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

Connecting the PowerPC Processor to Hardware - PowerPC Example B

v1.1 R.Williams 25-04-05

Xilinx Virtex-II Pro FPGAs provide one or more embedded PowerPC processor cores along with the usual array of Virtex-II FPGA technology including logic slices, dedicated multipliers and Block RAM.

Combining a PPC processor core with the Virtex-II FPGA fabric brings a huge amount of flexibility – the possible uses are numerous. However an interesting design challenge is created. How can the processor and FPGA be used together in a way that makes the best use of the power of each element?

In answering this puzzle it is first necessary to understand how FPGA logic can be connected to the many PowerPC interfaces, with software for the CPU interacting with VHDL written for the FPGA.

However, knowing this is not enough. The PowerPC provides many different interfaces that are designed for use in different ways. How these different interfaces are used will affect the overall performance of the system. The right balance needs to be achieved between the workload of the PowerPC and the workload of the FPGA.

This document discusses these issues through the use of an example that connects a hardware FFT to the PowerPC, so that it can be used as a hardware acceleration block. An example project is provided that combines the PowerPC and FPGA by implementing a CoreGen FFT component in the FPGA. The PowerPC feeds the FFT component with data and also controls and monitors the operation of that logic.

The example provides a framework that can be used to implement many other different types of FPGA function block. The use of a CoreGen FFT component is simply an example of one such type of function block that may be connected to the PowerPC.

This document then goes on to discuss how system performance is influenced by the different design choices that can be made. Alternative strategies are presented along with the performance results that can be expected for each case.

History

Rev 1.0	First written
Rev 1.1	Updated to use ISE 7.1

Other Relevant Documents

In addition to the Xilinx provided documentation for the Virtex-II Pro, and the software tools that go with it, there is another HUNT ENGINEERING document that we assume you have read before this document. This is the "Getting Started with the Embedded PowerPC" document, also referred to as PowerPC Example A. That document works through the design flow for making a design that uses the PowerPC, and how to compile and load a PowerPC program. These subjects will not be covered again in this document.

Choices of interface to the PowerPC

The PowerPC core in the silicon of the Virtex-II Pro FPGAs offers five interface buses. Conventional processors have an external memory interface bus, and the equivalent to that in the V-II Pro is the Processor Local Bus (PLB). The 64-bit PLB is intended for high performance accesses to peripherals like SDRAM.

From the PLB there is a bus bridge component that offers an On-chip Peripheral Bus (OPB). This 32-bit bus is intended for slower peripherals that can be accessed without stalling the PLB. It is also possible to connect peripherals like SDRAM to this interface, the advantage of that seems to be that it is a simpler interface and consumes less FPGA resource. This almost certainly comes at the price of lower speed.

Both of those interfaces are coupled together, so any accesses on either one, will affect the accesses on the other.

Then there is the Device Control Register (DCR) bus, intended for accessing peripheral control registers. This is not intended for data transfers and does not appear in the processor memory map. Rather it is accessed through a special register and instruction.

The PPC core also offers two On Chip memory interfaces, one on the Data side (DSOCM), and the other on the instruction side (ISOCM). Here Block RAMs from the FPGA can be connected as if they were on-chip memory of the processor. These interfaces allow you the designer to customise the amount of on chip instruction and data memory according to your architectural requirements.

The major problem associated with using conventional processors in a Real Time system, is that the (usually single) memory interface must be used not only for accessing the off chip memory, but also for accesses to devices like HERON FIFOs for system data flow. That causes interaction between the accesses for instructions and program data, and the data flow that must be guaranteed for real time operation.

The architecture of the PowerPC core allows us to decouple the two things, if we choose to connect the memory etc to the PLB/OPB interfaces, and to use the DSOCM for accesses to the HERON-FIFOs. This is made possible by the fact that Block RAMs are dual ported, and only one side is connected to the DSOCM.

Once you consider that FPGA logic can make accesses to the "on chip" memory of the processor without being affected by the instruction or program data accesses, other possibilities arise, such as using hardware accelerator blocks (implemented in FPGA gates) that can operate on data stored in DSOCM, writing the results back to DSOCM.

The primary intention of this document is to explain how to utilise this part of the architecture, and as an example an FFT core generated using Core Gen is connected to the DSOCM.

Connecting FPGA Function Blocks to the PowerPC

The Xilinx Embedded Design Kit (EDK) provides many components that can be quickly and easily placed in the Xilinx Platform Studio tool. These components include for example, PLB or OPB interfaces to external FLASH memory, I2C, Ethernet, and UART.

To support non-standard, or uncommon interfaces connected to the PLB/OPB it also provides general-purpose components that allow specialist blocks to be easily connected to the PowerPC. The connection to the PowerPC is taken care of by that block leaving you to complete the interface in your own VHDL design.

Using Block RAM to Interface FPGA logic and the PowerPC

The Virtex-II Pro FPGA provides dedicated memory blocks called Block RAM. The Block RAMs within the FPGA can be connected to any of the main bus interfaces of the PowerPC. Xilinx provide components for interfacing Block RAM to the PLB bus, the OPB bus, ISOCM and DSOCM interfaces.

Each Block RAM is a dual port memory. This allows one port of the memory to be connected to the PowerPC core (via PLB, OPB or OCM) and the other port to be connected to the FPGA. In this way, the PowerPC can be transferring data in or out of the Block RAM at the very same time that FPGA is also transferring data in or out of the other port.

When implementing a connection between the PowerPC and a function unit in the FPGA there are two main design points that must be considered.

Firstly a Block RAM 'module' must be correctly connected in the system such that one port is accessed by the chosen bus format of the PowerPC and the other port is accessed by FPGA logic.

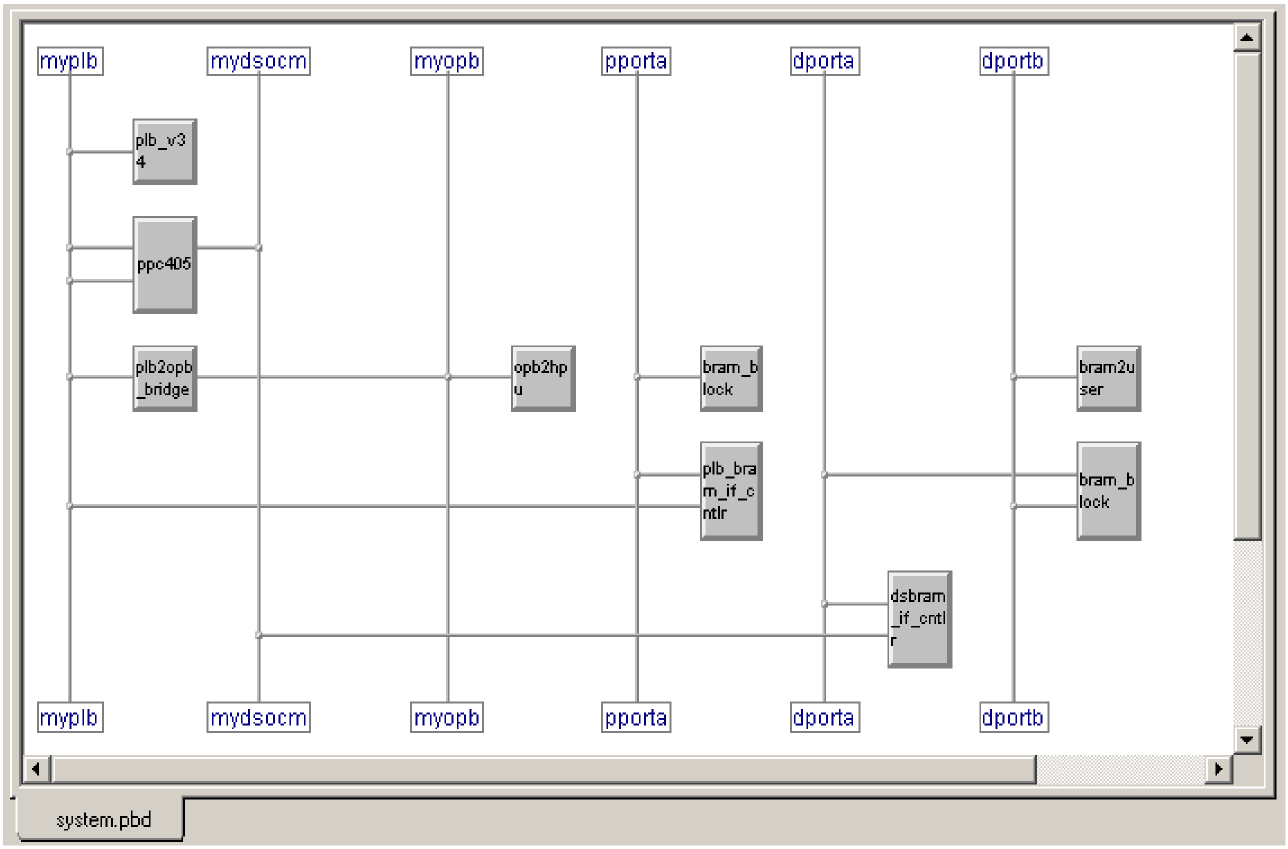
Secondly, a data synchronisation module must be created that controls access to that Block RAM such that each port is only accessed when it is correct to do so.

Defining the 'PowerPC to Function' Block RAM

When creating a processor and FPGA design for a Virtex-II Pro HERON-FPGA module both the EDK Platform Studio tool and the Project Navigator tool will be used. Platform Studio provides a means to control and specify how the PowerPC is used and what interfaces are connected to it, and Project Navigator manages the FPGA building process that combines all of the design elements into a single bitstream.

Block RAM can be specified either in Platform Studio (for use by the PowerPC) or Project Navigator (for use by the FPGA logic). It is thus not obvious which tool to use to define a Block RAM that interfaces between the two. The correct place for the Block RAM to be defined in this instance is Platform Studio. This is because in Platform Studio a specific component exists ('bram_block') which allows easy connection to any of the main interfaces of the PowerPC. The 'bram_block' component provides a variable amount of Block RAM according to the memory range specified when the component is placed in the design.

The component is placed in a 'Block Diagram' using a graphical schematic tool. Connections are drawn as wires between the Block RAM component and the PowerPC interface it is connected to. An example block diagram is shown below.



In this block diagram we have connected Block RAM via PLB and Block RAM via DSOCM. In the above diagram, the Block RAM on PLB is being used for storage of program code and data. The Block RAM on DSOCM will be used to interface to the FPGA logic.

The Block RAM placed on DSOCM has a 'dsbram_if_cntlr' to interface Block RAM to the PowerPC. The bus 'dporta' is used for Port A of the Block RAM and is connected between the 'bram_block' module and the 'dsbram_if_cntlr' module. The bus 'mydsocm' is connected between the 'dsbram_if_cntlr' and the processor core.

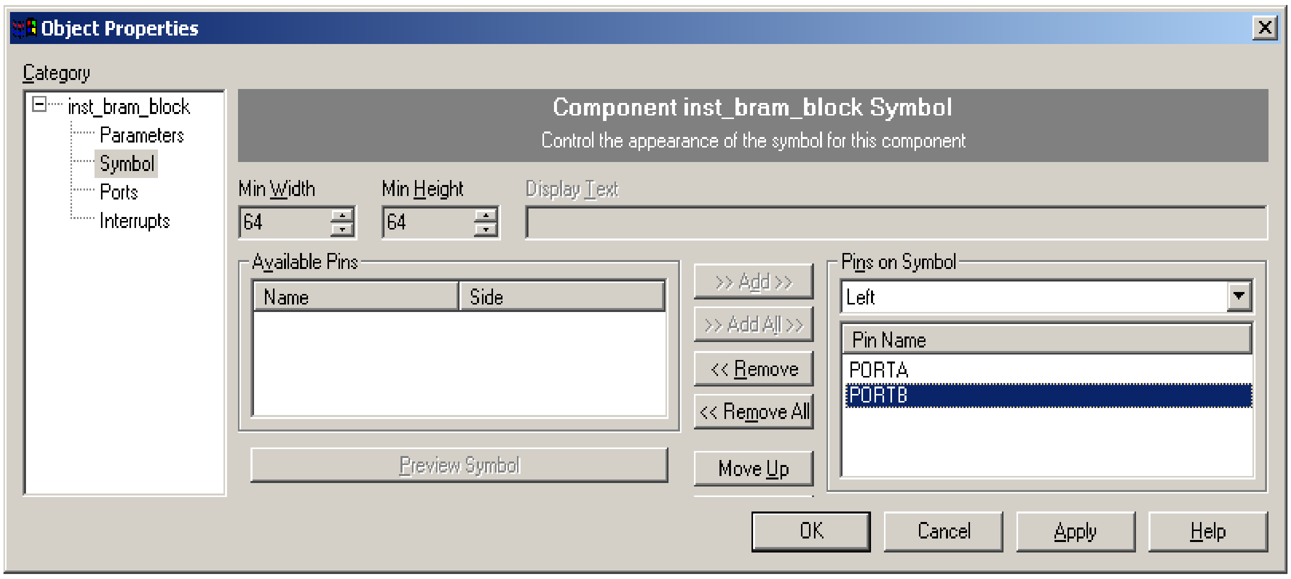
When connecting Block RAM in this way, the amount of Block RAM used is defined via the memory range specified for the 'dsbram_if_cntlr' module. Users should be aware that the larger the amount of Block RAM synthesized, the slower the DSOCM interface will run. This issue is explained in some detail in the EDK document 'PowerPC 405 Processor Block Reference Guide'.

The DSOCM interface is a good choice for connecting our interface Block RAM as it provides a dedicated high performance connection to the processor. The OCM interfaces of the PowerPC are particularly tailored for connecting to Virtex-II Block RAM. Unlike PLB and OPB, the OCM interfaces are not buses that are shared amongst several resources. There is no arbitration, just a straightforward connection between processor memory space and Block RAM.

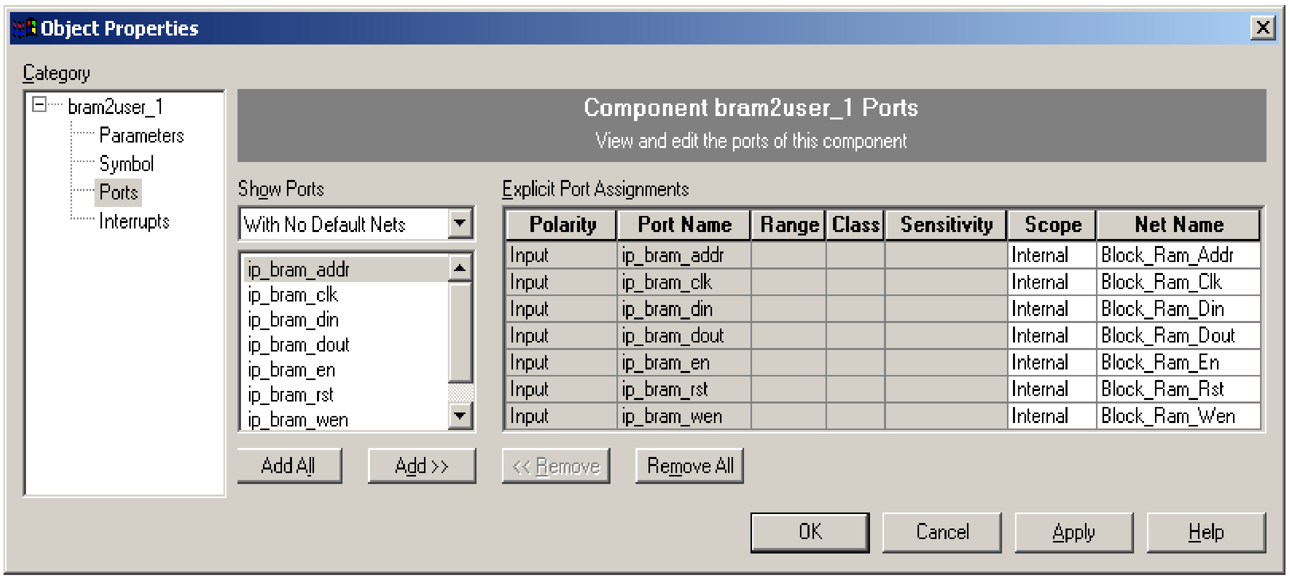
On the other side of the Block RAM we want to be able to make our own accesses using VHDL that we provide. To do that a component must be placed to present Block RAM signals that the FPGA logic can connect to. The 'bram2user' component has been supplied by HUNT ENGINEERING to do just this. It has a 'bus' like connection named Port-B that is provided for making a connection to Port B of the Block RAM module.

To be able to connect to Port B of the Block RAM we need to make Port B visible on the Block RAM component 'bram_block' in the Block Diagram. This is done by right clicking on the Block RAM component and selecting properties. Then click on the symbol view within the properties window.

In the Available Pins window is the port that we need to make visible 'PORTB'. Select this port and click '>>Add>>' to add to any chosen side of the component. The side can be chosen using the select box just below 'Pins On Symbol'. Shown below is a Block RAM block with Port B made visible on the left side of the symbol.



With the Port B pin of the Block RAM now visible, a Transparent module must be placed to form the bus connection between the Block RAM component and the 'bram2user' component. A bus name box can be added to this transparent component, to apply the name 'dportb'.



Polarity	Port Name	Range	Class	Sensitivity	Scope	Net Name
Input	ip_bram_addr				Internal	Block_Ram_Addr
Input	ip_bram_clk				Internal	Block_Ram_Clk
Input	ip_bram_din				Internal	Block_Ram_Din
Input	ip_bram_dout				Internal	Block_Ram_Dout
Input	ip_bram_en				Internal	Block_Ram_En
Input	ip_bram_rst				Internal	Block_Ram_Rst
Input	ip_bram_wen				Internal	Block_Ram_Wen

The ports of the 'bram2user' component are shown above. The 'bram2user' component takes the Port-B bus and splits it out to all of the individual signals that make up a single port of Block RAM. These individual signals are then presented as ports of the PowerPC system level 'design'. The PowerPC 'design' is the top level inside the Platform Studio tool and becomes a sub-level placed below the User-Ap entity in the Project Navigator project.

The 'bram2user' component is provided in the 'pcores' directory of the example project. This component can be used in every system design where the second port of a Block RAM must be presented to the FPGA logic for direct accessing.

In order to user the 'bram2user' component in a new project, the directory 'bram2user' found in the 'pcores' directory of the example must be copied into the 'pcores' directory of your new project. When this has been done Platform Studio will make the new component available for use the first time the project is opened with the 'bram2user' component in the pcores directory.

With the 'bram2user' component in place as described, there now exists a connection for 32-bit data between the processor core and the FPGA logic via that Block RAM. What remains is to create a data synchronisation module that can be used to provide control between the Hardware block and the processor.

Defining the 'PowerPC to Function' Synchronisation Module

The exact needs of a control module will be dependant on the Hardware connected to the Block RAM and the way that it is used in your system.

As part of the example, HUNT ENGINEERING have provided a module to handle data synchronisation between the PowerPC and FPGA logic. This module is named 'opb2hpu' and can be seen in the block diagram shown earlier in this document. It has been created using one of the general purpose interface components that Xilinx provide, customised by HUNT ENGINEERING adding VHDL.

As the name suggests the component provides an interface to OPB. The letters 'hpu' indicate a 'Hardware Processing Unit', so in fact the module is an OPB to HPU interface module.

The function of the 'opb2hpu' module is to control data access to the Block RAM. The internal logic of this module has been kept as general purpose as possible and provides an example of how to manage the key issues of data synchronisation between PowerPC and FPGA based hardware-processing unit.

This module can be used as a starting point as it is, or after modification, for many different types of data synchronisation. To use this module in your own project the 'opb2hpu' directory of the 'pcores' directory of the example must be copied into your own project 'pcores' directory. With this done, the file 'user_logic.vhd' can be modified according to your own particular needs.

The example 'user_logic' VHDL provides a function to inform the hardware processing unit that it should start to process data and a function to allow the PowerPC to detect that the processing has finished. An interrupt mechanism is also provided for those users who wish to interrupt the processor on completion of the FPGA hardware-processing unit.

One of the functions of the 'opb2hpu' component is to handle clock domain synchronisation between processor clock domains and FPGA clock domains. On the processor side, the component runs from the bus clock of the bus interface it is connected to, in this case the OPB clock signal is used. On the FPGA side, a clock signal must be provided by the FPGA logic. This FPGA clock signal is used to correctly control operation on the FPGA side. Any interaction between the two different clock domains must be correctly handled by the logic inside the 'opb2hpu' component.

The user logic VHDL creates an output signal called 'hpu_start'. This signal is asserted (set high) by a processor access over OPB and in our example is used to indicate that there is one complete block of data in the Block RAM ready to be processed. When the FPGA detects the assertion of the 'start' signal, the hardware processing is started.

When the calculation has completed and the results written to the Block RAM, the FPGA indicates completion using the 'user_logic' input signal 'hpu_done'.

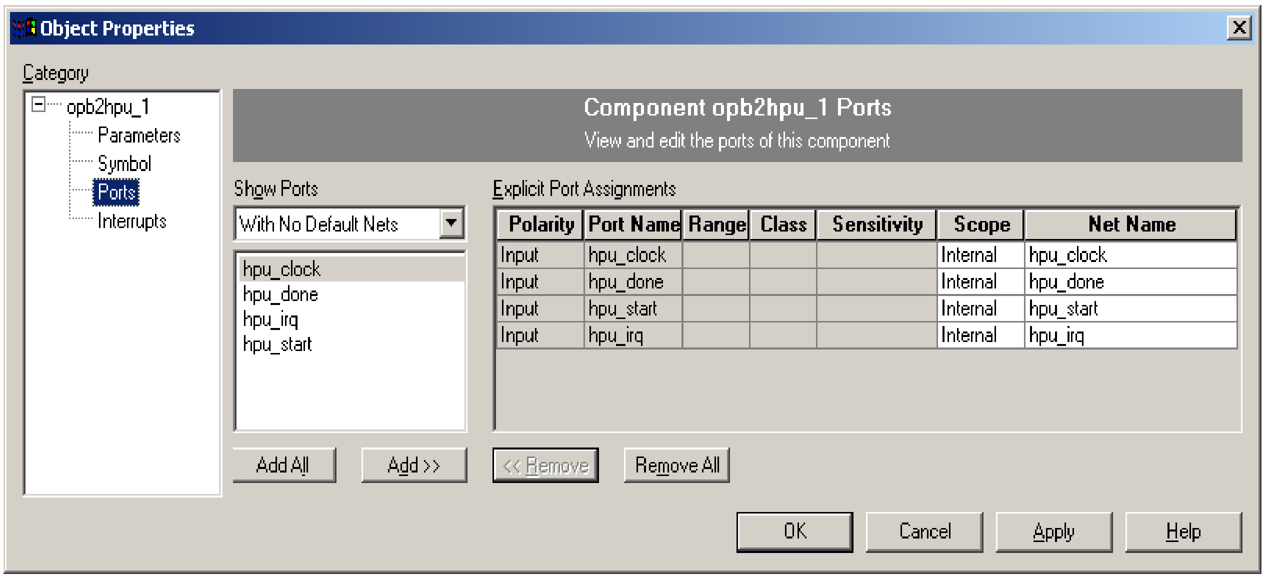
The PowerPC can detect this completion in two ways. Firstly, the processor can poll a bit in a register accessed over OPB in order to detect when the calculation and unloading has completed. Secondly, an interrupt can be enabled and on completion the interrupt output signal 'hpu_irq' will be asserted by the user logic. If the HPU interrupt signal is correctly connected to the processor and configured for use, an interrupt will occur to indicate completion.

Using the OPB 2 HPU Synchronisation Module

The 'opb2hpu' module connects to the OPB interface of the PowerPC. In order to use OPB in your system design, a PLB connection must be directly made to the processor core and a PLB2OPB bridge must be used to connect to an OPB bus.

The 'opb2hpu' module has a slave OPB port that must be connected to the OPB bus. The required connections can be seen in the block diagram presented earlier in this document.

In addition to the slave OPB port, the 'opb2hpu' module also presents two input signals and two output signals that must be connected appropriately. The 'hpu_clock' and 'hpu_done' input signals must be driven by FPGA logic, and the 'hpu_start' output signal must be used as an input to FPGA logic. The signal 'hpu_irq' must be connected within the PowerPC system so that it drives either the critical or non-critical interrupt pin of the processor core. In the example, the non-critical interrupt pin is driven by 'hpu_irq'.



Component opb2hpu_1 Ports

View and edit the ports of this component

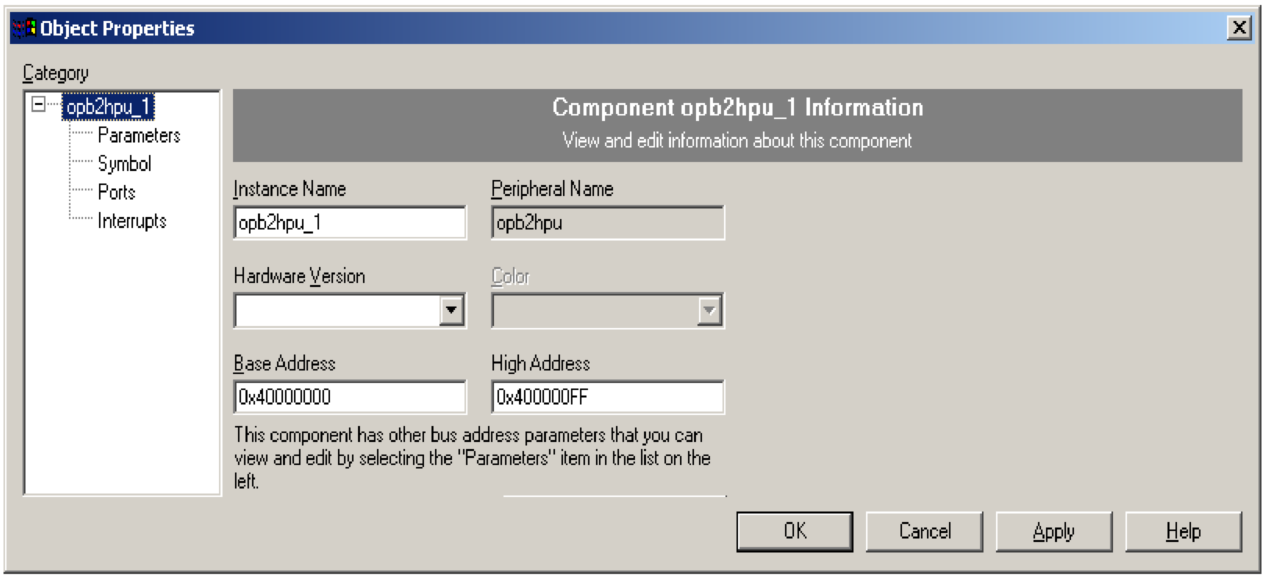
Show Ports: With No Default Nets

hpu_clock
hpu_done
hpu_irq
hpu_start

Explicit Port Assignments

Polarity	Port Name	Range	Class	Sensitivity	Scope	Net Name
Input	hpu_clock				Internal	hpu_clock
Input	hpu_done				Internal	hpu_done
Input	hpu_start				Internal	hpu_start
Input	hpu_irq				Internal	hpu_irq

The 'opb2hpu' module must have an address range specified before the Platform Studio project can be built. The module only needs a very small address range that is at least 16 bytes (4 words) in size. In our example a slightly larger range has been supplied to reduce the address decoding logic that will be generated. The example uses the address range 0x40000000 to 0x400000FF.



Component opb2hpu_1 Information

View and edit information about this component

Instance Name	opb2hpu_1
Peripheral Name	opb2hpu
Hardware Version	
Color	
Base Address	0x40000000
High Address	0x400000FF

This component has other bus address parameters that you can view and edit by selecting the "Parameters" item in the list on the left.

The address map of the 'opb2hpu' module is shown below. Please be aware, the address offsets are specified in 32-bit word steps, while the Example Address reflects the physical byte address for each function.

Word Address Offset	Example Address	Function
0x00	0x40000000	HPU Start Control and Status
0x01	0x40000004	HPU Interrupt Enable
0x02	0x40000008	HPU Interrupt Clear

At address offset 0x00 there is a register location used for asserting the start signal and a register used for returning status information. A write to address offset 0x00 of any value will assert the 'hpu_start' signal. Once this signal is asserted it will remain asserted until cleared by the assertion of 'hpu_done'.

Reading from address-offset 0x00 will return three bits of HPU status information as shown in the table below.

HPU Status Bit	Function
0	HPU Interrupt Enable
1	HPU Active
2	HPU Interrupt Status

The HPU Interrupt Enable bit returns the state of the Interrupt Enable. If this bit is high interrupts are enabled, and if this bit is low interrupts are disabled.

The HPU Active bit is high while the hardware-processing unit is active. When the 'hpu_start' signal is asserted HPU Active becomes asserted and remains asserted until cleared by the 'hpu_done' signal being set high. The HPU Active bit can be polled by PowerPC software to detect when the calculation and unloading stages have completed.

The HPU Interrupt Status bit reflects the state of the interrupt condition in the hardware-processing unit. While 'hpu_irq' is asserted this bit will return 1.

To enable interrupts an access must be made to address offset 0x01 with bit 0 set to 1. To disable interrupts, the access should bit 0 to 0. The interrupt enable defaults to 0 when the system is reset.

When the HPU interrupt has become asserted an access to address offset 0x02 can be used to clear the interrupt condition. Any interrupt service routine servicing the HPU interrupt must perform a write access to this address with any data value to clear the interrupt condition.

<u>**Modifying the OPB 2 HPU Module**</u>

The OPB2HPU module provides a starting point for many different types of synchronisation between PowerPC and FPGA logic. In our example the amount of data transferred in the Block RAM is unknown to the user logic VHDL of 'opb2hpu'. In fact the user logic does not need to know. The PowerPC program and FPGA logic both work together on groups of 256 words. A different function size could be used by changing the FPGA logic and PowerPC program, without redesigning the OPB to HPU component.

In applications where the transfer size dynamically changes, a register could be written from each side to indicate the amount of data that exists. You must remember when doing this that it is very important to correctly synchronise the reading and writing of the data count in consideration of the two different clock domains that exist within the module.

Examples of Using this Technique

A Core Gen FFT as a hardware acceleration unit

As an example, HUNT ENGINEERING have provided a project that uses a Core Gen FFT block as a Hardware acceleration block.

The example uses a DSOCM block RAM to provide data for and to receive the results from the hardware FFT block. The opb2hpu block is used to allow the PowerPC program and the Hardware FFT to be synchronised.

The FFT component that has been generated and used in the example is a Xilinx '256-point Radix-2 Minimum Resources' FFT. The same FFT CoreGen component can also be built as a 'Radix-4 Continuous I/O' or 'Radix-4 Burst I/O' version. Each of these three versions will provide a different FFT calculation time and will use a different amount of FPGA resources. The version used in the example is as the name suggests the version that uses the least resources. It is also the slowest version to complete a calculation of one 256-point transform.

In fact the XC2VP7 used on the HERON-FPGA9 does not have enough resources to fit the other types of 256 point FFT, because the required footprint for those does not fit alongside the PowerPC.

In the example because a 256-point FFT has been used the PowerPC code understands that before the start signal can be asserted, 256 points of data (256 words) must be written to the Block RAM from address 0 to address 255. With this done, the Processor writes to the control block to start the processing.

The user logic VHDL creates an output signal called 'hpu_start'. This signal is asserted (set high) by the processor access over OPB. When the FPGA logic detects the assertion of the 'start' signal, the FFT processing is started.

When the FFT unit has completed calculation the results must be 'unloaded' from the FFT component. This unload operation is performed by the FPGA logic transferring the results to addresses 256 to 511 of the same Block RAM component used for input data.

When the calculation stage and unload stage have completed the FPGA indicates completion using the 'user_logic' input signal 'hpu_done'.

The PowerPC can detect this completion in two ways. Firstly, the processor can poll a bit in a register accessed over OPB in order to detect when the calculation and unloading has completed. Secondly, an interrupt can be enabled and on completion the interrupt output signal 'hpu_irq' will be asserted by the user logic. If the HPU interrupt signal is correctly connected to the processor and configured for use, an interrupt will occur to indicate completion.

Connecting to HERON FIFOs

The OPB2HPU module can be used to connect to HERON FIFOs inside the FPGA design. When transferring data in and out of the HERON FIFO interfaces it is important to have both the means to pass data and to also have the means to synchronise the flow of data.

Just as with the FFT example, the 'bram2user' and 'opb2hpu' modules can be used to both pass data and synchronise the flow of that data.

The exact implementation of a connection between PowerPC and HERON FIFO is left to the user, as there are many different ways in which this may be done. However, whatever connection method is required, the following points should be considered.

In buffering data between the PowerPC and HERON FIFOs it is best to use Block RAM as a small FIFO like component. Using the HERON FIFOs it would be possible to implement a read-single-word or write-single-word function directly via PLB or OPB. However, to do so would be very inefficient and if the access was not implemented carefully may also hang the PowerPC processor core while the HERON FIFO flags indicate data cannot be read or written.

In order to keep the transfer of HERON FIFO data both efficient and decoupled from the PowerPC, Block RAM should be used to stream data in and out of the processor in blocks larger than a single word.

The actual amount of buffering used will vary with the requirements of the system, but if the block size is too small the overhead of the software accesses to the opb2hpu module will have a considerable effect and reduce overall bandwidth.

Conversely if the amount of buffering is too large there will be long delays between the transfer of each block and a large amount of Block RAM will be used to implement the connection. As the amount of Block RAM used increases the PowerPC to Block RAM clock rate will need to decrease as the amount of Block RAM interfacing logic increases.

However, it should be noted at this point that when connecting to Data Side On-Chip-Memory, there is a minimum Block RAM size of 8KBytes. This is a far larger amount than is needed to interface to one HERON FIFO. In fact when using Block RAM on DSOCM several FIFOs could be interfaced with a single 8Kbyte block. The Block RAM could be divided into separate address areas with each area handling data transfer for one HERON FIFO.

Connecting to HERON FIFOs in this way would require one instance of the module 'bram2user' and one instance of the module 'opb2hpu', along with the appropriate connections to DSOCM, Block RAM and OPB.

The 'opb2hpu' user logic VHDL must be written to automatically transfer data on the FPGA side independently of the processor core. While data transfer is possible to or from a HERON FIFO that transfer should proceed from or to the associated Block RAM address. When one FIFO 'block' size has been transferred an interrupt could then be used to indicate to the processor core that one block of data has been completed. Alternatively a register could be polled by the processor to find out if a block transfer has completed. Whether interrupts or polling is used is down to the system software design, but examples of each mechanism are shown in the FFT example.

In the example provided the FFT output data is mirrored out to HERON Output FIFO 0 as an example of one way of doing the FIFO access.

Performance

In making the example, and measuring the performance, it became obvious that many things affect the performance that can be achieved with the PowerPC. The Core Gen FFT block performance is well known as with other FPGA logic, in this case it is clocked at 100MHz, and is the least resources FFT as discussed above.

Initially the project was built following a Xilinx example for the PowerPC, which was measured by making the design continuously loop around loading a BRAM with input data, running the hardware FFT and detecting that the FFT is complete before starting the loop again. Some signals were put onto digital I/Os of the module, where they could be measured with an Oscilloscope.

In a real system the PowerPC could be running some other code during the FFT processing stage, but for the purposes of this example that was not happening.

The initial results were as follows:

Load the Block RAM	FFT	Detect completion	Total Loop
9.52us	16us	0.48us (polling)	26.0us

This example (PLB200) uses a single Block RAM placed on PLB to hold the instructions and program data. The Caches are not switched on.

In order to increase performance, the first thing that we tried was to enable the caches, which gave the following results:

Load the Block RAM	FFT	Detect completion	Total Loop
5.2us	16us	0.3us (polling)	21.5us

This shows a noticeable improvement to the Block RAM loading time.

The next thing we tried was to use an interrupt rather than polling for the detection of completion. This is the way that would need to be used if the PowerPC was running some other code in parallel with the FFT.

First with the Caches off:

Load the Block RAM	FFT	Detect completion	Total Loop
9.52us	16us	8.68us (interrupt)	34.2us

Then with the Cache on:

Load the Block RAM	FFT	Detect completion	Total Loop
5.2us	16us	0.8us (interrupt)	22.0us

This shows that the interrupt is slower than the polling, which is probably what we expect, but also that the caches make a big difference to the interrupt latency too.

Next we tried increasing the PowerPC CPU clock, from 200MHz to 300MHz.

With Caches off:

Load the Block RAM	FFT	Detect completion	Total Loop
8.6us	16us	0.5us (polling)	25.1us

Which doesn't show the speed improvement that we expect. Next, with caches on:

Load the Block RAM	FFT	Detect completion	Total Loop
5.24us	16us	0.16us (polling)	21.4us

This also does not show the expected speed increase of 3/2 as expected, as the performance is very similar to that of the 200MHz project.

Now with the Interrupts (and caches off):

Load the Block RAM	FFT	Detect completion	Total Loop
8.7us	16us	8.1us (interrupt)	32.8us

Cache on:

Load the Block RAM	FFT	Detect completion	Total Loop
5.2us	16us	0.6us (interrupt)	21.8us

Seeing that the use of the cache makes such a difference, it makes it clear that the memory access times are important to the performance of a PowerPC program. We can try putting the program into the 'On chip memory' to see how that compares.

To do this involves separating the instruction and program data memory spaces, requiring a linker script. With that projects were made that placed the Instructions and Data into the ISOCM and DSOCM. With the processor and OCM running at 200MHz, this brought the following results:

Load the Block RAM	FFT	Detect completion	Total Loop
2.84us	16us	0.26us (polled)	19.1us

The OCM is not cacheable, but it achieves a slightly better result than PLB with the cache on.

We tried to increase the processor clock to 300MHz, leaving the OCM speed at 200MHz, but the tools could not achieve this for us. It was necessary to reduce the clock rate of the OCM to 100MHz to achieve a routed design. Then the results were:

Load the Block RAM	FFT	Detect completion	Total Loop
5.24us	16us	0.36us (polled)	21.6us

Performance Conclusions

The example projects that have been discussed allow certain conclusions to be drawn about the advantages and disadvantages of the different methods of connecting to the PowerPC.

The purpose of this document is not to make specific recommendations as to how you must use the Virtex-II Pro. The exact use of the FPGA and processor core will always vary from system to system and no one approach will suit all situations. What is important is to understand how certain design changes can affect the performance of your system.

In analysing the FFT example we have created ten different sets of result. The results for each combination of system architecture and project settings have been compared by examining the overall loop time for each approach.

The slowest performance of the ten cases is when using PLB for both program code and program data. The PLB is also used when an OPB access is made, as OPB is a bus connected via PLB. In this situation there is a large degree of contention between many types of access all trying to share the same data bus. Increasing the processor core from 200MHz to 300MHz makes little or no performance improvement.

By enabling the use of instruction cache and data cache a significant improvement is made. In the FFT example the use of cache gives an increase in performance by a factor of about two over the situation where there is no cache at all.

A similar performance improvement is also achieved by changing the architecture to use dedicated program memory on ISOCM and DSOCM. The performance increase for the OCM-200 project is almost identical to the projects where cache is used. However as soon as the memory map is split to have separate ranges for code and data, it becomes necessary to use a linker script for controlling section placement. An example linker script has been supplied with the FFT example and further documentation is available from Xilinx and Gnu.

When increasing the CPU clock rate for the OCM projects from 300MHz to 200MHz the performance decreased slightly. This is due to the decrease in the clock rate from 200MHz to 100MHz for the DSOCM and ISOCM interfaces.

Clearly for the FFT example projects, the bandwidth to external memory is more significant than the pure clock rate of the core of the processor. This can be seen from the code required to store the sine wave in FFT connected Block RAM. Of course, this situation is very sensitive to the kind of computation being performed by the PowerPC.

For applications where a large amount of data accessing is performed to load and store general purpose CPU registers it is important to favour the performance of DSOCM over the performance of the core. Conversely, for applications that perform more cycles on CPU registers than on data fetching it is more important to favour CPU clock speed over external memory speed. The correct design choices must therefore be made by considering what kind of processing the PowerPC is to perform.

Finally, for the project build where interrupts were selected the overall performance was notably worse. This is due to the larger amount of code necessary to handle an interrupt service routine when compared to a simple `while` statement that polls a flag, but the use of interrupts brings other advantages to the system.

When creating and using interrupts, for each interrupt taken the processor must first save and then restore the 'context' of the core. That is, the current set of register values in use when the interrupt occurs are first stored to the stack, the interrupt routine is executed, and then those register values and therefore previous machine state are returned from the stack.

Although there will be some software overhead when adding features such as interrupts or thread management, this overhead is often very necessary for the system operation improvements that can be made with that new functionality. The decision for example on whether to include interrupts must be based on the requirements of each particular application.

In summary the key performance points are:

- Using a shared, arbitrated bus interface such as PLB will offer less performance than a dedicated direct connection to memory such as OCM. This situation will be made worse where the same PLB bus, and even PLB Block RAM, is shared with both the instruction memory unit and data memory unit.
- Using instruction cache and data cache allows large improvements over PLB performance. DSOCM and ISOCM are not cacheable but offer performance very similar to that of cache. The use of ISOCM and DSOCM creates a requirement for the use of linker scripts.
- Keep Block RAM quantities as small as possible according to the requirements of the system. Doing so will allow higher clock rates to be used to interface Block RAM to the processor while still satisfying all system time-specifications.
- When using interrupts a large amount of program space will be necessary to satisfy the vector table alignment on a 64KByte boundary and to provide a starting code word at address 0xFFFFFFFC. Creating a split instruction memory space can help to reduce Block RAM quantities at the expense of the extra memory interface complexity needed to handle two Block RAM interfaces.

FFT Performance

This example is based around the use of Xilinx CoreGen FFT component and as such it may be interesting to note how well the system performs with respect to the calculation of a FFT.

The FFT component that has been generated and used in the example is a Xilinx '256-point Radix-2 Minimum Resources' FFT. The same FFT CoreGen component can also be built as a 'Radix-4 Continuous I/O' or 'Radix-4 Burst I/O' version. Each of these three versions will provide a different FFT calculation time and will use a different amount of FPGA resources. The version used in the example is as the name suggests the version that uses the least resources. It is also the slowest version to complete a calculation of one 256-point transform.

How many resources are used will obviously be a significant factor when performing a FFT, as it must fit in the FPGA being used. Also the calculation time is important.

For the FFT CoreGen component used, the time taken to both compute the FFT and to unload the results from the component was 16μs. The component is connected to a 100MHz system clock which is not the 175Mhz maximum that the Xilinx data sheet shows. Keeping everything else constant in the design, operating at 175MHz the total calculation and unload time would be around 9μs.

There is also room for improvement in how the PowerPC and FFT component are used. The repetitive loop that is performed by the example C program is very ordered in that first data is loaded into a Block RAM, the Block RAM is then read and the FFT calculated, the FFT is then unloaded and the CPU waits for indication of completion before restarting.

While the FFT calculation is in progress, the PowerPC is doing nothing other than waiting for that process to complete. The system design could be easily modified such that the PowerPC writes to a second buffer area within the same Block RAM to create the next 256-points of data.

On completion of the first FFT calculation the FPGA design would then be made to read from the second buffer and immediately begin calculation. In this way, the PowerPC core would be performing in parallel with the FPGA and the total loop time would drop for the calculation of a single FFT.

As the time taken to load Block RAM while using cache or OCM is around 5-7μs, the slowest stage would therefore be the FFT engine itself, performing one calculation every 9μs (at 175MHz). In fact the FFT CoreGen component could work faster than this, as it is also capable of overlapping the unloading and loading stages with calculation.

At this point, it should be noted that being able to separate and examine the performance of each individual system element allows us to calculate the performance of the whole. In this way we have a means to quantify the performance of a PowerPC system, regardless of whether the 'Hardware Processing Unit' is performing a FFT, a convolution, or any of a number of other FPGA based functions.

Source Code for the Examples

The FFT Example project is provided in four different versions, each of which interface the FPGA to PowerPC in a different way. By looking at the differences between each approach we can see how certain design decisions can affect the performance of the system.

The FFT Example is organised as four distinct projects for Xilinx Platform Studio and one project for Xilinx Project Navigator.

In the root directory of the PowerPC FFT Example are four groups of Platform Studio project files. The first project is named '`PLB_200`', and includes the files '`PLB_200.xmp`', '`PLB_200.mhs`' and '`PLB_200.mss`'. The second project is named '`PLB_300`', and includes the files '`PLB_300.xmp`', '`PLB_300.mhs`' and '`PLB_300.mss`'.

The third project is named '`OCM_200`', and includes the files '`OCM_200.xmp`', '`OCM_200.mhs`' and '`OCM_200.mss`'. The fourth project is named '`OCM_300`', and includes the files '`OCM_300.xmp`', '`OCM_300.mhs`' and '`OCM_300.mss`'.

The system design and performance of each project is explained in the remainder of this document.

The sub-directory 'ISE' contains the Project Navigator project used to create the final bitstream. This project is used with each of the four Platform Studio projects.

The sub-directory 'Src' contains the VHDL source for the Project Navigator project.

The sub-directory 'pcores' contains the modules 'bram2user' and 'opb2hpu' along with a module for using a DCM to generate PowerPC system clocks. The DCM module is derived from the Xilinx provided DCM 'system_clock' module.

The PLB 200 Example Project

The first example project is the PLB 200 project. In this project Block RAM is provided on PLB for program code and program data. Block RAM is provided on DSOCM for interfacing to the FFT engine.

The project is configured to use a 200MHz clock for the PowerPC 'cpu' clock. A 100MHz clock is used for PLB and DSOCM.

The Block RAM on PLB is configured to be 64KBytes in size and the Block RAM on DSOCM is configured to the smallest possible amount of 8KBytes.

In order to create a bitstream for the PLB-200 project a net-list must first be created in Platform Studio. Please note, when building any of the example projects the current working files are those named '`system.*`'. That is, to build the PLB-200 project the '`PLB_200`' files must be copied over the 'system' files.

To do this, the file '`PLB_200.xmp`' must be copied and re-named '`system.xmp`', overwriting any existing file of that name. Similarly, the file '`PLB_200.mhs`' must be copied and re-named to '`system.mhs`' and the file '`PLB_200.mss`' must be copied and re-named to '`system.mss`'.

To generate the PowerPC system net-list open the Xilinx Platform Studio and then open the project file '`system.xmp`'.

Next the DCM module must be correctly prepared in order to create a cpu clock at 200MHz. In the 'vhdl' sub-directory of the system clock module provided in the 'pcores' directory are three files. The file '`system_clock.vhd`' is the file used by the synthesis tools. For 200MHz operation the file '`200_100_system_clock.vhd`' must be copied and renamed '`system_clock.vhd`'.

With this done select 'Tools→Generate Netlist' in Platform Studio. When the net-list generation has completed successfully the PowerPC design must be exported to Project Navigator. Please note, this MUST NOT be done using the menu item 'Tools→Export to ProjNav'. Instead, you must use the supplied batch file ‘`Export_to_ProjNav.bat`’ supplied in the root example directory.

When the export process has successfully completed open the ISE project in Project Navigator. To build the bitstream select the file ‘top.vhd’ in the Module View and double click on Generate Programming File in the Process View.

When the bitstream has been generated in Project Navigator you will need to return to Platform Studio to build the PowerPC program and combine the resulting executable with the bitstream so that the Block RAMs contain the correct pre-initialised data.

First select the menu item ‘Tools→Import from ProjNav’.

Before the C program can be built the program must be correctly set-up according to whether interrupts are required and whether instruction and data cache should be enabled.

At the top of the example C program ‘`PPC_FFT_Ex.c`’ are two `#defines`. To use polling, the `USE_INTERRUPTS #define` must remain commented. To enable the use of interrupts this line must be uncommented.

Similarly to disable the instruction and data caches the line `USE_CACHE` must be commented. To enable caches, this line must be uncommented.

When the appropriate choices have been made build the C program by selecting the menu item ‘Tools→Build All User Applications’. When this has completed combine the executable with the previously generated bitstream using ‘Tools→Update Bitstream’. The final bitstream ‘download.bit’, can now be downloaded to the target FPGA. This file is written to the implementation directory by Platform Studio.

System Performance

The program executed on the PowerPC runs in a loop repeated the same steps over and over. The first step in this loop is the writing to Block RAM of a sine wave. The sine wave forms the input data to the FFT engine built in FPGA gates. The sine wave is contained in 256 words of data written to Data Side OCM Block RAM.

When the Block RAM Load stage has completed, the PowerPC indicates that FFT processing can begin. This is done by performing an access to the ‘opb2hpu’ module to assert the ‘hpu_start’ signal.

With the ‘hpu_start’ signal asserted the FFT engine begins reading data from Block RAM and starts calculating the transform result. When the FFT engine completes calculation there is then a data unloading stage where the results are output from the FFT core and placed back in data side Block RAM.

If interrupts are not being used, the PowerPC program will be sat in an idle loop waiting for indication that the FFT calculation and unloading has completed. This is done by continuously polling the ACTIVE bit in the ‘opb2hpu’ module. Once this bit becomes de-asserted the loop restarts.

Alternatively, if interrupts are being used the PowerPC program will wait in an idle loop until the interrupt service routine executes as a result of the opb2hpu interrupt. When the interrupt service routine runs it sets a global variable that allows the loop to restart.

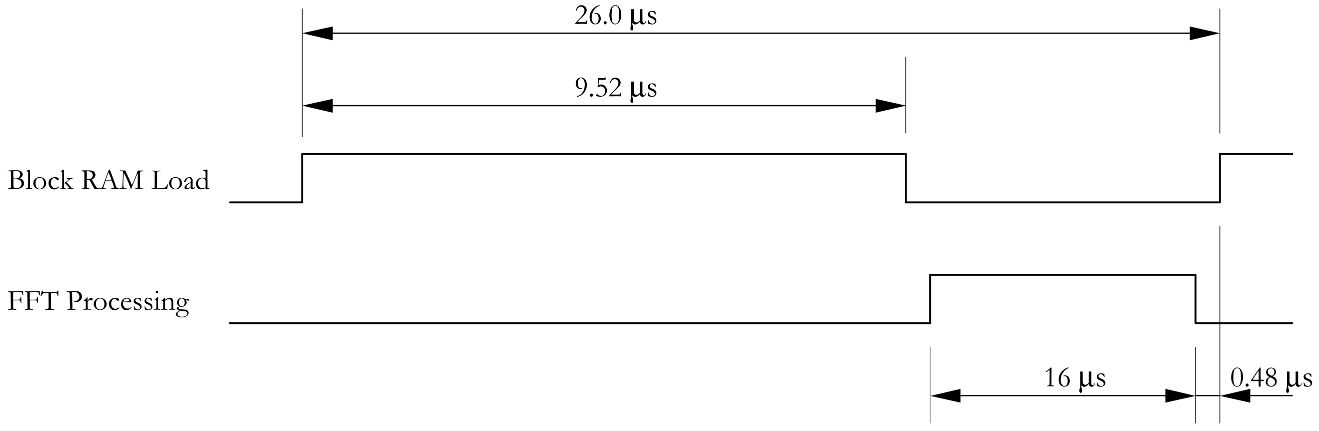
By observing the time taken for each stage of the loop to complete it becomes possible to understand how changes to the system architecture and clock speeds can affect system performance.

For each of the four Platform Studio projects, results are shown in this document when operating with interrupts on and off and with cache enabled and disabled.

PLB 200 Performance, No Interrupts, Cache Disabled

The following diagram shows the performance measurements taken for the PLB 200 project with polling rather than interrupts, and instruction and data cache disabled.

In the diagram the Block RAM Load waveform indicates the time taken for the sine wave to be written to Block RAM. The FFT processing waveform indicates the time taken for the FFT calculation and the unloading stages.

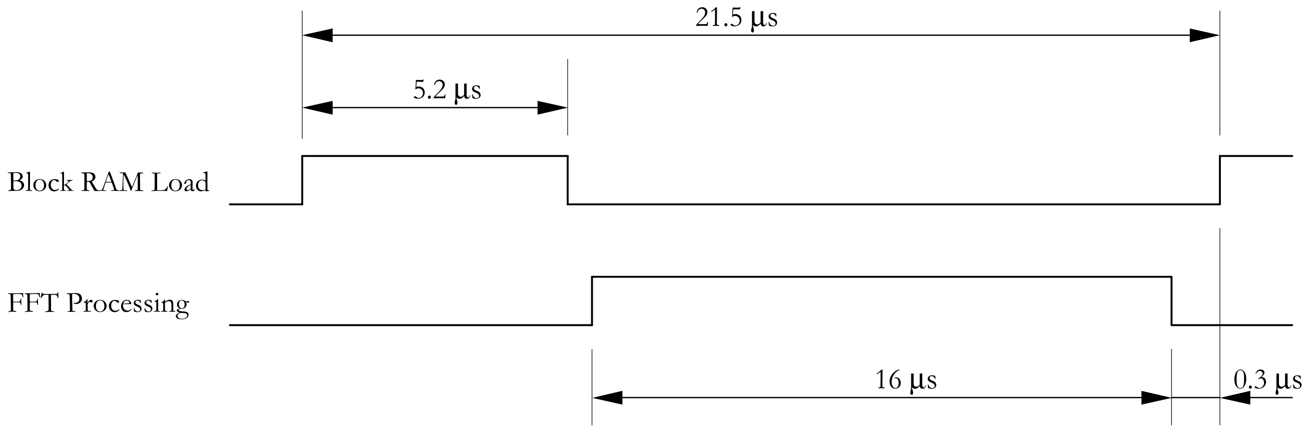


In the above diagram the largest factor in the overall loop time is the time taken for the FFT calculation to take place. The next largest factor is the Block RAM load time.

The time taken for the FFT calculation and unload to complete is dictated by the design of the CoreGen FFT component. Many different FFT designs are possible but for the purposes of the FFT example only one FFT component is needed to in order to study the way an external FPGA component can be connected to the PowerPC.

PLB 200 Performance, No Interrupts, Cache Enabled

If we keep all other design elements constant and enable the instruction cache and data cache we would expect the PowerPC performance to improve. The FFT engine will not be affected by this change and so we should still achieve a 16µs FFT calculation and unload delay.

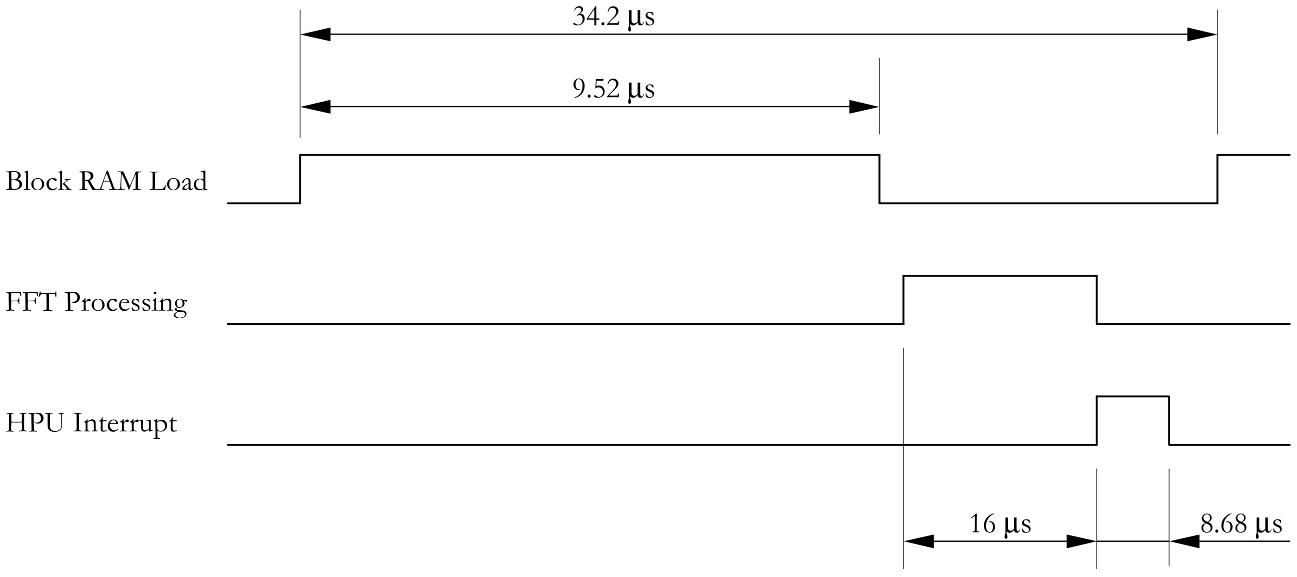


With the caches enabled performance is noticeably better.

PLB 200 Performance, Interrupts, Cache Disabled

The performance should be similar to that with no interrupts and cache disabled. The only difference in the system is that the processor now takes an interrupt after completion of the FFT rather than polling a register.

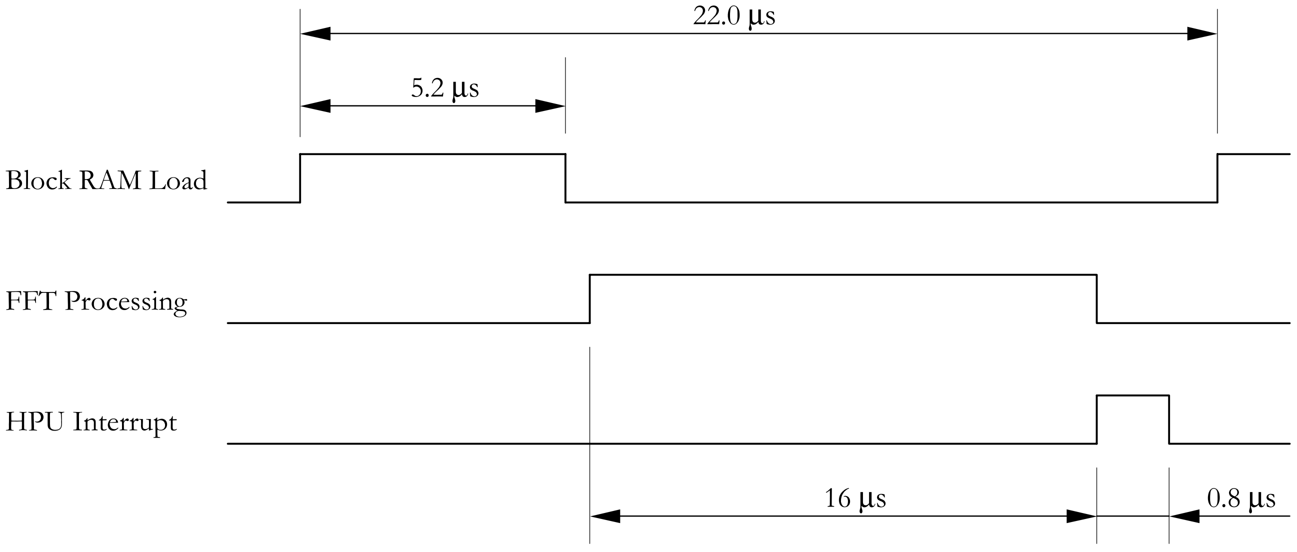
In the example program there is actually more code involved in implementing an interrupt service routine than in polling a register value. This is due to the necessity to save the current context when an interrupt occurs and to restore the context on return from interrupt.



The time taken from the 'hpu_irq' interrupt signal becoming asserted to being cleared by the interrupt service routine is 8.68µs. This interrupt processing time gives a larger overall loop time than polling.

PLB 200 Performance, Interrupts, Cache Enabled

Again, if we keep all other design elements constant and enable the instruction cache and data cache we would expect the PowerPC performance to improve.



The PLB 300 Example Project

The next example project is the PLB 300 project. As with the PLB 200 project there is Block RAM on PLB for program code and program data, and Block RAM on DSOCM for interfacing to the FFT engine.

In order to examine potential performance gains the project remains unchanged from PLB 200 to PLB 300 apart from an increase to the clock rate of the processor core. The project is configured to use a 300MHz clock for the PowerPC 'cpu' clock, with a 100MHz clock for PLB and DSOCM.

The Block RAM on PLB is configured to be 64KBytes in size and the Block RAM on DSOCM is configured to the smallest possible amount of 8KBytes.

In order to create a bitstream for the PLB-300 project a net-list must first be created in Platform Studio. Again you should note that when building any of the example projects the current working files are those named '`system.*`'. That is, to build the PLB-300 project the '`PLB_300`' files must be copied over the 'system' files.

To do this, the file '`PLB_300.xmp`' must be copied and re-named '`system.xmp`', overwriting any existing file of that name. Similarly, the file '`PLB_300.mhs`' must be copied and re-named to '`system.mhs`' and the file '`PLB_300.mss`' must be copied and re-named to '`system.mss`'.

To generate the PowerPC system net-list open the Xilinx Platform Studio and then open the project file '`system.xmp`'.

Next the DCM module must be correctly prepared in order to create a cpu clock at 300MHz. In the 'vhdl' sub-directory of the system clock module provided in the 'pcores' directory are three files. The file '`system_clock.vhd`' is the file used by the synthesis tools. For 300MHz operation the file '`300_100_system_clock.vhd`' must be copied and renamed '`system_clock.vhd`'.

With this done select 'Tools→Generate Netlist' in Platform Studio. When the net-list generation has completed successfully the PowerPC design must be exported to Project Navigator. Please note, this MUST NOT be done using the menu item 'Tools→Export to ProjNav'. Instead, you must use the supplied batch file '`Export_to_ProjNav.bat`' supplied in the root example directory.

When the export process has successfully completed open the ISE project in Project Navigator. To build the bitstream select the file 'top.vhd' in the Module View and double click on Generate Programming File in the Process View.

When the bitstream has been generated in Project Navigator you will need to return to Platform Studio to build the PowerPC program and combine the resulting executable with the bitstream so that the Block RAMs contain the correct pre-initialised data.

First select the menu item 'Tools→Import from ProjNav'.

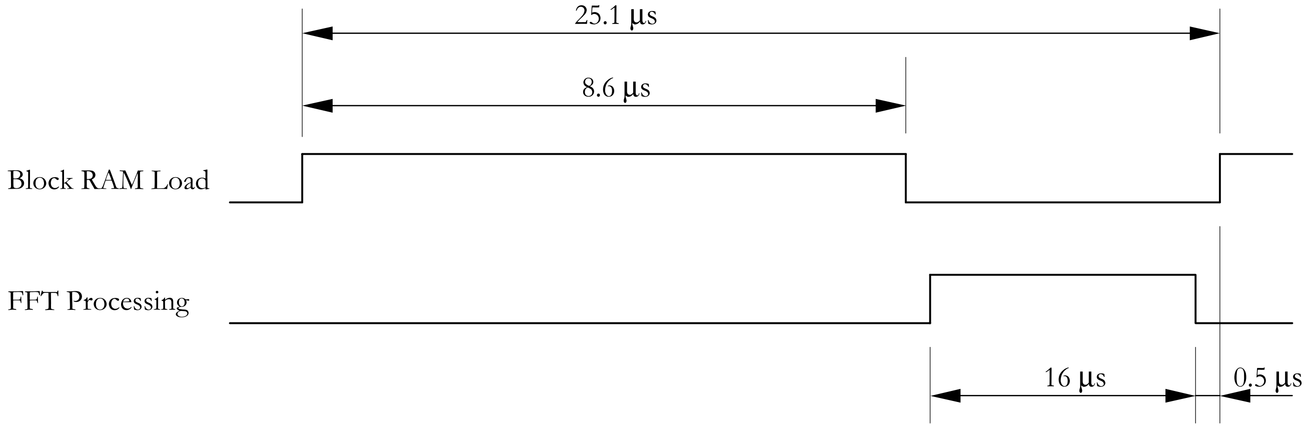
Before the C program can be built the program must again be correctly set-up according to whether interrupts are required and whether instruction and data cache should be enabled. Do this using the `USE_INTERRUPTS` and `USE_CACHE` lines as described for the first project.

When the appropriate choices have been made build the C program by selecting the menu item 'Tools→Build All User Applications'. When this has completed combine the executable with the previously generated bitstream using 'Tools→Update Bitstream'. The final bitstream 'download.bit', can now be downloaded to the target FPGA. This file is written to the implementation directory by Platform Studio.

PLB 300 Performance, No Interrupts, Cache Disabled

The following diagram shows the performance measurements taken for the PLB 300 project with polling rather than interrupts, and instruction and data cache disabled.

The PLB 300 design is very similar to the PLB 200 design, with the only difference being an increase to the CPU clock rate. Accordingly performance improvements can be expected for tasks directly related to the processor, but not for those that are FPGA specific.

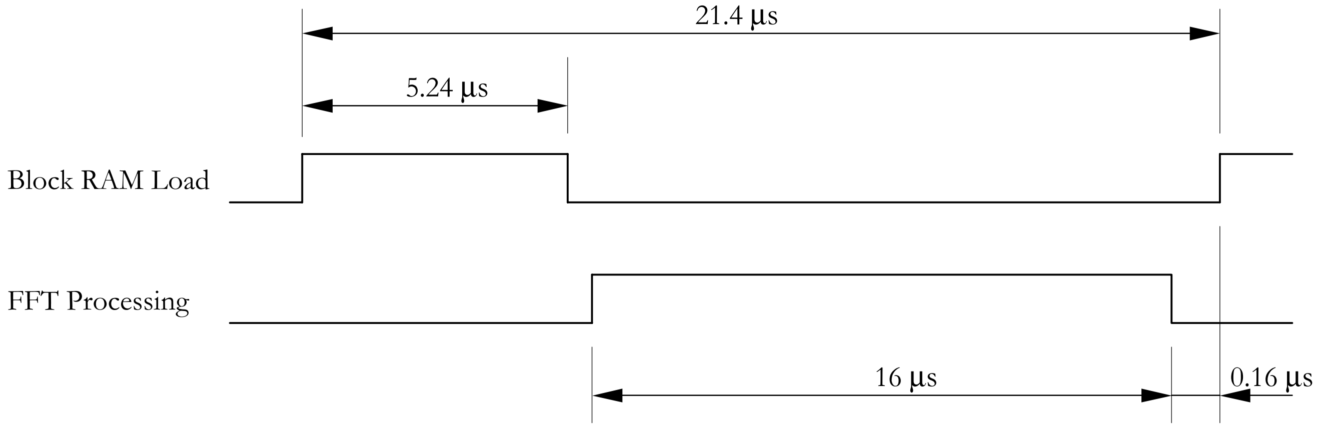


In the above diagram the largest factor in the overall loop time is still the time taken for the FFT calculation.

The time taken for the FFT calculation and unload to complete is dictated by the design of the CoreGen FFT component and remains at 16µs.

PLB 300 Performance, No Interrupts, Cache Enabled

If we keep all other design elements constant and enable the instruction cache and data cache we would expect the PowerPC performance to improve. The FFT engine will not be affected by this change and so we should still achieve a 16µs FFT calculation and unload delay.

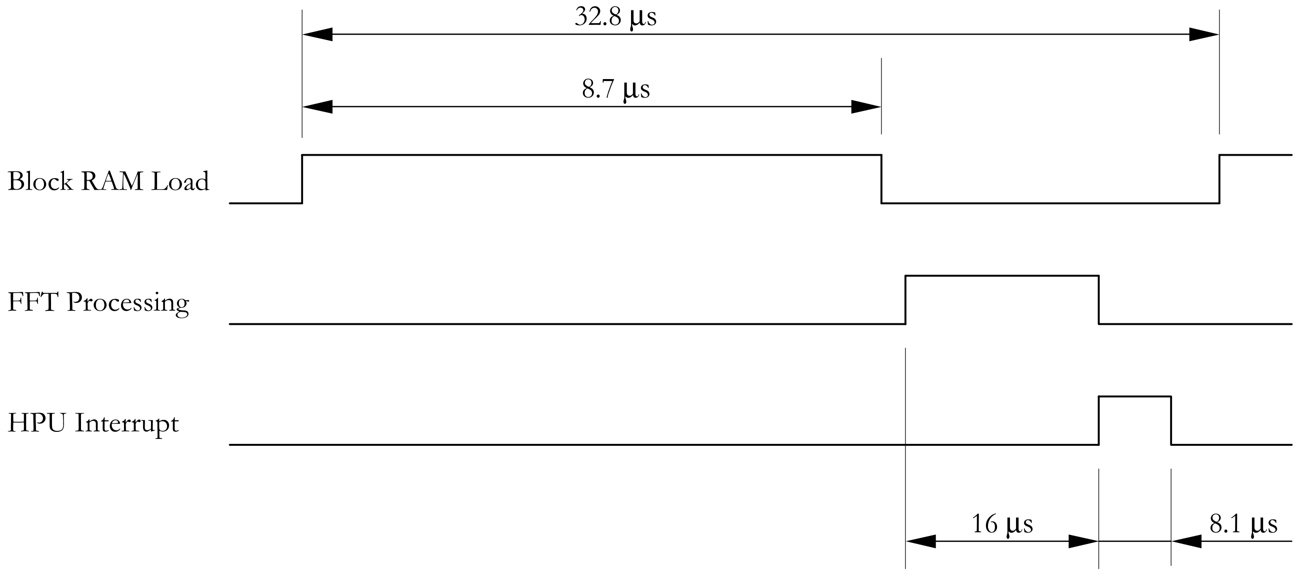


Again, with the caches enabled performance is noticeably better.

PLB 300 Performance, Interrupts, Cache Disabled

The performance should be similar to that with no interrupts and cache disabled. The only difference in the system is that the processor now takes an interrupt after completion of the FFT rather than polling a register.

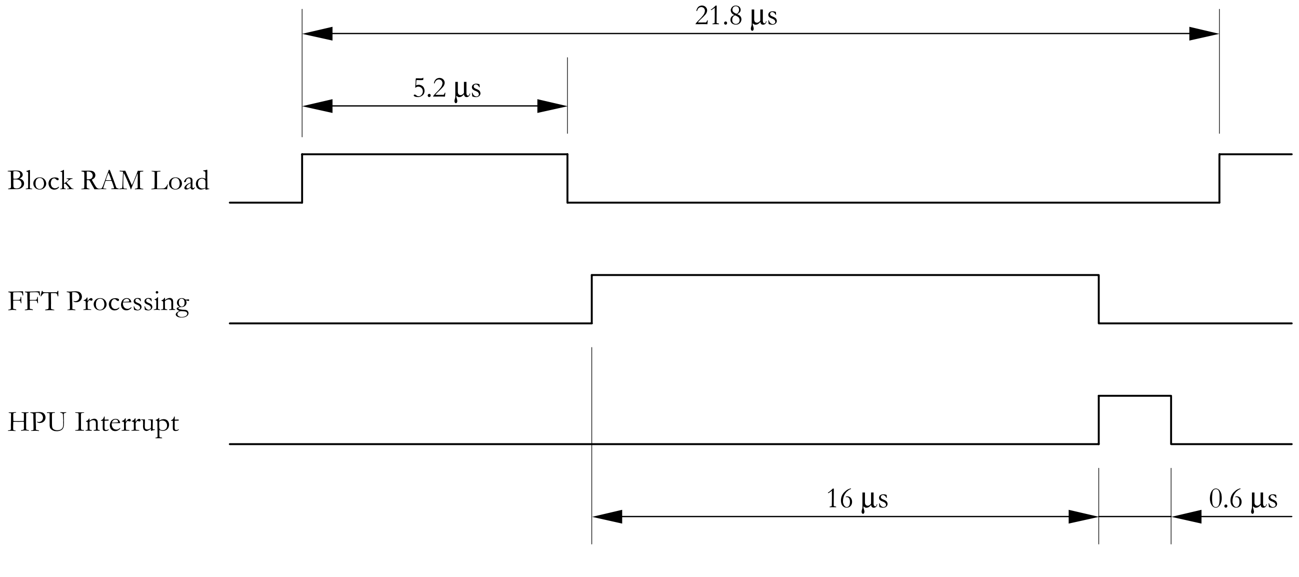
In the example program there is actually more code involved in implementing an interrupt service routine than in polling a register value. This is due to the necessity to save the current context when an interrupt occurs and to restore the context on return from interrupt.



The time taken from the 'hpu_irq' interrupt signal becoming asserted to being cleared by the interrupt service routine is 8.1µs. This interrupt processing time gives a larger overall loop time than polling.

PLB 300 Performance, Interrupts, Cache Enabled

Again, if we keep all other design elements constant and enable the instruction cache and data cache we would expect the PowerPC performance to improve.



The OCM 200 Example Project

The third example project is the OCM 200 project. This project is created by taking the first project and using On-Chip-Memory interfaces rather the PLB for both program instruction and data accesses.

By doing this we would expect to achieve performance gains from the PLB 200 project for several reasons. With both Instruction Side On-Chip-Memory (ISOCM) and Data Side On-Chip-Memory (DSOCM) we now have two completely independent connections to code space and data space. With the PLB based projects, a single PLB bus was shared along with one large area of Block RAM for all code and data accesses.

The On-Chip-Memory interfaces inherently offer a higher performance connection to memory, as they are not arbitrated buses like PLB and OPB. In the OCM projects there is a straight forward connection between the instruction memory controller and Block RAM and there is also a straight forward connection between the data memory controller and Block RAM.

For the OCM 200 example project, the Block RAM on ISOCM is configured to be 4KBytes in size and the Block RAM on DSOCM is configured to be 8KBytes. The Block RAM on DSOCM is used both to interface to the FPGA FFT engine and to act as data space for the C program, and has therefore been increased in size from the PLB 200 and PLB 300 projects.

In order to create a bitstream for the OCM-200 project a net-list must first be created in Platform Studio. Again you should note that when building any of the example projects the current working files are those named '`system.*`'. That is, to build the OCM-200 project the '`OCM_200`' files must be copied over the 'system' files.

To do this, the file '`OCM_200.xmp`' must be copied and re-named '`system.xmp`', overwriting any existing file of that name. Similarly, the file '`OCM_200.mhs`' must be copied and re-named to '`system.mhs`' and the file '`OCM_200.mss`' must be copied and re-named to '`system.mss`'.

To generate the PowerPC system net-list open the Xilinx Platform Studio and then open the project file '`system.xmp`'.

Next the DCM module must be correctly prepared in order to create a cpu clock at 200MHz. In the 'vhdl' sub-directory of the system clock module provided in the 'pcores' directory are three files. The file '`system_clock.vhd`' is the file used by the synthesis tools. For 200MHz operation the file '`200_100_system_clock.vhd`' must be copied and renamed '`system_clock.vhd`'.

With this done select 'Tools→Generate Netlist' in Platform Studio. When the net-list generation has completed successfully the PowerPC design must be exported to Project Navigator. Please note, this MUST NOT be done using the menu item 'Tools→Export to ProjNav'. Instead, you must use the supplied batch file '`Export_to_ProjNav.bat`' supplied in the root example directory.

When the export process has successfully completed open the ISE project in Project Navigator. To build the bitstream select the file 'top.vhd' in the Module View and double click on Generate Programming File in the Process View.

When the bitstream has been generated in Project Navigator you will need to return to Platform Studio to build the PowerPC program and combine the resulting executable with the bitstream so that the Block RAMs contain the correct pre-initialised data. Select the menu item 'Tools→Import from ProjNav'.

Before the C program can be built for the OCM projects the program must be set-up to not use interrupts and to not use instruction and data cache. Do this by commenting the `USE_INTERRUPTS` and `USE_CACHE` lines as described for the first project.

When the appropriate choices have been made build the C program by selecting the menu item 'Tools→Build All User Applications'. When this has completed combine the executable with the previously generated bitstream using 'Tools→Update Bitstream'. The final bitstream 'download.bit', can now be downloaded to the target FPGA.

Using Interrupts

The OCM 200 project uses a different amount of Block RAM for instruction and data than is used by the PLB based projects. It is as a result of this that the OCM project cannot be used with interrupts unless the amount of instruction space were increased.

When building an application for the PowerPC in the Virtex-II Pro, the program start address is always 0xFFFFFFFC. Therefore, it is always necessary to create an area of Block RAM that is addressable by the instruction unit which extends to the address 0xFFFFFFFC.

When building an application that uses interrupts, an interrupt vector table will be created. This table provides hooks for interrupt service routines and is the first area of code accessed when an interrupt occurs. There is a requirement when placing the vector table that it is aligned on a 64-KByte address offset.

In order to satisfy both the start address requirements and vector table requirements it is therefore necessary to have program Block RAM from 0xFFFF0000 to 0xFFFFFFFF. This equates to a 64-KByte Block RAM.

For the OCM 200 and OCM 300 projects, building the design with this amount of ISOCM would result in a failure to meet the timing specifications when operating at a 100MHz OCM clock. The alternatives are to reduce the amount of Block RAM in order to meet time-specs or to lower the clock rate of instruction side OCM.

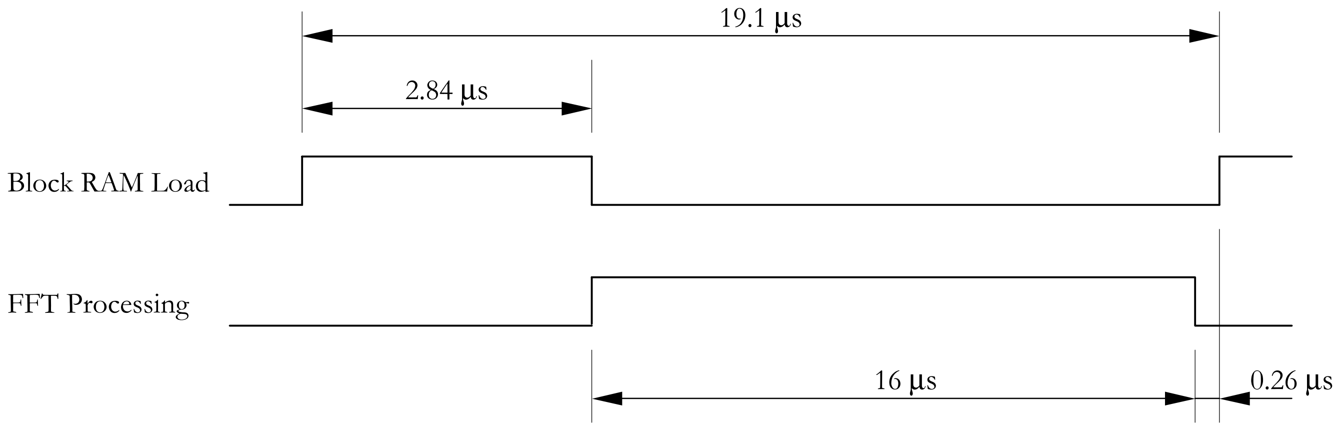
Using Cache

The Cache units for the PowerPC 405 work by caching instruction and data fetches over PLB. The On-Chip-Memory interfaces of the processor core are provided as a means to interface high-speed memory to the processor. The OCM interfaces are designed such that memory can be accessed at a rate close to the performance level of accessing Cache.

Data stored in memory that is accessed using the On-Chip-Memory interfaces is not cacheable. Accordingly, the OCM 200 and OCM 300 projects must only be used with the caching feature disabled in the example C program.

OCM 200 Performance

The following diagram shows the performance measurements taken for the OCM 200 project. The OCM 200 design uses on-chip-memory interfaces for both program and data and should therefore provide a performance increase over the PLB based approach, with cache disabled.



The performance with OCM should be close to that seen when using instruction and data caches.

The OCM 300 Example Project

The final example project is the OCM 300 project. This project is created by taking the OCM-200 project and increasing the processor clock to 300MHz.

In order to create a bitstream for the OCM-300 project a net-list must first be created in Platform Studio. Again you should note that when building any of the example projects the current working files are those named '`system.*`'. That is, to build the OCM-300 project the '`OCM_300`' files must be copied over the 'system' files.

To do this, the file '`OCM_300.xmp`' must be copied and re-named '`system.xmp`', overwriting any existing file of that name. Similarly, the file '`OCM_300.mhs`' must be copied and re-named to '`system.mhs`' and the file '`OCM_300.mss`' must be copied and re-named to '`system.mss`'.

To generate the PowerPC system net-list open the Xilinx Platform Studio and then open the project file '`system.xmp`'.

Next the DCM module must be correctly prepared in order to create a cpu clock at 300MHz. In the 'vhdl' sub-directory of the system clock module provided in the 'pcores' directory are three files. The file '`system_clock.vhd`' is the file used by the synthesis tools. For 300MHz operation the file '`300_100_system_clock.vhd`' must be copied and renamed '`system_clock.vhd`'.

With this done select 'Tools→Generate Netlist' in Platform Studio. When the net-list generation has completed successfully the PowerPC design must be exported to Project Navigator. Please note, this MUST NOT be done using the menu item 'Tools→Export to ProjNav'. Instead, you must use the supplied batch file '`Export_to_ProjNav.bat`' supplied in the root example directory.

When the export process has successfully completed open the ISE project in Project Navigator. To build the bitstream select the file 'top.vhd' in the Module View and double click on Generate Programming File in the Process View.

When the bitstream has been generated in Project Navigator you will need to return to Platform Studio to build the PowerPC program and combine the resulting executable with the bitstream so that the Block RAMs contain the correct pre-initialised data. Select the menu item 'Tools→Import from ProjNav'.

Before the C program can be built for the OCM projects the program must be set-up to not use interrupts and to not use instruction and data cache. Do this by commenting the `USE_INTERRUPTS` and `USE_CACHE` lines as described for the first project.

When the appropriate choices have been made build the C program by selecting the menu item 'Tools→Build All User Applications'. When this has completed combine the executable with the previously generated bitstream using 'Tools→Update Bitstream'. The final bitstream 'download.bit', can now be downloaded to the target FPGA.

Increasing Clock Speed - Decreasing Bus Speed

When using each of the main interfaces of the PowerPC there is a trade-off between the maximum clock speed of the processor core and the speed used to clock the individual memory interfaces.

The ideal situation would be to increase all clock speeds within the design such that the entire system runs at the maximum clock rate of the processor. However, this is impossible in practice as there is a significant difference in the performance potential of the processor core and the external memory interfaces that it connects to.

When connecting to either PLB, ISOCM or DSOCM a large amount of logic exists to handle each type of access. As has been found with each of the example projects, a realistic interface speed is around 100 to 200MHz, whereas the processor core is certainly capable of running at speeds in excess of 300MHz.

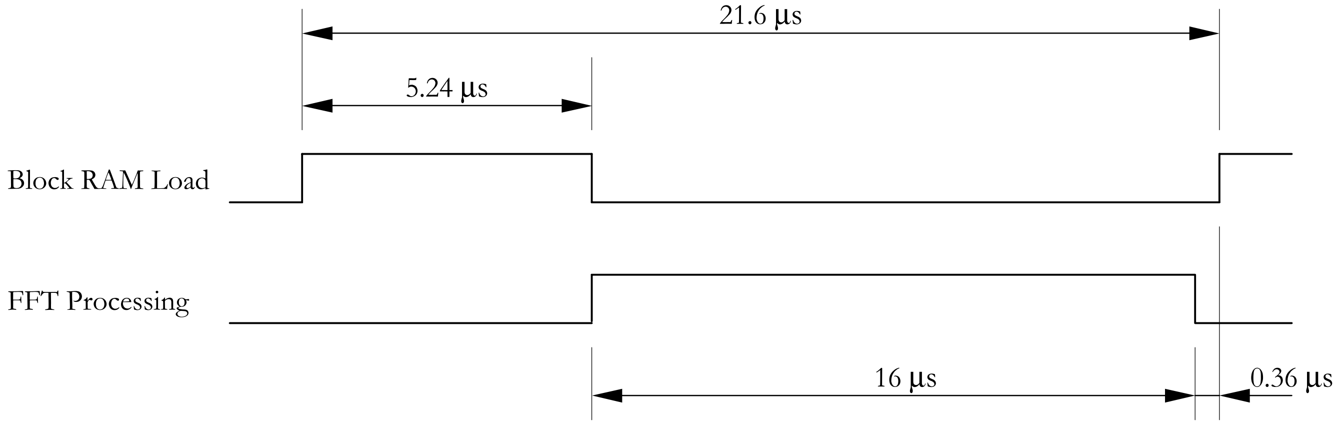
The OCM-200 project is in fact able to operate with Instruction Side and Data Side OCM interfaces running at 200MHz. This is possible as the interface to OCM is significantly simpler in terms of the amount of logic than the interface to PLB. Also, the amount of Block RAM used for ISOCM and DSOCM has been kept deliberately low in order to enable 200MHz operation. The result is a system where both the processor core and instruction and data side accesses are all running at 200MHz.

An alternative to this situation is to create a design where more processor-specific performance is obtained. This is done in the OCM 300 project by increasing the processor clock rate to 300MHz. However, the OCM interfaces will not now route with a clock speed of 200Mhz. This is stated by the tools, so as a compromise the OCM interfaces have been dropped to 100MHz in order to route while meeting system time-specs.

The results of this different approach to clock rates can be seen when we compare the OCM 200 performance results with the OCM 300 performance results.

OCM 300 Performance

The following diagram shows the performance measurements taken for the OCM 300 project.


21.6 μs
5.24 μs
Block RAM Load
FFT Processing
16 μs
0.36 μs