 Full Flag (active low) input
DOF0CONT2	AN13	O/P FIFO #0 Almost full flag (active low) input
DOF1CONT0	AJ13	O/P FIFO #1 Write enable (active high) output
DOF1CONT1	AM13	O/P FIFO #1 Full Flag (active low) input
DOF1CONT2	AN14	O/P FIFO #1 Almost full flag (active low) input
DOF2CONT0	AE15	O/P FIFO #2 Write enable (active high) output
DOF2CONT1	AF15	O/P FIFO #2 Full Flag (active low) input

Signal name	FPGA pin	Description
DOF2CONT2	AG16	O/P FIFO #2 Almost full flag (active low) input
DOF3CONT0	AE14	O/P FIFO #3 Write enable (active high) output
DOF3CONT1	AF14	O/P FIFO #3 Full Flag (active low) input
DOF3CONT2	AG15	O/P FIFO #3 Almost full flag (active low) input
DOF4CONT0	AN11	O/P FIFO #4 Write enable (active high) output
DOF4CONT1	AL12	O/P FIFO #4 Full Flag (active low) input
DOF4CONT2	AP11	O/P FIFO #4 Almost full flag (active low) input
DOF5CONT0	AN12	O/P FIFO #5 Write enable (active high) output
DOF5CONT1	AL13	O/P FIFO #5 Full Flag (active low) input
DOF5CONT2	AP12	O/P FIFO #5 Almost full flag (active low) input

These FIFO signals should be used via the library symbol supplied by HUNT ENGINEERING and are only mentioned here for completeness.

Data In FIFO:

Signal name	FPGA pin	Description
DI0	AK27	HERON FIFO Data bit input
DI1	AN30	HERON FIFO Data bit input
DI2	AJ25	HERON FIFO Data bit input
DI3	AL26	HERON FIFO Data bit input
DI4	AM26	HERON FIFO Data bit input
DI5	AF23	HERON FIFO Data bit input
DI6	AH25	HERON FIFO Data bit input
DI7	AP30	HERON FIFO Data bit input
DI8	AH22	HERON FIFO Data bit input
DI9	AK25	HERON FIFO Data bit input
DI10	AN28	HERON FIFO Data bit input
DI11	AG22	HERON FIFO Data bit input
DI12	AP28	HERON FIFO Data bit input
DI13	AP26	HERON FIFO Data bit input
DI14	AE21	HERON FIFO Data bit input
DI15	AL24	HERON FIFO Data bit input
DI16	AK26	HERON FIFO Data bit input
DI17	AN31	HERON FIFO Data bit input
DI18	AH26	HERON FIFO Data bit input
DI19	AL27	HERON FIFO Data bit input
DI20	AM27	HERON FIFO Data bit input
DI21	AF22	HERON FIFO Data bit input
DI22	AH24	HERON FIFO Data bit input
DI23	AP31	HERON FIFO Data bit input
DI24	AH23	HERON FIFO Data bit input
DI25	AK24	HERON FIFO Data bit input
DI26	AN29	HERON FIFO Data bit input
DI27	AG21	HERON FIFO Data bit input
DI28	AP29	HERON FIFO Data bit input
DI29	AN27	HERON FIFO Data bit input
DI30	AE20	HERON FIFO Data bit input
DI31	AL25	HERON FIFO Data bit input
F1CLK	AL18	I/P FIFO Clock output to input of buffer. Use to drive correct frequency.
DICLK/GCLKx	AK19	I/P FIFO Clock output of buffer – use with DLL for internal logic
DIF0CONT0	AL23	I/P FIFO #0 Read enable (active high) output
DIF0CONT1	AJ21	I/P FIFO #0 output enable (active low) output

DIF0CONT2	AJ20	I/P FIFO #0 Empty Flag (active low) input
DIF0CONT3	AK22	I/P FIFO #0 Almost Empty flag (active low) input
DIF1CONT0	AL22	I/P FIFO #1 Read enable (active high) output
DIF1CONT1	AJ22	I/P FIFO #1 output enable (active low) output
DIF1CONT2	AH20	I/P FIFO #1 Empty Flag (active low) input
DIF1CONT3	AK21	I/P FIFO #1 Almost Empty flag (active low) input
DIF2CONT0	AF20	I/P FIFO #2 Read enable (active high) output
DIF2CONT1	AJ24	I/P FIFO #2 output enable (active low) output
DIF2CONT2	AG19	I/P FIFO #2 Empty Flag (active low) input
DIF2CONT3	AD18	I/P FIFO #2 Almost Empty flag (active low) input
DIF3CONT0	AF21	I/P FIFO #3 Read enable (active high) output
DIF3CONT1	AJ23	I/P FIFO #3 output enable (active low) output
DIF3CONT2	AG20	I/P FIFO #3 Empty Flag (active low) input
DIF3CONT3	AD19	I/P FIFO #3 Almost Empty flag (active low) input
DIF4CONT0	AM24	I/P FIFO #4 Read enable (active high) output
DIF4CONT1	AN24	I/P FIFO #4 output enable (active low) output
DIF4CONT2	AP24	I/P FIFO #4 Empty Flag (active low) input
DIF4CONT3	AN22	I/P FIFO #4 Almost Empty flag (active low) input
DIF5CONT0	AM23	I/P FIFO #5 Read enable (active high) output
DIF5CONT1	AN23	I/P FIFO #5 output enable (active low) output
DIF5CONT2	AP23	I/P FIFO #5 Empty Flag (active low) input
DIF5CONT3	AN21	I/P FIFO #5 Almost Empty flag (active low) input

These FIFO signals should be used via the library symbol supplied by HUNT ENGINEERING and are only mentioned here for completeness.

SDRAM:

Signal name	FPGA pin	Description
SDRAM_A0	F21	SDRAM Address bit output
SDRAM_A1	F22	SDRAM Address bit output
SDRAM_A2	G23	SDRAM Address bit output
SDRAM_A3	G22	SDRAM Address bit output
SDRAM_A4	B24	SDRAM Address bit output
SDRAM_A5	B23	SDRAM Address bit output
SDRAM_A6	D23	SDRAM Address bit output
SDRAM_A7	D22	SDRAM Address bit output
SDRAM_A8	D24	SDRAM Address bit output
SDRAM_A9	C24	SDRAM Address bit output
SDRAM_A10	B27	SDRAM Address bit output
SDRAM_A11	A26	SDRAM Address bit output
SDRAM_A12	A29	SDRAM Address bit output
SDRAM_DQ0	J22	SDRAM Data bit in/out
SDRAM_DQ1	B28	SDRAM Data bit in/out
SDRAM_DQ2	F23	SDRAM Data bit in/out
SDRAM_DQ3	H23	SDRAM Data bit in/out
SDRAM_DQ4	D25	SDRAM Data bit in/out
SDRAM_DQ5	E25	SDRAM Data bit in/out
SDRAM_DQ6	G24	SDRAM Data bit in/out
SDRAM_DQ7	A30	SDRAM Data bit in/out
SDRAM_DQ8	B29	SDRAM Data bit in/out
SDRAM_DQ9	F24	SDRAM Data bit in/out
SDRAM_DQ10	H22	SDRAM Data bit in/out
SDRAM_DQ11	D26	SDRAM Data bit in/out
SDRAM_DQ12	E24	SDRAM Data bit in/out
SDRAM_DQ13	G25	SDRAM Data bit in/out

SDRAM_DQ14	A31	SDRAM Data bit in/out
SDRAM_DQ15	D27	SDRAM Data bit in/out
SDRAM_DQ16	C26	SDRAM Data bit in/out
SDRAM_DQ17	K23	SDRAM Data bit in/out
SDRAM_DQ18	B30	SDRAM Data bit in/out
SDRAM_DQ19	C27	SDRAM Data bit in/out
SDRAM_DQ20	J24	SDRAM Data bit in/out
SDRAM_DQ21	B32	SDRAM Data bit in/out
SDRAM_DQ22	E26	SDRAM Data bit in/out
SDRAM_DQ23	H25	SDRAM Data bit in/out
SDRAM_DQ24	K22	SDRAM Data bit in/out
SDRAM_DQ25	B31	SDRAM Data bit in/out
SDRAM_DQ26	C28	SDRAM Data bit in/out
SDRAM_DQ27	J23	SDRAM Data bit in/out
SDRAM_DQ28	C33	SDRAM Data bit in/out
SDRAM_DQ29	F27	SDRAM Data bit in/out
SDRAM_DQ30	H24	SDRAM Data bit in/out
SDRAM_DQ31	F26	SDRAM Data bit in/out
SDRAM_CLK_ SRC0	A23	SDRAM Clock source 0, output. Use to drive correct SDRAM frequency.
SDRAM_CLK_ SRC1	A28	SDRAM Clock source 1, output. Use to drive correct SDRAM frequency.
SDRAM_CLK	K18	SDRAM Clock input – use with DLL for internal logic
SDRAM_DQM	J21	SDRAM Data mask control output
SDRAM_BA0	K21	SDRAM Bank address bit output
SDRAM_BA1	A24	SDRAM Bank address bit output
SDRAM_RAS	C23	SDRAM Row address strobe output
SDRAM_CAS	E21	SDRAM Column address strobe output
SDRAM_WE	E22	SDRAM Write enable output
SDRAM_CS0	H21	SDRAM Chip select output
SDRAM_CS1	H20	SDRAM Chip select output
SDRAM_CKE0	K20	SDRAM Clock enable output
SDRAM_CKE1	C22	SDRAM Clock enable output

These SDRAM signals should be used via the library symbol supplied by HUNT ENGINEERING and are only mentioned here for completeness.

IO on Connectors

Signal name	FPGA pin	Description
A0 (L95N_1)	H17	General purpose I/O with selectable I/O format
A1 (L95P_1)	H16	General purpose I/O with selectable I/O format
A2 (L04N_1)	D8	General purpose I/O with selectable I/O format
A3 (L04P_1)	E7	General purpose I/O with selectable I/O format
A4 (L05N_1)	H11	General purpose I/O with selectable I/O format
A5 (L05P_1)	J10	General purpose I/O with selectable I/O format
A6 (L06N_1)	D6	General purpose I/O with selectable I/O format
A7 (L06P_1)	C6	General purpose I/O with selectable I/O format
A8 (L19N_1)	B4	General purpose I/O with selectable I/O format
A9 (L19P_1)	B5	General purpose I/O with selectable I/O format
A10 (L20N_1)	J12	General purpose I/O with selectable I/O format
A11 (L20P_1)	J11	General purpose I/O with selectable I/O format
A12 (L21N_1)	F10	General purpose I/O with selectable I/O format
A13 (L21P_1)	G9	General purpose I/O with selectable I/O format
A14	A5	General purpose I/O with selectable I/O format
B0 (L27N_1)	E11	General purpose I/O with selectable I/O format
B1 (L27P_1)	E10	General purpose I/O with selectable I/O format
B2 (L28N_1)	C7	General purpose I/O with selectable I/O format
B3 (L28P_1)	C8	General purpose I/O with selectable I/O format
B4 (L29N_1)	H13	General purpose I/O with selectable I/O format

B5 (L29P_1)	H12	General purpose I/O with selectable I/O format
B6 (L30N_1)	D9	General purpose I/O with selectable I/O format
B7 (L30P_1)	C9	General purpose I/O with selectable I/O format
B8 (L49N_1)	A6	General purpose I/O with selectable I/O format
B9 (L49P_1)	A7	General purpose I/O with selectable I/O format
B10(L50N_1)	K14	General purpose I/O with selectable I/O format
B11(L50P_1)	K13	General purpose I/O with selectable I/O format
B12(L51N_1)	B8	General purpose I/O with selectable I/O format
B13(L51P_1)	A9	General purpose I/O with selectable I/O format
B14	B10	General purpose I/O with selectable I/O format
C0 (L69N_1)	F14	General purpose I/O with selectable I/O format
C1 (L69P_1)	F13	General purpose I/O with selectable I/O format
C2 (L70N_1)	D12	General purpose I/O with selectable I/O format
C3 (L70P_1)	D13	General purpose I/O with selectable I/O format
C4 (L71N_1)	J15	General purpose I/O with selectable I/O format
C5 (L71P_1)	J14	General purpose I/O with selectable I/O format
C6 (L72N_1)	E13	General purpose I/O with selectable I/O format
C7 (L72P_1)	E14	General purpose I/O with selectable I/O format
C8 (L73N_1)	A11	General purpose I/O with selectable I/O format
C9 (L73P_1)	A12	General purpose I/O with selectable I/O format
C10(L74N_1)	H15	General purpose I/O with selectable I/O format
C11(L74P_1)	H14	General purpose I/O with selectable I/O format
C12(L75N_1)	F15	General purpose I/O with selectable I/O format
C13(L75P_1)	G15	General purpose I/O with selectable I/O format
C14	B14	General purpose I/O with selectable I/O format

Clocks

Signal name	FPGA pin	Description
OSC0	AK16	User Oscillator input from socketed Xtal Osc
OSC3	E19	User Oscillator input from surface mount Xtal Osc
		Repeated from above
FCLK0	AM17	O/P FIFO Clock output to input of buffer. Use to drive correct frequency.
DOCLK/GCLKx	AK17	O/P FIFO Clock output of buffer – use with DLL for internal logic
F1CLK	AL18	I/P FIFO Clock output to input of buffer. Use to drive correct frequency.
DICLK/GCLKx	AK19	I/P FIFO Clock output of buffer – use with DLL for internal logic
SDRAM_CLK_ SRC0	A23	SDRAM Clock source 0, output. Use to drive correct SDRAM frequency.
SDRAM_CLK_ SRC1	A28	SDRAM Clock source 1, output. Use to drive correct SDRAM frequency.
SDRAM_CLK	K18	SDRAM Clock input – use with DLL for internal logic

LEDs

Signal name	FPGA pin	Description
LED0	AJ19	General Purpose LED
LED1	AJ18	General Purpose LED
LED2	AH19	General Purpose LED
LED3	AH18	General Purpose LED
LED4	AM19	General Purpose LED

Control connections to HERON connectors

Signal name	FPGA pin	Description
ADDR/FLAGSEL	AN3	Selector driven by Carrier board to determine the function of the almost flags
BOOTEN	AM7	This module can drive this low if it wishes the carrier to operate regardless of the Config signal
UDPRES	AM8	This module can drive this to reset the carrier YOU MUST DRIVE THIS SIGNAL HIGH IF NOT USING IT!
RESET	AM16	Reset input from Carrier card
CONFIG	AM14	Open collector signal

Carrier and Module ID

Signal name	FPGA pin	Description
CID0	AE11	Carrier ID driven by the carrier board
CID1	AJ5	Carrier ID driven by the carrier board
CID2	AH6	Carrier ID driven by the carrier board
CID3	AL2	Carrier ID driven by the carrier board
MID0	AD10	Module ID driven by the carrier board
MID1	AE10	Module ID driven by the carrier board
MID2	AK4	Module ID driven by the carrier board
MID3	AJ4	Module ID driven by the carrier board

Uncommitted Module Interconnects

Signal name	FPGA pin	Description
UMI0	AM15	Uncommitted Module interconnect
UMI1	AJ16	Uncommitted Module interconnect
UMI2	AJ17	Uncommitted Module interconnect
UMI3	AL17	Uncommitted Module interconnect

Serial options

Signal name	FPGA pin	Description
T1IN_A	E3	Serial Data Bit output from FPGA (connected to MAX3160)
T2IN_A	G5	Serial Data Bit output from FPGA (connected to MAX3160)
R1OUT_A	F5	Serial Data Bit input to FPGA (connected to MAX3160)
R2OUT_A	K10	Serial Data Bit input to FPGA (connected to MAX3160)
RS485/232_A	E2	Control input to MAX3160
HDPLX_A	D2	Control input to MAX3160
FAST_A	K11	Control input to MAX3160

User interface between HSB device and FPGA (N.B. Some signals also used during configuration of FPGA.

Signal name	FPGA pin	Description
Data0	AG10	Data Bit for read/write via HSB
Data1	AH11	Data Bit for read/write via HSB
Data2	AK7	Data Bit for read/write via HSB
Data3	AK8	Data Bit for read/write via HSB
Data4	AK28	Data Bit for read/write via HSB
Data5	AL29	Data Bit for read/write via HSB
Data6	AG24	Data Bit for read/write via HSB
Data7	AG25	Data Bit for read/write via HSB
Address Strobe	AL5	Rising edge strobes the 8 bit address from the Data pins
Data Strobe	AM31	Low to show an access in progress
R/notW	AL30	State of this pin when Data Strobe is active defines whether the operation is read(High) or write (low)
Ready	AM4	The FPGA asserts this signal low when either :- a) during a Write to the FPGA the data has been latched b) during a read from the FPGA the data has been driven

These signals should be used via the library symbol HE_USER supplied by HUNT ENGINEERING and are only mentioned here for completeness.