



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
www.hunteng.co.uk
www.hunt-dsp.com



Using the Hardware Interface Layer V2.0 in your FPGA Design

v 1.3 R.Williams 11-10-04

For modules in the HERON-FPGA and HERON-IO families, HUNT ENGINEERING provide a comprehensive VHDL support package. The VHDL package consists of a “top level”, with corresponding user constraints file, VHDL sources and simulation files for the Hardware Interface Layer, and User VHDL files as part of many 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.

The file user_ap.vhd is where the user implements the design for the FPGA, using the simplified interfaces provided by the Hardware Interface layer.

This document discusses the key interfaces that are provided in the Hardware Interface Layer, and how this layer should be used by the logic that is created in the User_Ap part of the FPGA design.

History

Rev 1.0	First written
Rev 1.1	Added SDRAM and A-to-D interface sections
Rev 1.2	Added SPI interface section
Rev 1.3	Added DDR SDRAM interface section

The Hardware Interface Layer

There are several common interfaces that are included in the Hardware Interface Layer for all HERON-FPGA and HERON-IO modules. These interfaces are described below, and include the system 'Reset', clock sources, HERON-FIFO read interface, HERON-FIFO write interface and HSB message interface.

For some modules there are module specific interfaces that do not appear on all module types. A good example of this is the A/D and D/A interfaces of many of the HERON-IO modules. These module specific interfaces are introduced in this document, but for a full description you should also refer to the user manuals and examples provided with those modules.

General

The Reset into the FPGA can be used by the Users logic, but remember this reset does not "clear" the contents of the FPGA.

The OSCx inputs are used to provide oscillator inputs to the FPGA. On all boards, there is usually one dedicated OSC input, which is connected to a 100MHz crystal oscillator soldered to the module. This OSC input will be used by the example projects for generating any required internal clock signals.

UDPRES can be driven to reset the carrier board and all other modules on it. Use with caution! Normally this will be driven inactive (high).

LED signals can be driven by your design, low to make the LED switch on. This can be useful for seeing the status of your internal logic.

Note that Digital I/Os, where they exist, and UMIs are connected using LVTTTL buffers in top.vhd. If other formats are required for these signals top.vhd must be modified to provide these buffers, and then the port list of user_ap may need to change.

Clock Sources

Here are the various possibilities of clock inputs to your design. Normally the only one that has a clock signal is OSC3 – used in the examples.

Use the others if you are connecting other external oscillators, or clock I/O via cables.

Serial I/Os

These are the signals connected to the external RS485/232 buffers and to an external differential LVPECL buffer, if available on the module type you are using.

FIFO Clock Interface

This interface is the interface that is configured by the settings in the Config module. Depending on the settings in the CONFIG module, you drive the Global SRC_FCLK or the separate clocks with the FIFO clock frequency that you need, and the output Global or separate FCLK signals provide the clock for your internal logic that is correctly aligned with the clock on the HERON FIFO.

Input FIFOs

This interface is where you read data from the Input FIFOs of the HERON interface. The Six FIFO Read Interface component, HE_RD_6F is provided for reading from all six FIFOs concurrently.

The interface provides six read request signals INFIFO_READ_REQ[5:0]. These signals are used to indicate that data is wanted from each of the external FIFOs. The read request signals must be used in combination with the data valid signals INFIFO_DVALID[5:0].

Data read from each of the external FIFOs is presented on one of the output data busses, INFIFO0_D[31:0] for FIFO 0, INFIFO1_D[31:0] for FIFO1, INFIFO2_D[31:0] for FIFO2, INFIFO3_D[31:0] for FIFO3, INFIFO4_D[31:0] for FIFO4 and INFIFO5_D[31:0] for FIFO 5.

When a read request signal is asserted (set high, logic 1) this indicates that data is requested from the corresponding FIFO interface. If there is any data to be read from the external FIFO, that data is read at the earliest possible time, and when that data is presented on the INFIFOn_D data bus, the corresponding data valid signal is asserted (set high, logic 1).

The outputs INFIFO_SINGLE[5:0] indicate whether there is any data to be read from the external FIFOs. If INFIFO_SINGLE_n is asserted (set high, logic 1) this indicates that there is at least one word to be read from external FIFO _n.

The outputs INFIFO_BURST[5:0] indicate whether there are multiple data elements to be read from the external FIFOs. If INFIFO_BURST_n is asserted (set high, logic 1) this indicates that there are at least four words to be read from external FIFO _n.

Input FIFO Latency

For data read through the external HERON FIFO input data bus there is one clock cycle of delay introduced by a single 32-bit input register. This input register is required to initially register the external data into the part. With this input register, it is possible to transfer one word of data on every clock cycle, where that data is one clock delayed from what was read from the external FIFO.

The exception to this one clock cycle data delay is where the external FIFO has been empty and the first word of data arrives in the external FIFO. In this case where the FIFO is not being accessed, data is presented on the fourth clock edge following the clock cycle where the external FIFO indicates there is data available for reading.

Output FIFOs

This interface is where you write data to the Output FIFOs of the HERON interface. The Six FIFO Write Interface component, HE_WR_6F is provided for writing to all six FIFOs concurrently.

The interface provides six write enable signals OUTFIFO_WRITE[5:0]. These signals are used to control whether data is written to each of the external FIFOs. There is one control signal for each of the six external FIFOs. OUTFIFO_WRITE₀ will control writing to HERON FIFO 0, OUTFIFO_WRITE₁ will control writing to HERON FIFO1, and so on. To assert a write control signal, it must be set high (logic 1). To de-assert a write control signal it must be set low (logic 0).

During any one clock period, only one of the six write control signals can be asserted.

To write to a different FIFO, the first write control signal must be de-asserted at the same time, or before the second write control signal is asserted.

Data to be output to any of the external FIFOs must be presented on the 32-bit input bus OUTFIFO_D[31:0]. For a clock cycle where a write access is being performed (OUTFIFO_WRITE_n = 1) the corresponding data must be presented on the OUTFIFO_D input.

The interface provides six 'ready' signals OUTFIFO_READY[5:0], one for each of the external FIFOs. When one of the OUTFIFO_READY signals is high (asserted), this indicates that data can be written to the external FIFO. At the point at which the OUTFIFO_READY signal is first sampled low (deasserted) the interface can accept no more than one more data access before writing is stopped to that FIFO. Writing to that FIFO may resume when the OUTFIFO_READY signal returns high.

Output FIFO Latency

There is a single 32-bit output register used to buffer internal data on the OUTFIFO_D bus to the external HERON-FIFO data bus. With this output register, there is one clock cycle of delay from when the data is written to the six FIFO interface component, to when it is written into the external HERON FIFO.

HERON-FIFO Interface Examples

There are two generic examples provided on the HUNT ENGINEERING CD that demonstrate how to correctly use the HERON Input FIFO interface and the HERON Output FIFO interface. These examples are not included for each and every module type as the issues of interfacing to the HERON-FIFOs is the same regardless of module type.

It is recommended that you work through these examples in order to gain an understanding of the different data transfer possibilities presented by these interfaces.

HE USER Message Interface

This is actually an interface to the control FPGA on the module. The interface is used to configure the User FPGA over the HSB. Once configured this interface allows your FPGA design to interact with the HSB, via that control FPGA.

There are two functional parts to the HE_USER component. The first part is used to receive incoming messages, and the second part is used to send messages to another module or to devices on the carrier card. Each part can be used on its own, or both parts can be used together. If either function is not being used, the unused inputs must be driven low except the MSG_READY input which, if unused, must be driven high.

The MSG_CLOCK input must be connected to the system clock used by your application. The Low-to-High transition on this clock will cause changes in state of the interface. Both the receive message and send message functions require a free running clock to be supplied on this input. If neither the receive message or send message functions are to be used, a free running clock must still be connected to this component.

Slave I/F

The receive message function of the HE_USER component is used to respond to accesses made to read or write locations inside the FPGA. When a 'User Write' access is made using this interface, the MSG_WEN signal will become asserted (set high), and held asserted until the access is acknowledged. Once the access has been acknowledged, the MSG_WEN signal is deasserted (set low).

Similarly when a 'User Read' access is made using this interface, the MSG_REN signal will become asserted (set high), and held asserted until the access is acknowledged. Once the access has been acknowledged, the MSG_REN signal is deasserted (set low).

To acknowledge the write access or read access, the MSG_READY input must be driven high. (While the interface is inactive, the MSG_READY signal should be driven low.)

If the receive message function is not being used, simply leave the MSG_WEN and MSG_REN

outputs unconnected, and connect the MSG_READY input to VCC.

When receiving a message, if the access is a write, the address given on the 8-bit bus MSG_ADDRESS[7:0], and the data given on the 8-bit bus MSG_DIN[7:0] will be valid. When the access is acknowledged by asserting the MSG_READY signal, this indicates that the data has been registered.

If the access is a read, the address given on the 8-bit bus MSG_ADDRESS[7:0] will be valid. When the access is acknowledged by asserting the MSG_READY signal, there must be valid data driven on the 8-bit data bus MSG_DOUT[7:0].

When not using the receiving messages with this component, the MSG_DIN and MSG_ADDRESS bus outputs may be left unconnected.

Master I/F

When using this interface to send HSB messages as a “master” the inputs MSG_DOUT[7:0], MSG_SEND, MSG_SEND_ID, MSG_LAST_BYTE and MSG_CE are all used. In addition the outputs MSG_DONE, MSG_COUNT and MSG_CLEAR will be used by the surrounding logic.

The MSG_CE is used to enable the MSG_CLOCK signal when used by the message sending function. This is because the message sending function has an operating limit of 6.25MHz. Therefore where the MSG_CLOCK input is being driven at a frequency greater than 6.25MHz the CE input must be used to control the speed of operation. For example, when connecting a free running 50MHz clock to the MSG_CLOCK input, the MSG_CE input must be asserted (driven high) for only 1 in every eight clock cycles in order to set the operating frequency of the message sending function to 6.25MHz.

The MSG_SEND input is used to begin message transmission. This input must be asserted (set high) when the message is ready for transmission. It must remain asserted until the message sending process has been started. This can be seen by the MSG_CLEAR output becoming asserted (set high).

The MSG_SEND_ID input is used to indicate when a message byte must be replaced by the carrier ID and module ID information provided on the pins of the module. For a data transmission byte where the ID information must be sent, the MSG_SEND_ID signal must be asserted (set high).

The MSG_LAST_BYTE input is used to indicate when the last byte to be transmitted is being presented on the MSG_DOUT[7:0] input. To indicate that the last byte is being presented it must be asserted (set high) during that byte.

The message sending function of the HE_USER component is designed to be directly connected to a memory component that contains the message bytes in consecutive locations. The address of this memory should be driven by a counter, and the output of the memory connected directly to the MSG_DOUT[7:0] input.

The address counter should be controlled by the MSG_COUNT and MSG_CLEAR outputs of this component. The MSG_COUNT signal should be connected directly to the count enable of the counter. The counter will then count, resulting in a change of memory address, when the MSG_COUNT signal is asserted (set high).

The MSG_CLEAR signal should be connected directly to the asynchronous clear of the counter. The MSG_CLEAR signal is asserted (set high) at the beginning of message transmission, and should be used to initialise the memory address so that the memory is outputting the first data byte in the first memory location.

The MSG_DONE output will be asserted (set high) when the message has been successfully transmitted.

If the message sending function is not being used the inputs MSG_DOUT[7:0], MSG_SEND, MSG_SEND_ID, MSG_LAST_BYTE and MSG_CE should all be set low (connected to GND). The outputs MSG_DONE, MSG_COUNT and MSG_CLEAR may be left unconnected.

See the generic example “fpgaHSBmastering” for an example of using this “Master” interface.

Digital I/O Interface

Many of the HERON-FPGA and HERON-IO modules provide a large number of digital I/O connectors that can be used for your own digital I/O. They are connected to the user_ap using bi-directional LVTTTL buffers. If this is correct for you then you can simply connect to the CONN_n_IN and CONN_n_OUT busses and enable the direction using the CONN_n_EN busses.

If you require a different I/O format then you need to edit the top.vhd. For instructions on editing the top.vhd file, you will need to refer to the relevant section in the user manual for the module you are using.

Analogue I/O Interfaces

HERON-IO modules typically include both ADC and DAC interfaces. With both interfaces there will be a sample clock that is either internally sourced or externally sourced. For each analogue I/O application you will need to consider how the sample clock is sourced and what to drive on the signal SRC_SCLK. You must also ensure that the internal sample clock SCLK is used to clock all logic that directly interfaces to the ADC and DAC components.

ADC Interface

ADC interfaces will vary in terms of the number of bits of digital data produced by the external A-to-D converter. Regardless of the width of this sample data bus, there will be one input bus to the USER_AP entity for each ADC input channel named 'ADC_x(N downto 0), with x being the channel identifier and N being the number of bits of conversion, minus 1.

The ADC interface will produce a new sample on each clock cycle. This data must be clocked into the next stage of processing using the sample clock SCLK.

Usually, there will also be an out-of-range signal produced by the A/D converter. This will be passed into the USER_AP entity to be valid at the same time as the value on the corresponding ADC data bus.

DAC Interface

DAC interfaces will vary in terms of the number of bits of digital data sent to the external D-to-A converter. Regardless of the width of this sample data bus, there will be one output bus from the USER_AP entity for each DAC output channel named 'DAC_x(N downto 0), with x being the channel identifier and N being the number of bits of data, minus 1.

The output sample data will be registered inside the Hardware Interface Layer component HE_DAC on the rising edge of the associated SCLK sample clock. The sample data generated by your application must be generated from logic clocked by the same SCLK sample clock.

ADC Interface Examples

For each HERON-IO module type there will be several examples provided on the HUNT ENGINEERING CD that demonstrate using these examples. Please work through these examples and also refer to the User Manual for those boards, in order to fully understand the capabilities of each of the interfaces.

SDRAM Interface

This section discusses the Single Data Rate (SDR) SDRAM memory interface. This interface is simply referred to as the 'SDRAM Interface'. For a discussion of the Double Data Rate (DDR) SDRAM memory interface, please refer to the following section entitled 'DDR Interface'.

Some modules are available with one or more banks of SDRAM memory. If you are using a module that has an SDRAM interface, this will be presented via the HE_SDRAM component of the Hardware Interface Layer. This component presents all of the signals necessary to read and write a bank of SDRAM memory. Issues such as SDRAM initialisation and refresh are performed internally by the HE_SDRAM component, leaving the user free to concentrate on simply reading and writing the memory. The HE_SDRAM component presents two ports to the user. A read port for data to be read out of the external SDRAM, and a write port for data to be written to the SDRAM.

SDRAM Clock

The SDRAM interface operates at 133MHz. This frequency is obtained by using a Digital Clock Manager (DCM) to frequency synthesize 133MHz from an input clock of 100MHz. This is automatically done inside the HE_SDRAM component.

The generated 133MHz clock is provided as input to the USER_AP entity as the signal SDCLK. This clock signal must be used to clock all logic that directly interfaces to the SDRAM write port and read port.

SDRAM Write Port

The HE_SDRAM component provides a 'write port' that allows data to be written to the SDRAM. SDRAM memory requires a small amount of time to open a 'row' of memory before a read or write can be performed. When a row has been opened, the SDRAM can transfer one word of data on each clock cycle.

The SDRAM is organised as 32-bit wide memory. The SDRAM interface always runs at 133MHz. Therefore, once a row has been opened, data can be written in a burst of consecutive clock cycles at a data rate of 532Mbytes/sec.

Before a write access can be performed to the memory, the write address must first be initialised. Once the write address has been set up, it will automatically increment after each word is written. Therefore, the next address initialisation is only required when a new area of memory is to be written.

The signal SDRAM_WR_ADDR(25 downto 0) must be driven with the 26-bit address for the next write access. This value is latched by the SDRAM write interface when the address enable SDRAM_WR_AEN is asserted (set high). The address enable must be set high for one clock cycle while the address bus is valid, and then immediately de-asserted.

With the address initialised data can be written to the SDRAM. To begin a memory write operation, the SDRAM_WR_READY signal must be asserted (set high) to indicate that there is data to write. When the SDRAM interface detects the write-data-ready condition, it will open the SDRAM row indicated by the previously initialised write address.

When the row-open process has completed data will be transferred. For each word transferred, the SDRAM_WR_DONE signal will be asserted (set high) by the SDRAM interface. On each clock cycle where the SDRAM_WR_DONE signal is asserted, valid data is expected on the write data bus, SDRAM_WR_DATA(31 downto 0).

The signal SDRAM_WR_BURST signal should be used along with the SDRAM_WR_READY signal to indicate that there are multiple words of data to transfer. While both signals are asserted (set high) this indicates that the SDRAM interface can burst data to the external memory. The write burst will end if either the end of the row is reached, or if the burst signal becomes de-asserted. If the end of row is reached, the burst must end so that the row can be closed. If there is still more data to transfer, the next new row will be opened when the previous has been closed.

SDRAM Read Port

The HE_SDRAM component provides a 'read port' that allows data to be read from the SDRAM. SDRAM memory requires a small amount of time to open a 'row' of memory before a read or write can be performed. When a row has been opened, the SDRAM can transfer one word of data on each clock cycle.

The SDRAM is organised as 32-bit wide memory. The SDRAM interface always runs at 133MHz. Therefore, once a row has been opened, data can be written in a burst of consecutive clock cycles at a data rate of 532Mbytes/sec.

Before a read access can be performed to the memory, the read address must first be initialised. Once the read address has been set up, it will automatically increment after each word is written. Therefore, the next address initialisation is only required when a new area of memory is to be read.

The signal SDRAM_RD_ADDR (25 downto 0) must be driven with the 26-bit address for the next read access. This value is latched by the SDRAM read interface when the address enable SDRAM_RD_AEN is asserted (set high). The address enable must be set high for one clock cycle while the address bus is valid, and then immediately de-asserted.

With the address initialised data can be read from the SDRAM. To begin a memory read operation, the SDRAM_RD_READY signal must be asserted (set high) to indicate that there is space to receive data. When the SDRAM interface detects the read-data-ready condition, it will open the SDRAM row indicated by the previously initialised read address.

When the row-open process has completed data will be transferred. Unlike the write interface however, the SDRAM_RD_DONE signal does not directly indicate that a data word is being output. This is because the external SDRAM memory has a data access delay of several clock cycles. For each assertion of SDRAM_RD_DONE this indicates that an item of data has been requested from the memory.

For each word actually transferred, the SDRAM_RD_DVAL signal will be asserted (set high) by the SDRAM interface. On each clock cycle where the SDRAM_RD_DVAL signal is asserted, valid data is presented on the read data bus, SDRAM_RD_DATA (31 downto 0).

The signal SDRAM_RD_BURST signal should be used along with the SDRAM_RD_READY signal to indicate that there are multiple words of data to be read. While both signals are asserted (set high) this indicates that the SDRAM interface can accept a burst of data from the external memory. The read burst will end if either the end of the row is reached, or if the burst signal becomes de-asserted. If the end of row is reached, the burst must end so that the row can be closed. If there is still more data to transfer, the next new row will be opened when the previous has been closed.

SDRAM INTERFACE EXAMPLES

There are two simple memory read and memory write examples discussed in the document 'Accessing SDRAM in Your FPGA Design'. This document is available on the HUNT ENGINEERING CD and can be located using the CD browser tool and navigating to the 'Getting Started with FPGA' area.

You will also find VHDL example projects that show the use of the SDRAM interface. These examples exist for module types that include the SDRAM interface.

DDR Interface

This section discusses the Double Data Rate (DDR) SDRAM memory interface. This interface is simply referred to as the 'DDR Interface'. For a discussion of the Single Data Rate (SDR) SDRAM memory interface, please refer to the previous section entitled 'SDRAM Interface'.

Some modules are available with one or more banks of DDR SDRAM memory. If you are using a module that has a DDR SDRAM interface, this will be presented via the HE_DDR component of the Hardware Interface Layer. This component presents all of the signals necessary to read and write a bank of DDR memory. Issues such as DDR initialisation and refresh are performed internally by the HE_DDR component, leaving the user free to concentrate on simply reading and writing the memory. The HE_DDR component presents either two or four ports to the user.

For modules with one bank of memory, there is a read port for data to be read out of the external DDR, and a write port for data to be written to the DDR memory. For modules with two banks of memory, there are two read ports and two write ports.

DDR Clock

The DDR interface operates at 200MHz. This frequency is obtained directly from a free running 200MHz clock source fed into the FPGA. The 200MHz clock source is input to a DCM that generates the necessary internal and external clock signals for DDR operation.

The DDR read interface and DDR write interface are managed using FIFO components internal to the HE_DDR component. These FIFOs are asynchronous, in that within the HE_DDR internal logic data is read and written at 200MHz. On the user side of these FIFOs data is transferred in a clock domain created by the user logic.

For a module with two separate banks of DDR memory, the signal DDR_A_USER_CLK must be correctly driven by the user in order to use the read FIFO and write FIFO of DDR Bank A. The signal DDR_B_USER_CLK must be correctly driven by the user in order to use to the read FIFO and write FIFO of DDR Bank B.

DDR Write Port

The HE_DDR component provides a 'write port' that allows data to be written to the DDR memory. DDR memory is designed for high burst performance by using rising edge and falling edge data. This is called 'double-data-rate' (DDR) transfer.

For one clock cycle of the memory, one word of data is transferred on the rising clock edge, and one word is transferred on the falling edge. The result is two words every clock cycle. DDR memory requires a small amount of time to open a 'row' of memory before a read or write can be performed, but when the row has been opened data can be transferred at a very high rate.

Due to the nature of DDR memory, data needs to be written in bursts to make the best use of the available bandwidth. The write port therefore requires data is written to the memory in groups of four 32-bit words to ensure bursting is efficient.

DDR Write Port FIFOs

Data is written to the DDR memory through FIFO components internal to the HE_DDR component. The memory write interface that is presented to the USER_AP entity therefore appears as a FIFO interface through which all memory write operations are performed.

To perform one write access to the DDR memory, four words of data must be written to the write interface, along with one word that defines the starting address for the burst, and two bytes of data mask information. All of this is done through the use of four separate write interface FIFOs.

The DDR write interface is organised as a rising-edge data FIFO, a falling edge data FIFO, an address FIFO and a data mask FIFO. The data words written to the rising edge FIFO and falling edge FIFO are 32-bits wide. The data written to the address FIFO is 25-bits wide and the data written to the data mask FIFO is 8-bits wide.

When transferring data words 'data-0', 'data-1', 'data-2' and 'data-3' to memory each of these FIFOs would be written as follows:

- Data word 0 is written to the write port rising edge FIFO
- Data word 1 is written to the write port falling edge FIFO
- Data mask is written for data word 0 and data word 1 to the data mask FIFO
- Data word 2 is written to the next location of write port rising edge FIFO
- Data word 3 is written to the next location of write port falling edge FIFO
- Data mask is written for data word 2 and data word 3 to the data mask FIFO
- An address word is written to the write port address FIFO

As data must always be written in blocks of four words, a Data Mask function has been provided. The Data Mask function allows individuals bytes to be explicitly enabled or disabled in the data sent to the DDR memory. For any byte that whose Data Mask bit is disabled, no data will be written to the memory at that location. This allows blocks of data smaller than four words to be written to the DDR memory interface.

Conversely, where bursting of larger amounts of data is required, this is simply done by keeping the write data FIFOs full with corresponding addresses that fall within the same row.

That is, if there are several blocks of four that can be written at addresses within the same row, the DDR interface will continue to transfer to the opened row as long as valid data exists in the rising and falling write data FIFOs.

As each address word is read from the write address FIFO it is compared to the currently open row. If the top 15-bits of the next address exactly match the currently open row, then the next block of four data words will be transferred immediately to the already open memory.

In this way by ensuring a consecutive set of addresses are used in the values written to the address FIFO, bursts much larger than four words can be created to the external DDR memory.

DDR Write Port FIFO Signals

For the rising edge, falling edge, address and data mask write FIFOs, one FIFO clock signal is required. This signal is also used for the read interface. For DDR Memory Bank A, the signal is named `DDR_A_USER_CLK`. For DDR Memory Bank B, the signal is named `DDR_B_USER_CLK`.

This clock signal must be sourced inside the `USER_AP` part of the design hierarchy. The resulting clock signal presented to the `HE_DDR` component is connected to the write clock input of all four write interface FIFOs.

All logic that directly interfaces to the FIFOs must be clocked with the same clock signal driven out to the `HE_DDR` component.

For the following section, all signal names given will be for the first bank of DDR memory. All DDR Memory Bank A signals begin '`DDR_A_`'. For interfacing to DDR Memory Bank B, simply rename the signal to start '`DDR_B_`'.

The write address FIFO is 25-bits wide. Data for the write address FIFO must be driven on the bus `DDR_A_WR_ADDR` (24 downto 0). To write one address value to the write address FIFO the signal `DDR_A_WR_ADDR_WEN` must be asserted (set high) for one clock cycle. The `DDR_A_WR_ADDR` bus must be valid in the clock cycle where `DDR_A_WR_ADDR_WEN` is asserted.

The write address FIFO provides two flags to indicate whether a write can proceed. The signal `DDR_A_WR_ADDR_FF` is the full flag for the write address FIFO. When this signal is asserted (set high) the write address FIFO is full and any further writes will be ignored. The signal `DDR_A_WR_ADDR_AF` is the almost full flag. When this signal is asserted (set high) the write address FIFO is almost full. Only one more write can be completed while the FIFO is almost full.

The write rising data FIFO is 32-bits wide. Data for the rising data FIFO must be driven on the bus `DDR_A_WR_DATA(39 downto 8)`. Please note, the `DDR_A_WR_DATA` bus is shared for accesses to the rising data FIFO, the falling data FIFO and the data mask FIFO. Ensure you use the correct portion of this bus according to the FIFO you are updating.

To write one word to the rising data FIFO the signal `DDR_A_WR_RISE_WEN` must be asserted (set high) for one clock cycle. The `DDR_A_WR_DATA(39 downto 8)` bus must be valid in the clock cycle where `DDR_A_WR_RISE_WEN` is asserted.

The rising data FIFO provides two flags to indicate whether a write can proceed. The signal `DDR_A_WR_RISE_FF` is the full flag for the rising data FIFO. When this signal is asserted (set high) the rising data FIFO is full and any further writes will be ignored. The signal `DDR_A_WR_RISE_AF` is the almost full flag. When this signal is asserted (set high) the rising data FIFO is almost full. Only one more write can be completed while the FIFO is almost full.

The write falling data FIFO is 32-bits wide. Data for the falling data FIFO must be driven on the bus `DDR_A_WR_DATA(71 downto 40)`. Please note, the `DDR_A_WR_DATA` bus is shared for accesses to the rising data FIFO, the falling data FIFO and the data mask FIFO. Ensure you use the correct portion of this bus according to the FIFO you are updating.

To write one word to the falling data FIFO the signal `DDR_A_WR_FALL_WEN` must be asserted (set high) for one clock cycle. The `DDR_A_WR_DATA(71 downto 40)` bus must be valid in the clock cycle where `DDR_A_WR_FALL_WEN` is asserted.

The falling data FIFO provides two flags to indicate whether a write can proceed. The signal `DDR_A_WR_FALL_FF` is the full flag for the falling data FIFO. When this signal is asserted (set high) the falling data FIFO is full and any further writes will be ignored. The signal `DDR_A_WR_FALL_AF` is the almost full flag. When this signal is asserted (set high) the falling data FIFO is almost full. Only one more write can be completed while the FIFO is almost full.

The write data mask FIFO is 8-bits wide. Data for the data mask FIFO must be driven on the bus `DDR_A_WR_DATA(7 downto 0)`. Please note, the `DDR_A_WR_DATA` bus is shared for accesses to the rising data FIFO, the falling data FIFO and the data mask FIFO. Ensure you use the correct portion of this bus according to the FIFO you are updating.

To write one byte to the data mask FIFO the signal `DDR_A_WR_DM_WEN` must be asserted (set high) for one clock cycle. The `DDR_A_WR_DATA(7 downto 0)` bus must be valid in the clock cycle where `DDR_A_WR_DM_WEN` is asserted.

The data mask FIFO provides two flags to indicate whether a write can proceed. The signal `DDR_A_WR_DM_FF` is the full flag for the data mask FIFO. When this signal is asserted (set high) the data mask FIFO is full and any further writes will be ignored. The signal `DDR_A_WR_DM_AF` is the almost full flag. When this signal is asserted (set high) the data mask FIFO is almost full. Only one more write can be completed while the FIFO is almost full.

Each bit of the byte value written to the data mask FIFO corresponds to an enable for one byte lane of data. If the corresponding bit is set low, the data byte is written to memory (enabled) and if the corresponding bit is set high, the data byte is not written (disabled). The bits in the data mask are defined as follows:

Bit 0	Byte 0 of falling data
Bit 1	Byte 1 of falling data
Bit 2	Byte 2 of falling data
Bit 3	Byte 3 of falling data
Bit 4	Byte 0 of rising data
Bit 5	Byte 1 of rising data
Bit 6	Byte 2 of rising data
Bit 7	Byte 3 of rising data

DDR Read Port

The HE_DDR component provides a ‘read port’ that allows data to be read from the DDR memory. DDR memory is designed for high burst performance by using rising edge and falling edge data. This is called ‘double-data-rate’ (DDR) transfer.

For one clock cycle of the memory, one word of data is transferred on the rising clock edge, and one word is transferred on the falling edge. The result is two words every clock cycle. DDR memory requires a small amount of time to open a ‘row’ of memory before a read or write can be performed, but when the row has been opened data can be transferred at a very high rate.

Due to the nature of DDR memory, data needs to be read in bursts to make the best use of the available bandwidth. The read port therefore requires data is read from memory in groups of four 32-bit words to ensure bursting is efficient.

DDR Read Port FIFOs

Data is read from the DDR memory through FIFO components internal to the HE_DDR component. The memory read interface that is presented to the USER_AP entity therefore appears as a FIFO interface through which all memory read operations are performed.

To perform one read access from the DDR memory, one word must be written to define the starting address for the burst. Once the access from memory has been completed four words of data will be presented to the read interface. All of this is done through the use of three separate read interface FIFOs.

The DDR read interface is organised as a rising-edge data FIFO, a falling edge data FIFO and an address FIFO. The data words read from the rising edge FIFO and falling edge FIFO are 32-bits wide. The data written to the address FIFO is 25-bits wide.

When transferring data words ‘data-0’, ‘data-1’, ‘data-2’ and ‘data-3’ from memory each of these FIFOs would be used as follows:

- An address word is written to the read port address FIFO
- ...when the access to memory has completed...
- Data word 0 is read from the read port rising edge FIFO
- Data word 1 is read from the read port falling edge FIFO
- Data word 2 is read from the read port rising edge FIFO
- Data word 3 is read from the read port falling edge FIFO

As data must always be read in blocks of four words this may mean more words of data are read than were required. Unlike the write port, no data mask function is provided. Instead, words of data that are read but not required must simply be discarded by the logic that reads from the read rising data FIFO and read falling data FIFO. This will allow blocks of data smaller than four words to be read from the DDR memory interface.

Conversely, where bursting of larger amounts of data is required, this is simply done by keeping the read data FIFOs empty and providing multiple addresses that fall within the same row.

That is, if there are several blocks of four that need to be read from addresses within the same row, the DDR interface will continue to transfer from the opened row as long as there is space in the rising and falling read data FIFOs and appropriate addresses exist in the address FIFO.

As each address word is read from the read address FIFO it is compared to the currently open row. If the top 15-bits of the next address exactly match the currently open row, then the next block of four data words will be transferred immediately from the already open memory.

In this way by ensuring a consecutive set of addresses are used in the values written to the address FIFO, bursts much larger than four words can be created from the external DDR memory.

DDR Read Port FIFO Signals

For the rising edge, falling edge and address FIFOs, one FIFO clock signal is required. This signal is also used for the write interface. For DDR Memory Bank A, the signal is named `DDR_A_USER_CLK`. For DDR Memory Bank B, the signal is named `DDR_B_USER_CLK`.

This clock signal must be sourced inside the `USER_AP` part of the design hierarchy. The resulting clock signal presented to the `HE_DDR` component is connected to the clock inputs of all three read interface FIFOs.

All logic that directly interfaces to the FIFOs must be clocked with the same clock signal driven out to the `HE_DDR` component.

For the following section, all signal names given will be for the first bank of DDR memory. All DDR Memory Bank A signals begin '`DDR_A_`'. For interfacing to DDR Memory Bank B, simply rename the signal to start '`DDR_B_`'.

The read address FIFO is 25-bits wide. Data for the read address FIFO must be driven on the bus `DDR_A_RD_ADDR(24 downto 0)`. To write one address value to the read address FIFO the signal `DDR_A_RD_ADDR_WEN` must be asserted (set high) for one clock cycle. The `DDR_A_RD_ADDR` bus must be valid in the clock cycle where `DDR_A_RD_ADDR_WEN` is asserted.

The read address FIFO provides two flags to indicate whether a write can proceed. The signal `DDR_A_RD_ADDR_FF` is the full flag for the read address FIFO. When this signal is asserted (set high) the read address FIFO is full and any further writes will be ignored. The signal `DDR_A_RD_ADDR_AF` is the almost full flag. When this signal is asserted (set high) the read address FIFO is almost full. Only one more write can be completed while the FIFO is almost full.

The read rising data FIFO is 32-bits wide. Data from the rising data FIFO must be read on the bus `DDR_A_RD_DATA(31 downto 0)`. Please note, the `DDR_A_RD_DATA` bus is shared for accesses from the rising data FIFO and the falling data FIFO. Ensure you use the correct portion of this bus according to the FIFO you are reading.

To read one word from the rising data FIFO the signal `DDR_A_RD_RISE_REN` must be asserted (set high) for one clock cycle. The `DDR_A_RD_DATA(31 downto 0)` bus will become valid in the following clock cycle from where `DDR_A_RD_RISE_REN` was asserted.

The rising data FIFO provides two flags to indicate whether a read can proceed. The signal `DDR_A_RD_RISE_EF` is the empty flag for the rising data FIFO. When this signal is asserted (set high) the rising data FIFO is empty and any further reads will be ignored. The signal `DDR_A_RD_RISE_AE` is the almost empty flag. When this signal is asserted (set high) the rising data FIFO is almost empty. Only one more read can be completed while the FIFO is almost empty.

The read falling data FIFO is 32-bits wide. Data from the falling data FIFO must be read on the bus `DDR_A_RD_DATA(63 downto 32)`. Please note, the `DDR_A_RD_DATA` bus is shared for accesses from the rising data FIFO and the falling data FIFO. Ensure you use the correct portion of this bus according to the FIFO you are reading.

To read one word from the falling data FIFO the signal `DDR_A_RD_FALL_REN` must be asserted (set high) for one clock cycle. The `DDR_A_RD_DATA(63 downto 32)` bus will become valid in the following clock cycle from where `DDR_A_RD_FALL_REN` was asserted.

The falling data FIFO provides two flags to indicate whether a read can proceed. The signal `DDR_A_RD_FALL_EF` is the empty flag for the falling data FIFO. When this signal is asserted (set high) the falling data FIFO is empty and any further reads will be ignored. The signal `DDR_A_RD_FALL_AE` is the almost empty flag. When this signal is asserted (set high) the falling data FIFO is almost empty. Only one more read can be completed while the FIFO is almost empty.

DDR Data Count

At any one point in time, the internal logic of the HE_DDR component can only be performing either a read or a write, if the DDR memory interface is active.

The user logic however is free to update the write interface FIFOs and read interface FIFOs at the same time. The internal logic of the HE_DDR component must therefore decide based on this activity whether to perform a read or a write.

As the FIFOs decouple read and write activity between the USER_AP logic and the HE_DDR internal logic it is not always obvious from the USER_AP side of the FIFOs whether a write of data has completed. Sometimes the data to be written may still be waiting inside the rising data and falling data write FIFOs as a different DDR access is currently underway.

There are design situations where it is necessary to know how much data has been read or written inside the HE_DDR component. A simple example of such a situation where this is important is where the DDR memory interface is being used to create a large FIFO.

When creating a FIFO using DDR memory, logic in the USER_AP part of the design must keep track of how many items are in the FIFO, or rather, how many words of data have been stored in DDR SDRAM.

For example, if 8 words are written to the write interface FIFOs, the USER_AP logic might wish to record that the 'DDR FIFO' now contains 8 items. So if this is done, the read side of DDR FIFO will now think it correct to allow a read from memory of up to 8 words. While the 8 words are still in the write interface FIFOs and not in external memory this is incorrect.

In order to avoid this situation the HE_DDR component provides a data count bus named DDR_A_DATA_COUNT (31 downto 0) for Bank A and a bus named DDR_B_DATA_COUNT (31 downto 0) for Bank B. For each data count there is a count reset signal. For Bank A this signal is DDR_A_USER_RST and for Bank B this signal is DDR_B_USER_RST.

The reset signal is used to set the count value to zero. This reset signal is asynchronous, but to ensure correct operation it should be asserted for at least 10ns.

The data count logic records the number of accesses made to the DDR memory. For each group of four words written to memory (one write access) the data count increments by one, and for each group of four words read from memory (one read access) the data count decrements by one. The counter is 32-bits in size and will roll around when counting up from all bits set, and will roll when counting down from zero.

An example of the use of the data count function can be seen in the SDRAM_FIFO example provided for all boards that have a DDR interface.

DDR INTERFACE EXAMPLES

There are two simple memory read and memory write examples discussed in the document 'Accessing DDR Memory in Your FPGA Design'. This document is available on the HUNT ENGINEERING CD and can be located using the CD browser tool and navigating to the 'Getting Started with FPGA' area.

You will also find VHDL example projects that show the use of the DDR memory interface. These examples exist for module types that include the DDR memory interface.

SPI Interface

Some modules are available with a Serial Peripheral Interface (SPI). The Serial Peripheral Interface on these modules will be used to interface to a specific device connected to the FPGA. For example, on a board with a DAC component, there may be a Serial Peripheral Interface connection for controlling operation of the DAC.

If you are using a module that has a SPI interface, this will be presented via the HE_SPI component of the Hardware Interface Layer. This component presents all of the signals necessary to read and write data via SPI.

The SPI interface must be provided with a free running clock signal set to 100MHz or lower. All signals of the HE_SPI component are synchronised to the SPI clock signal SPI_CLOCK.

For every read and write performed over SPI, an instruction byte must first be transmitted by the FPGA. The instruction byte specifies the address for the serial access, whether the access is a read or a write, and the number of bytes to be transferred.

SPI Write

To transmit data over SPI the SPI_R_W signal must be set low. The SPI_COUNT(2 downto 0) signal must be set to the number of bytes to transmit, and the SPI_ADDRESS(4 downto 0) signal must be set to the start address for the access.

For transfers of more than one byte, please refer to the data sheet for the external device you are communicating with to ascertain whether the address auto-increments or auto-decrements.

Next the data to be written must be latched into the HE_SPI interface. This is done by driving data onto the SPI_WR_DATA(7 downto 0) bus and asserting the appropriate SPI load signal. It is possible to transmit either 1, 2, 3 or 4 bytes of data for each SPI transaction.

Inside the HE_SPI interface are four transmit data registers, data-out-0, data-out-1, data-out-2 and data-out-3. When writing data from the FPGA to the external device, the HE-SPI interface starts with the data in register data-out-0. If more than one byte is to be transmitted it then moves on to the next data register.

To set the contents of data-out-0, the data must be driven on the SPI_WR_DATA(7 downto 0) bus while the SPI_LOAD_0 signal is asserted (set high). Data is registered on the rising edge of the SPI clock signal, SPI_CLOCK. To set the contents of data-out-1, the data must be driven on the SPI_WR_DATA bus while the SPI_LOAD_1 signal is asserted. To set the contents of data-out-2, the data must be driven on the SPI_WR_DATA bus while the SPI_LOAD_2 signal is asserted. To set the contents of data-out-3, the data must be driven on the SPI_WR_DATA bus while the SPI_LOAD_3 signal is asserted.

When the address, count, read-write control and data values are correctly set, the data can be written over SPI. Data is transmitted by asserting the signal SPI_TRANSMIT (setting high). The SPI_R_W, SPI_COUNT and SPI_ADDRESS signals are all registered on the rising edge of SPI_CLOCK when SPI_TRANSMIT is asserted. Once asserted, SPI_TRANSMIT must remain asserted until the SPI_ACTIVE signal goes high to indicate transfer has begun. When SPI_ACTIVE becomes asserted (set high), the SPI_TRANSMIT signal must be de-asserted.

When SPI_ACTIVE returns low, this indicates data transmission has completed.

SPI Read

To read data over SPI the SPI_R_W signal must be set high. The SPI_COUNT(2 downto 0) signal must be set to the number of bytes to receive, and the SPI_ADDRESS(4 downto 0) signal must be set to the start address for the access.

For transfers of more than one byte, please refer to the data sheet for the external device you are communicating with to ascertain whether the address auto-increments or auto-decrements.

When the address, count and read-write signals are correctly set, the data read can be started over SPI by asserting the signal `SPI_TRANSMIT` (setting high). The `SPI_R_W`, `SPI_COUNT` and `SPI_ADDRESS` signals are all registered on the rising edge of `SPI_CLOCK` when `SPI_TRANSMIT` is asserted. Once asserted, `SPI_TRANSMIT` must remain asserted until the `SPI_ACTIVE` signal goes high to indicate transfer has begun. When `SPI_ACTIVE` becomes asserted (set high), the `SPI_TRANSMIT` signal must be de-asserted.

When `SPI_ACTIVE` returns low, this indicates data transmission has completed.

When the transmission of data has completed one or more bytes of read data will be available on the output busses `SPI_RD_DATA_n`.