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

Accessing Multiple FIFOs in your FPGA Design

v 1.0 R.Williams 12-12-02

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.

With Version 2.0 of the Hardware Interface Layer, HUNT ENGINEERING introduced two new interface functions that provide full access to multiple HERON input FIFOs and multiple HERON output FIFOs.

This document discusses the capabilities of the new FIFO interface components, using several different examples to illustrate the many different ways that these new components can be used. For users that have an existing project using version 1.x of the Hardware Interface Layer, this document can be used alongside the document 'Converting HIL v1.x Projects to v2.0' which discusses how to move to the new v2.0 Hardware Interface Layer.

History

Rev 1.0	First written

Multiple FIFO Access Components

With Version 2.0 of the Hardware Interface Layer, HUNT ENGINEERING introduced two new interface components for reading and writing multiple HERON FIFOs.

The component HE_RD_6F enables an FPGA design to read from up to six HERON Input FIFOs concurrently, and has been added in place of the existing component HE_RD_1F. The component HE_WR_6F enables an FPGA design to write to up to six HERON Output FIFOs concurrently, and has been added in place of the existing component HE_WR_1F.

The interface components HE_RD_6F and HE_WR_6F translate the access of the external HERON FIFO interfaces provided on all HERON modules into a simple high performance internal interface to which a user can connect their own logic.

The Six HERON-FIFO Read Interface

The HE_RD_6F component works by providing six 'read-request' signals. Each time a particular FIFO read request is asserted the HE_RD_6F component will try to provide data for that FIFO. Whether any data can be transferred will depend on the state of the external HERON Input FIFO being accessed.

When the external HERON Input FIFO contains data and the read-request is asserted, data will be passed to the user application on one of six data busses, `INFIFO0_D` to `INFIFO5_D`. For each clock cycle where one of these busses contains valid FIFO data, a corresponding 'data-valid' signal is asserted by the HE_RD_6F component.

For example, if the user application has requested data from FIFO 2 by asserting the FIFO 2 read-request signal INFIFO_READ_REQ(2), then when data is available it will be placed on the `INFIFO2_D` data bus and in the same clock cycle the data valid signal `INFIFO_DVALID(2)` will be asserted.

In addition to the data transfer control signals the HE_RD_6F component provides two status signals per FIFO. The signals `INFIFO_SINGLE` and `INFIFO_BURST` indicate the state of the read FIFO interface, with the `INFIFO_SINGLE` flag representing whether there is at least a single word to read, and the `INFIFO_BURST` flag representing that there are multiple data elements to be read from the external HERON FIFOs.

The Six HERON-FIFO Write Interface

The HE_WR_6F component provides six write control signals that are asserted by the user application when transferring data to the external HERON Output FIFOs. In combination with the write control signals, the HE_WR_6F component provides six 'ready' signals that indicate whether it is possible to transfer to the external FIFOs or whether those FIFOs are full.

While it is possible to transfer data to one of the external HERON Output FIFOs, the `OUTFIFO_READY` signal will be asserted for that FIFO (eg for FIFO 3 the signal `OUTFIFO_READY(3)` will be asserted). For each clock cycle where both the `OUTFIFO_READY` signal and the `OUTFIFO_WRITE` signal are asserted, data will be transferred to the FIFO using the `OUTFIFO_D` data bus.

While this situation exists, data can continue to be written to the same FIFO on every clock cycle. When the external FIFO approaches becoming full, the `OUTFIFO_READY` signal will become de-asserted. When this occurs, the user application must stop writing to the output FIFO. For the first clock cycle where the 'ready' signal first becomes de-asserted it is possible to write one more word of data, but after this last access, data transfer must be paused until the 'ready' signal returns to being asserted.

The HE_WR_6F component provides one data bus for transferring data. During any single clock period, only one of the six write control signals can be asserted; during that clock cycle the data bus must contain the data to be written to that respective FIFO. This requires that the user application performing any necessary data path switching necessary to place the correct data onto the `OUTFIFO_D` data bus.

Scheduling FIFO Access

The key difference in using the new interfaces is that it is now possible to read from six FIFOs concurrently and write to six FIFOs concurrently. However, for all read accesses the input data must be transferred using one shared HERON input bus, and similarly for all write accesses the output data must be transferred using one shared HERON output bus.

The sharing of one bus for input and one bus for output requires a mechanism for controlling how the input and output busses are shared amongst FIFO connections. With the Six FIFO Read and Six FIFO Write interface components, FIFO scheduling is handled in one of two ways depending on the requirements of your application. The method used will depend on the data bandwidth required by your FPGA design.

HERON FIFO Bandwidth

For a HERON FIFO interface operating at 100MHz, the maximum data rate through the Input FIFO interface or Output FIFO interface is 400Mbytes/sec. This is because the maximum transfer rate is one word of data every clock cycle, where each word is comprised of four bytes.

For the Six FIFO Write Interface one word of data can be transferred on each and every clock cycle while transferring data to the same FIFO. When you switch to a different output FIFO, the first word of data can be written to the newly selected FIFO immediately after the last word is written to the current output FIFO. As a result, the bandwidth available through the Six FIFO Write Interface is 400MBytes/sec.

For the Six FIFO Read Interface however, a different situation exists. For the read FIFO interface there must be one idle cycle in which no data is transferred, each time we change from one FIFO to the next. This means that when reading from multiple HERON FIFOs we must consider how the number of idle cycles will affect the available bandwidth.

If the all of the 'read-request' signals are tied high (asserted) by the application, then the HE_RD_6F component will automatically control the sequencing of FIFO accesses. This is done as follows; for each FIFO that has data that can be transferred to the user application, the read interface component will transfer a single word of data and then switch to the next ready FIFO. If all FIFOs have data, the HE_RD_6F component will service FIFO 0, then FIFO 1, then FIFO 2 all the way to FIFO 5. It will then return to service FIFO 0.

By behaving in this way, it is necessary that one idle cycle is performed in between one word of data transfer from a rotating choice of FIFO. The effect of this is that there are as many 'idle' cycles as there 'data-transfer' cycles. This results in a halving of the maximum bandwidth to 200Mbytes/sec.

The alternative method of bandwidth control is for the user application to decide when to access each FIFO. By using this method it is up to the user application to control how many words of data are read from each FIFO before moving onto the next ready FIFO. The change in FIFO number will insert the necessary idle cycle, which will be reflected by one idle cycle between a change of data valid assertions.

Whether a user application will tie all required read request signals high, or whether the application will actively control the read requests will depend on the bandwidth requirements of that application. It is a simple choice, made by considering the combined input FIFO bandwidth as follows.

Total Bandwidth thru. All Input FIFOs	FIFO Access Method
≤ 200Mbytes / second	Tie all required read-requests high
> 200Mbytes / second	Actively control read-requests

The Multiple FIFO Access Examples

There are several different examples that are presented in the remainder of this document that illustrate the many different ways in which the Six FIFO interface components may be used.

There are two examples provided as VHDL modules. These examples show how to control the process of reading from three FIFOs concurrently and write to three FIFOs concurrently. The first of these examples operates with the required 'read request' signals tied high. For this example, data is requested from FIFOs 0, 1 and 2 as soon as it is available. The scheduling of input FIFOs is automatically performed by the Six FIFO Read Interface component in the Hardware Interface Layer.

In the second VHDL example, the 'read request' signals are actively controlled inside the user application in order to control the sharing of the input FIFO bandwidth.

In addition to the VHDL examples, there are several other examples that present signal waveforms to show how the FIFO interfaces behave when used in different ways. There are four of these examples that discuss reading from the HERON Input FIFOs, and one example that discusses writing to the HERON Output FIFOs.

Finally, in the Appendix at the end of this document there is a discussion of how to decide on an algorithm for actively controlling the 'read_request' signals. This appendix is intended to be used along with the second VHDL example, Example2, and with Example 5 that shows bandwidth sharing with signal waveforms used for illustration.

Example 1: VHDL Example of Multiple FIFO Access with Tied Read Requests

In the same directory of the HUNT ENGINEERING CD that contains this document there is a VHDL source module named 'Example1.vhd'. The file Example1.vhd contains code that demonstrates reading and writing multiple HERON FIFOs.

In this example, data is requested from Input FIFOs 0, 1, and 2. This is done by permanently asserting `INFIFO_READ_REQ(0)`, `INFIFO_READ_REQ(1)` and `INFIFO_READ_REQ(2)`.

As data arrives in the external HERON Input FIFOs, the HE_RD_6F Hardware Interface Layer component automatically alternates between FIFOs, presenting the data on the input data busses along with asserting the appropriate `INFIFO_DVALID(0)`, `INFIFO_DVALID(1)` and `INFIFO_DVALID(2)` data valid signals.

The data valid signals are used as the write enables for three FIFOs internal to the user application. These FIFOs are simple CoreGenerator generated FIFOs. Each is 15 words deep, and 32-bits wide, with a full flag, empty flag, almost full flag and almost empty flag.

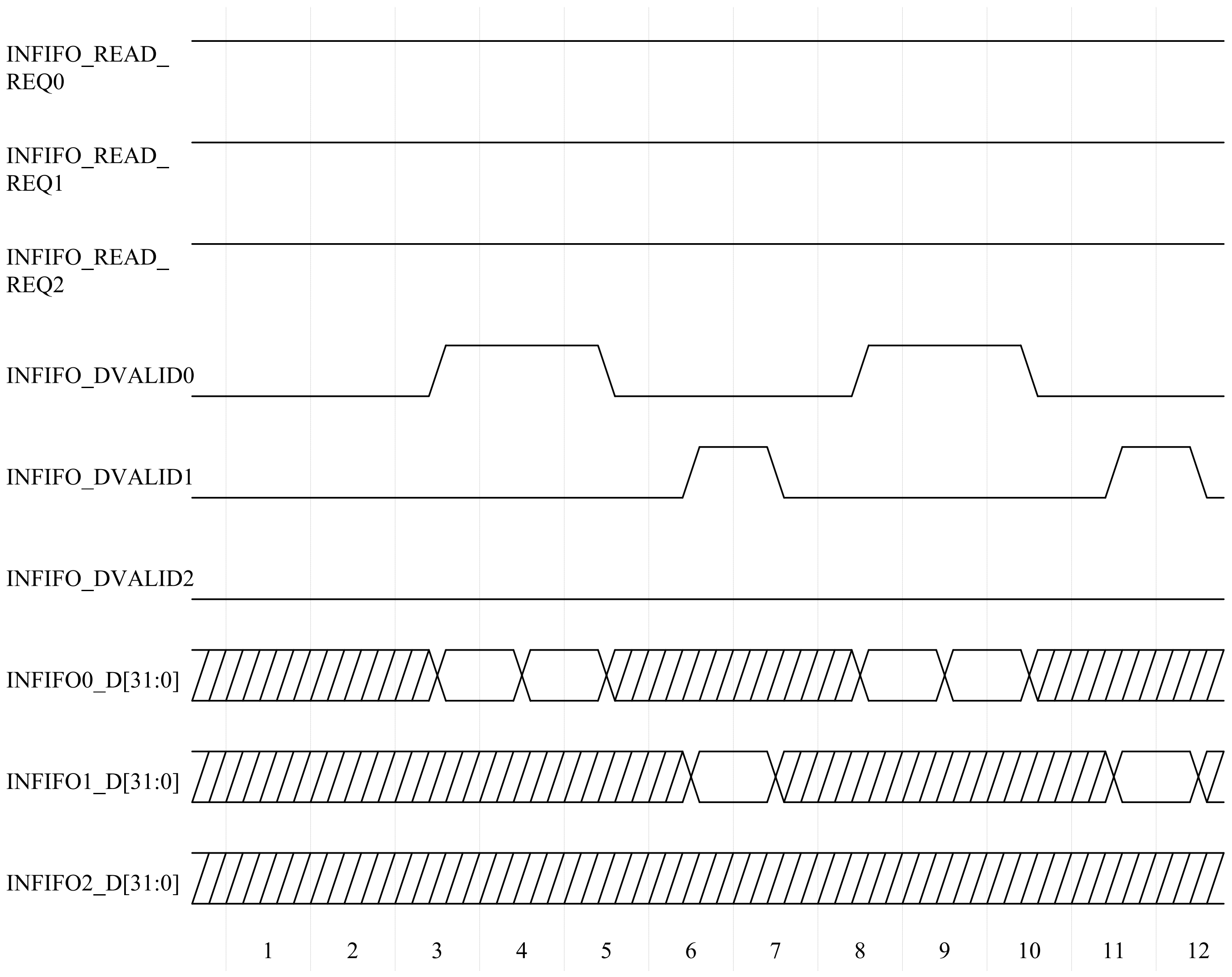
On the output side of the internal FIFOs is a simple state machine that rotates from internal FIFO A to internal FIFO B to internal FIFO C. For each FIFO, if it is not empty and the associated HERON Output FIFO is ready (`OUTFIFO_READY(n)` asserted), data is transferred to the HERON FIFO.

Example 2: VHDL Example of Multiple FIFO Access with Active Read Requests

Again, in the same directory of the HUNT ENGINEERING CD that contains this document there is a VHDL source module named 'Example2.vhd'. The file Example2.vhd contains code that demonstrates reading and writing multiple HERON FIFOs. This example differs from Example1 in that the input of data from FIFOs 0, 1 and 2 is dynamically controlled.

For this example, suppose we require a bandwidth of 175Mbytes/sec for FIFO 0 data transfer, 150Mbytes/sec for FIFO 1, and 25Mbytes/sec for FIFO2. This is achieved through a simple state machine that alternates the assertion of read requests in order to share the total available bandwidth. The calculation that was performed to work out the bandwidth sharing can be found in the appendix.

Example 3: Reading from Three FIFOs



In the above diagram, data is requested from FIFOs 0, 1 and 2 by the assertion of all three respective read-request signals. With these signals held asserted, the Six FIFO Read Interface will automatically present the data for each FIFO as it becomes available.

In this example, there is a lot of data to be presented from FIFO 0, some data for FIFO 1 and no data for FIFO 2. The interface will decide when to change from processing one FIFO to processing the next.

During clock cycle 2, the interface is only able to present data from FIFO 0. The data is read from external FIFO 0 and output in clock cycle 3. The interface continues to output data for FIFO 0 for the next clock cycle, as there is still no data for FIFO 1 or FIFO2.

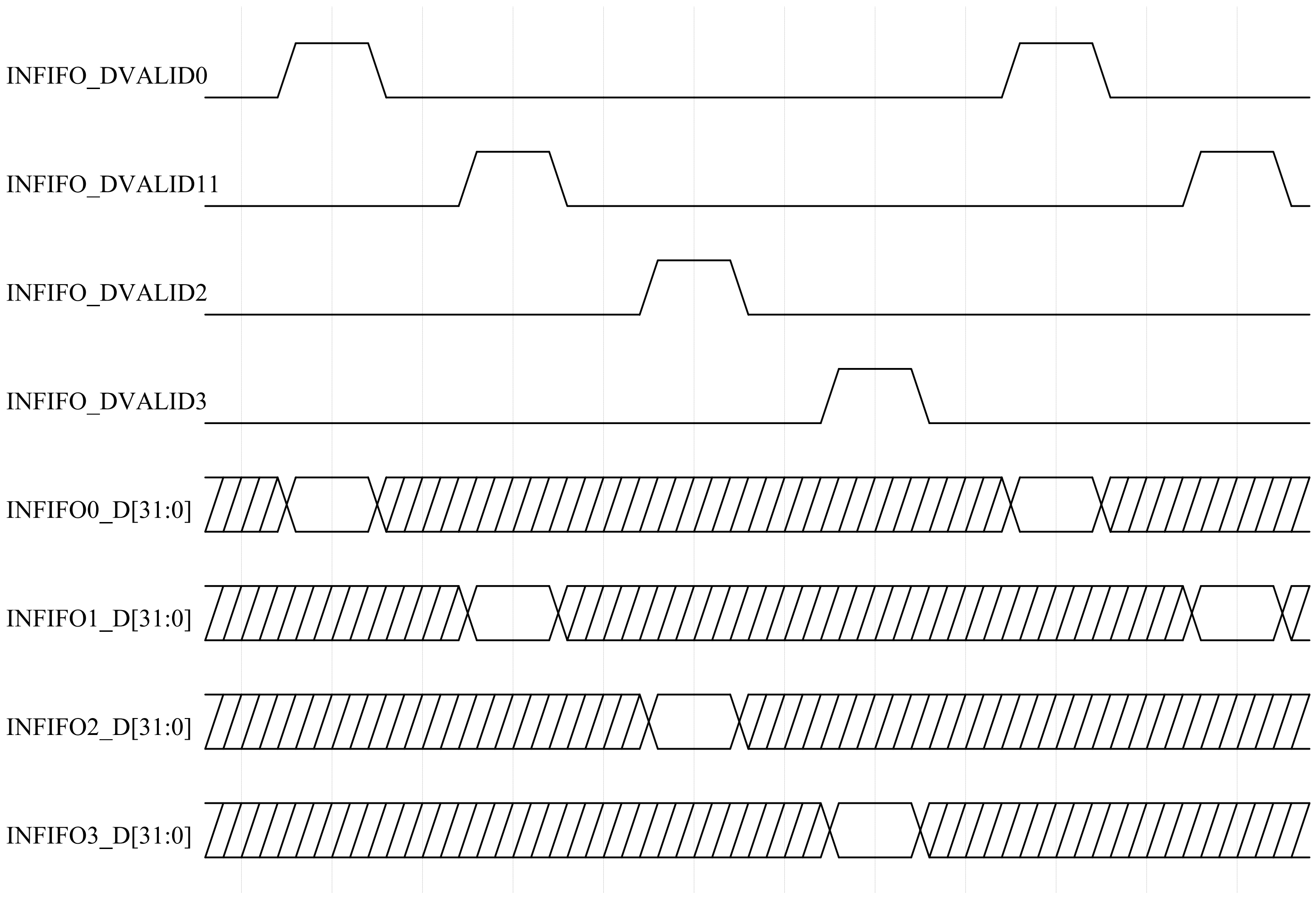
If data is available for reading from two or more FIFOs at the same time, the interface will output one word at a time from each FIFO with one clock cycle of idle time between each data element transfer.

In clock cycle 4 of our example, FIFO 0 still has data available, but data arrives in FIFO1. In this case the interface alternates between FIFO 1 and FIFO 0. Note the clock cycle between the two valid items, imposed by switching the access from one FIFO to another.

As each word of data is presented on each of the data busses, the corresponding data valid signal (`INFIFO_DVALIDn`) is asserted (driven high). As such, the data valid signal can be directly connected to the clock enable or write enable of the next stage of data processing.

In this example the interface outputs `INFIFO_SINGLEn` and `INFIFO_BURSTn` are not shown. These signals are provided for making decisions about when to read from each of the external FIFOs. For this example, we want to read from each FIFO as soon as data is available, and so the read requests have been constantly asserted. As there is no decision making required as to when to assert a read request signal, the SINGLE and BURST signals are ignored.

Example 4: Constant Data Input from Multiple FIFOs



In this example the read requests for FIFOs 0, 1, 2 and 3 are all asserted. In addition to this, the external FIFOs contain a large amount of data for all four FIFOs. In this situation where the interface is able to transfer data from several FIFOs all at the same point in time, the interface automatically transfers one word at a time from a FIFO and then switches to another FIFO.

The rotation around FIFOs is based on a round-robin scheme. If all six FIFOs are ready to transfer data and if all six read requests are asserted, the sequence will be 0, 1, 2, 3, 4, 5, 0, 1, 2 , 3, 4, 5 and so on.

As the data from the external HERON FIFOs is transferred over a shared 32-bit data bus, each FIFO must have a unique output enable to prevent contention. When switching from one FIFO to the next, there must be one idle cycle where the output enable of the first FIFO becomes de-asserted while the output enable of the second becomes asserted. This results in one idle cycle between the assertion of the data valid signals for two different FIFOs.

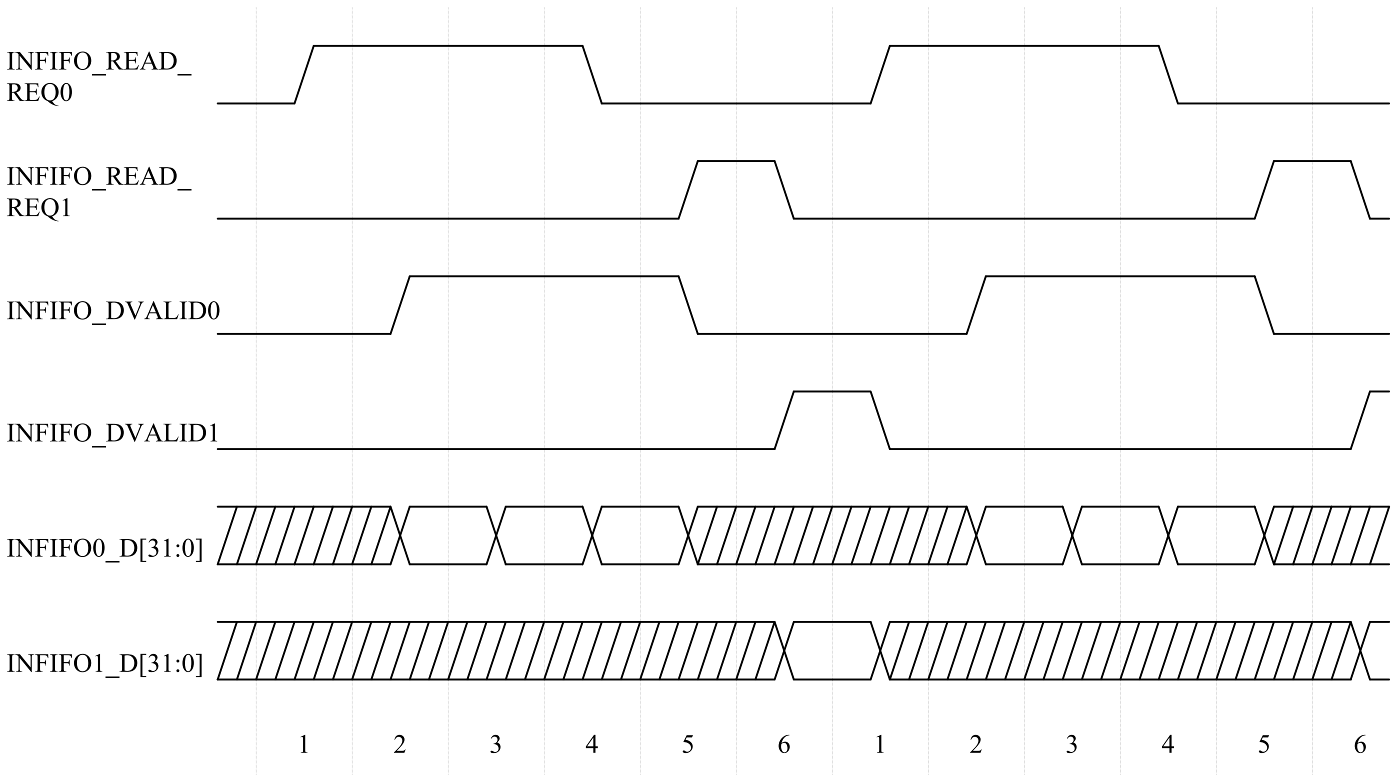
For a 100MHz HERON FIFO interface, the maximum data rate is 400Mbytes/sec (100MHz x 4 bytes). However, with one idle cycle per word as is the case for this example, the overall available bandwidth becomes 200Mbytes/sec. To find the bandwidth limit for each FIFO in use, this figure must be divided by the number of active FIFOs.

For this example, the maximum data rate that can be sustained for each FIFO is 50Mbytes/sec (200Mbytes/sec divided by 4).

When the combined date rate of all of the active FIFOs is greater than 200 Mbytes/sec, or when one FIFO requires a transfer rate of greater than 200/N Mbytes/sec (where N is the number of active FIFOs) you must use a scheme that does not continuously assert all of the read request signals.

In these cases you will need to actively control the assertion and de-assertion of the read request signals in order to control how the bus bandwidth is shared between FIFOs. The following examples show how this is done when using the Six FIFO Read Interface.

Example 5: Read Interface Bandwidth Sharing



In this example there are two active FIFOs, FIFO 0 and FIFO 1. The required data rate for FIFO 0 is 200Mbytes/sec and the required data rate for FIFO 1 is 50Mbytes/sec.

The read request signals for FIFOs 2 to 5 are de-asserted (driven low). The data rate through external FIFOs 0 and 1 is high enough to keep data constantly available for reading.

If we were to tie the read request signals high (asserted) for both FIFOs, this would result in one word transferred alternately from each FIFO, with one idle cycle in-between each word of data. For a 100MHz HERON FIFO connection, this would automatically reduce the maximum data rate from 400Mbytes/sec to 200Mbytes/sec to account for the idle cycle. This would result in a data rate of 100Mbytes/sec for FIFO 0 and 100Mbytes/sec for FIFO 1. For this example the data rate for FIFO 1 can be met with 50Mbytes/sec to spare, but the data rate for FIFO 0 cannot be met.

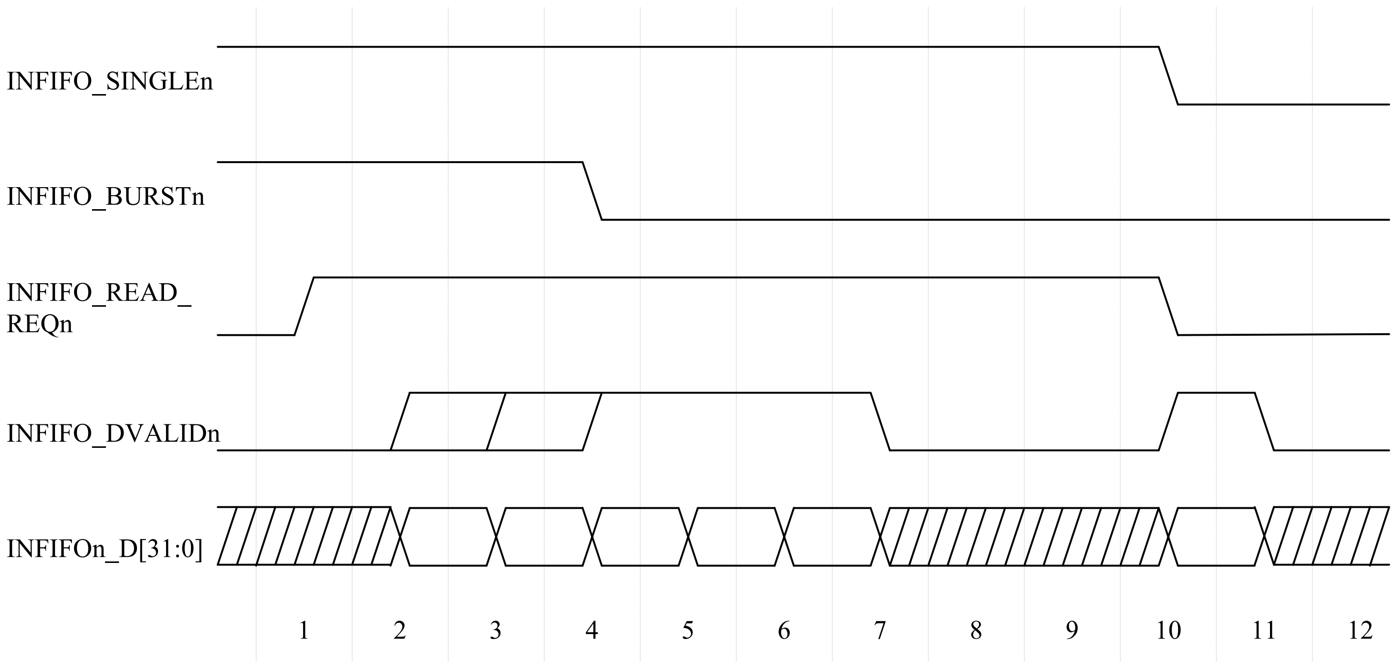
By asserting the read request for FIFO 0 longer than the read request for FIFO 1 the bandwidth can be split such that FIFO 0 is favoured over FIFO 1. In this example, the signal `INFIFO_READ_REQ0` is asserted for 3 consecutive clock cycles and the signal `INFIFO_READ_REQ1` is only asserted for one cycle. This creates a pattern that is repeated every six clock cycles as shown above.

The resulting bandwidth for each FIFO is now calculated as follows. For FIFO 0, 3 words are read in each 6-clock cycles. This gives a data rate of 3/6 * 400Mbytes/sec, or 200Mbytes/sec. This meets the required bandwidth for FIFO 0. For FIFO 1, 1 word is read every 6-clock cycles, giving a data rate of 1/6 * 400, or 66Mbytes/sec. This exceeds the bandwidth required for FIFO 1 of 50Mbytes/sec.

To reach a total bandwidth of close to the 400Mbytes/sec maximum, the sizes of the blocks taken from each FIFO will need to be made larger. This has the effect of reducing the number of idle cycles per active cycle.

For example if we change our choices above to use 6 words from FIFO0 then 2 words from FIFO1, we can see the resulting bandwidths become 6/10*400 = 240Mbytes/sec and 2/10*400 = 80Mbytes/sec.

Example 6: Read Interface Signal Timing



This example demonstrates the signal timing for the read interface. At the start of the diagram the signals `INFIFO_SINGLEn` and `INFIFO_BURSTn` are both asserted. This indicates that there is enough data to provide a constant stream on the bus `INFIFOn_D`.

The data path from the external FIFO to the data bus `INFIFOn_D` contains a pipeline. This pipeline can contain up to two words of data in addition to the data in the external FIFO. When the read request signal `INFIFO_READ_REQn` is first asserted, the data that is presented may come from the two-word buffer internal to the HE_RD_6F interface, or it may come directly from the external FIFO.

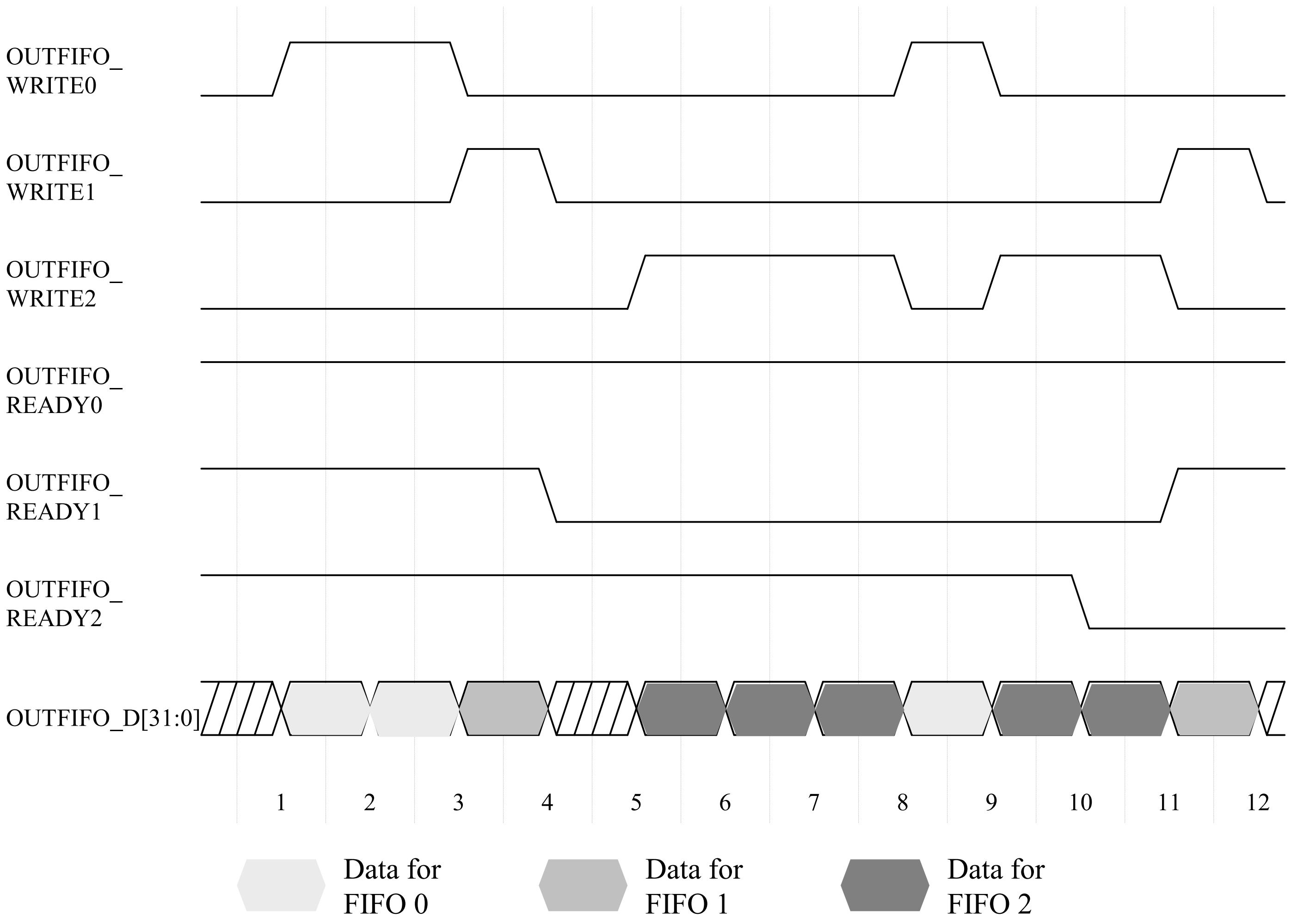
If there is data already in the pipeline, then as the read request is asserted in clock cycle 1, data becomes presented along with the assertion of `INFIFO_DVALIDn` in clock cycles 2 and 3. If the pipeline is empty however, the first data is data read from the external FIFO and this data is presented from clock cycle 4 onwards.

While the signal `INFIFO_BURSTn` is asserted data can be output by the read interface on consecutive clock cycles, as long as the `INFIFO_READ_REQn` signal remains asserted. When the burst signal first becomes de-asserted data will continue to be output for the next three clock cycles as shown in cycles 4 to 6.

While the burst signal remains de-asserted and the single-word available signal, `INFIFO_SINGLEn`, is asserted, one word of data will be output every 4 clock cycles (assuming no activity on any other FIFO).

The data valid signal follows the read request signal by one clock cycle, according to whether data is available either in the two-word internal buffer or the external FIFO. That is, from the first assertion of `INFIFO_READ_REQn`, the first assertion of `INFIFO_DVALIDn` will be no earlier than one clock cycle later. Also, from the de-assertion of `INFIFO_READ_REQn`, `INFIFO_DVALIDn` can only stay asserted for one more cycle.

Example 7: Writing to Three FIFOs



In the above diagram, data is being written to FIFOs 0, 1 and 2. Data can only be written to each FIFO when the respective ready signal is asserted (driven high). Data can only be written to one FIFO at a time, although data transfer can switch immediately from one FIFO to the next without needing any time for idle cycles.

The interface provides one 32-bit input for the data to be written to each FIFO. This data bus must be correctly driven on each clock cycle where any one of the write signals is asserted (driven high).

In the diagram, two words of data are consecutively written to FIFO 0, immediately followed by one word written to FIFO 1. Following the write to FIFO 1, the ready signal becomes de-asserted for that FIFO. When the ready signal is de-asserted only one more word can be written. In this example, data output for FIFO 1 stops until the ready signal is re-asserted in clock cycle 11.

Later on, writing data to FIFO 2 results in the `OUTFIFO_READY2` signal becoming de-asserted in clock cycle 10. When this happens one more word is written and then data output to FIFO 2 is stopped.

Appendix: Active Read Request Algorithms

When dynamically asserting Input FIFO 'read request' signals to share Input FIFO bandwidth, it is up to the user application to implement a scheme that meets the needs of that particular design. The scheme that you decide on when doing this is entirely up to you. However, what is presented here is a simple way of implementing a fair scheme that correctly shares the bandwidth according to the requirements of each Input FIFO connection.

Remember valid data is only transferred in cycles where the relevant `INFIFO_DVALID` signal is asserted. The scheme we show below controls the read request signals in a manner that will allow the bandwidths we require. The actual flow of data will be controlled by the availability of data in the FIFOs.

Calculation Example A

Let us assume we want to share the 400Mbytes/sec Input FIFO bandwidth amongst three FIFOs as follows:

FIFO 0 => 175 Mbytes/sec

FIFO 1 => 150 Mbytes/sec

FIFO 2 => 25 Mbytes/sec

Step i. First calculate the total bandwidth requirement.

Bt = 175 + 150 + 25 = 350 Mbytes/sec

Check that it is less than 400 Mbytes/sec and greater than 200Mbytes/sec. If our total bandwidth requirement is greater than 400 Mbytes/sec, we cannot proceed as we are exceeding what the total available bandwidth. If the total is less than 200Mbytes/sec then there is no need to dynamically assert the FIFO read request signals. In this case, simple tie all required read request signals high.

Step ii. Calculate the number of idle cycles for one loop of all FIFOs, and the maximum amount of idle time that we can allow.

Ci = Number of FIFOs = 3

Ti = Total Available Bandwidth – **Bt** = 400 – 350 = 50 Mbytes/sec.

Step iii. Calculate the minimum number of cycles, including data transfer cycles and idle cycles we require for one loop of all FIFOs.

$$\text{Cmin} \geq \frac{\text{No. of Idle Cycles } (\textbf{Ci}) \ * \ \text{Total Available Bandwidth}}{\text{maximum allowed idle time}}$$

Cmin ≥ (3 * 400) / 50

Cmin ≥ 24

Step iv. Weight each FIFO, relative to the smallest requirement (in this case, 25 Mbytes/sec).

Smallest requirement -> FIFO 2 -> 25 Mbytes/sec.

FIFO 0 requirement -> 7 * FIFO 2.

FIFO 1 requirement -> 6 * FIFO 2.

Therefore **W0** = 7, **W1** = 6, **W2** = 1.

Step v. Perform iterations of 'N' to calculate how many cycles to allocate to each FIFO. This is done by multiplying N against the 'weight' of each FIFO, and adding the number of idle cycles that are required. This total must be greater than or equal to the figure that was calculated for Cmin.

Iteration 0 : **N**=1 **Ct** = (**N** x **W0**) + (**N** x **W1**) + (**N** x **W2**) + **Ci**
Ct = (1 x 7) + (1 x 6) + (1 x 1) + 3
Ct = 17 cycles.

!! In this case, our total is less than Cmin. Therefore we must increase N by one, and try again.

Iteration 1 : **N**=2 **Ct** = (2 x 7) + (2 x 6) + (2 x 1) + 3
Ct = 14 + 12 + 2 + 3
Ct = 31 cycles.

In this case, Ct ≥ Cmin, which is what we need. Therefore we now know how many cycles to allocate each FIFO as follows:

FIFO 0	Transfers data for **14** consecutive cycles
IDLE	**1** idle cycle performed to change from FIFO 0 to FIFO 1.
FIFO 1	Transfers data for **12** consecutive cycles
IDLE	**1** idle cycle performed to change from FIFO 1 to FIFO 2.
FIFO 2	Transfers data for **2** consecutive cycles
IDLE	**1** idle cycle performed to change from FIFO 2 to back to FIFO 0.

Step v. We can check the FIFO allocation we have obtained by calculating the bandwidth each FIFO will have, as follows.

We will loop around all FIFOs every **31** cycles.

For FIFO 0, we will transfer **14** words, every **31** cycles.
Therefore FIFO 0 bandwidth is (14/31) * 400 = **180.65** Mbytes/sec.
This is greater than our requirement of 175 Mbytes/sec, so this has been satisfied. ✓

For FIFO 1, we will transfer **12** words, every **31** cycles.
Therefore FIFO 1 bandwidth is (12/31) * 400 = **154.84** Mbytes/sec.
This is greater than our requirement of 150 Mbytes/sec, so this has been satisfied. ✓

For FIFO 2, we will transfer **2** words, every **31** cycles.
Therefore FIFO 2 bandwidth is (2/31) * 400 = **25.81** Mbytes/sec.

This is greater than our requirement of 25 Mbytes/sec, so this has been satisfied.✓