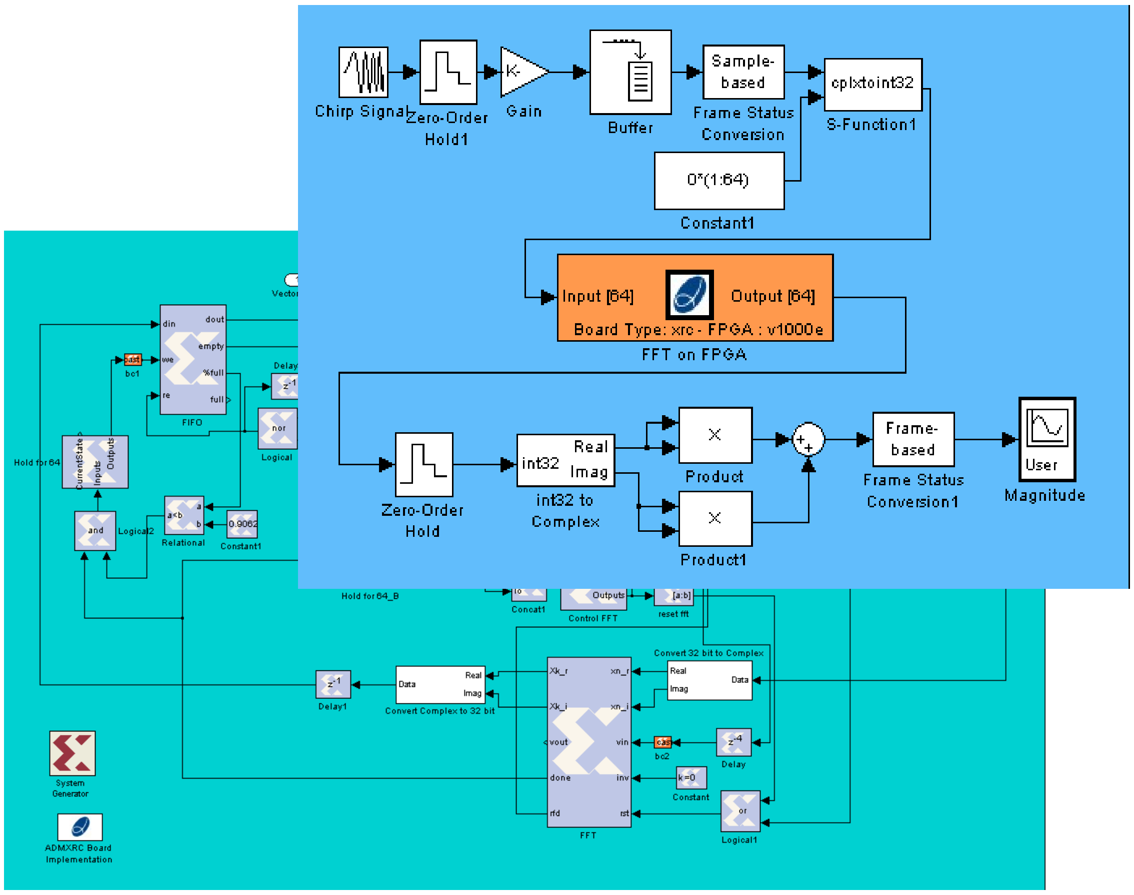
XRC Board Level Application Library v1.7 (For System Generator v6.2)

ALPHA DATA *parallel systems ltd*



Alpha Data Parallel Systems Ltd.
58 Timber Bush
Edinburgh EH6 6QH
Scotland
UK

Phone: +44 (0) 131 555 0303
Fax: +44 (0) 131 555 0728
Email: support@alphadata.co.uk

1 Introduction

The XRC Board Level Application Library used in conjunction with Xilinx's System Generator v6.1 and The MathWorks Simulink allows a simple method of building FPGA applications which run on the Alpha Data Hardware platforms, and fully exploit the powerful I/O functionality. The Alpha Data XRC BLA Library provides a number of different blocks which can be used to construct a System Generator design targeted for an FPGA on an ADM-XRC, ADM-XRC-II, ADP-XRC-II, ADM-XRC-II-Pro-Lite (ADM-XPL) or ADM-XRC-II-Pro (ADM-XP) board. Template blocks are provided, modelling the PCI I/O (slave mode and demand mode DMA) of the ADM-XRC board. Additional blocks allow the SRAM and I/O ports to be accessed in the design.

The template blocks include an icon which automatically steps through the synthesis and implementation steps to produce a bitstream which can be loaded onto an FPGA (This requires Synplify Synplicity Pro, Synplicity or Xilinx XST and the Xilinx ISE tools)

This FPGA can be tested as hardware in the loop using the FPGA configuration Simulink block. This hardware in the loop approach is higher speed, but less flexible than the cosimulation blockset method. This blockset is primarily aimed at creating board level applications rather than for hardware in the loop testing of individual modules (as in the case of the cosim blockset). The FPGA can also be controlled using the Matlab toolbox or C code using the ADM-XRC SDK API functions. The blockset also includes a scope component which records signals in on board SRAM. After the test run has completed, these can be downloaded into the Matlab variable space for analysis.

2 Installation Instructions

The XRC Board Level Application Library is provided as a zip file and a Matlab installation script:

xrc_application_blockset.zip
setup_xrc_application.m

1) Open Matlab

2) Change Directory to where xrc_application_blockset.zip and setup_xrc_application.m have been downloaded.

3) Type setup_xrc_application

4) Quit Matlab

3 ADMXRC Simulink Blockset Tutorial

3.1 Opening the Simulink Blockset

The ADMXRC Simulink blockset consists of a number of Simulink blocks which can be inserted into Simulink models. There are two methods of accessing these blocks: they can be copied and pasted from the Alpha Data Library, or they may be dragged from the Alpha Data folder in the Simulink Library Browser.

To view the Alpha Data Library, type:

```
>> open adlibrary
```

This will display the library as shown below in figure 1.

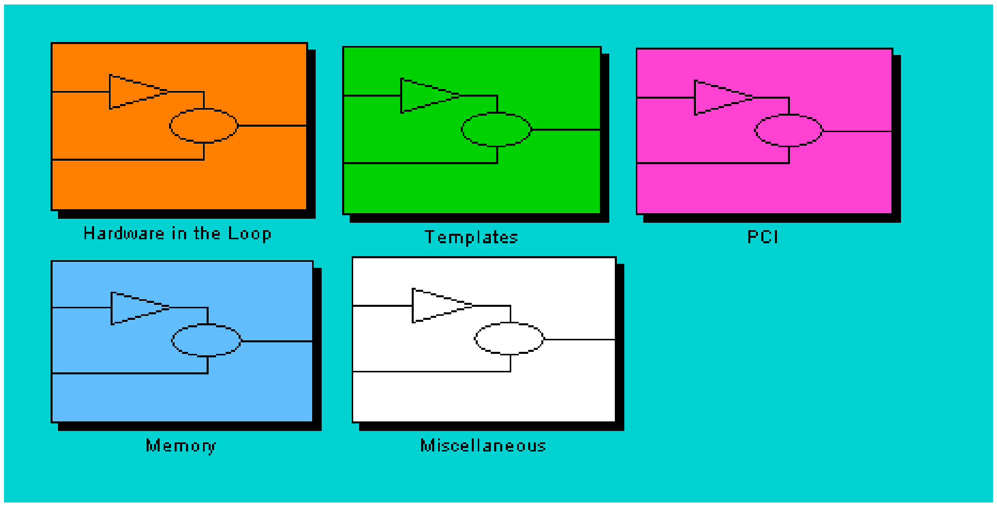
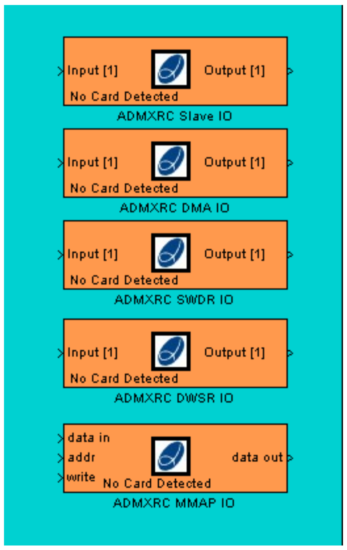
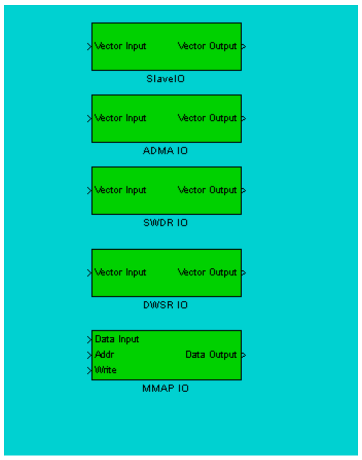
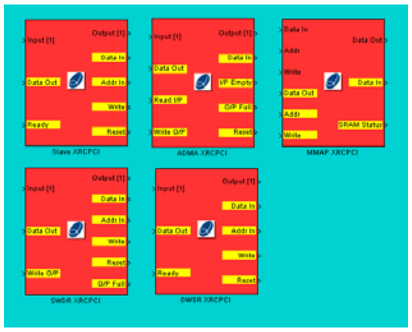
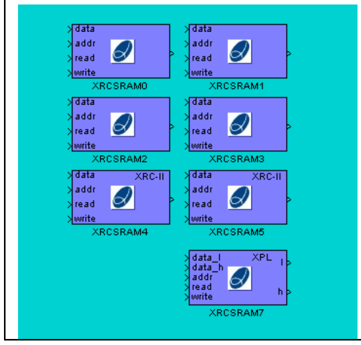
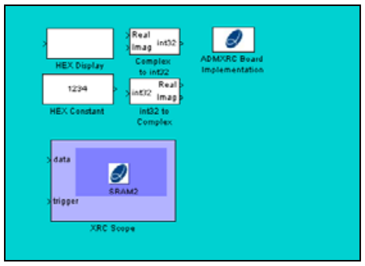


Figure 1: Alpha Data ADMXRC Simulink Library

The library contains 6 groups of blocks described overleaf.

Hardware in the Loop blocks for communicating with the FPGA  ADMXRC Slave IO ADMXRC DMA IO ADMXRC SWDR IO ADMXRC DWSR IO ADMXRC MMAP IO No Card Detected	Matching Templates for building System Generator Designs. 	PCI Interface Models modelling communication between the FPGA and Simulink 
Memory Models allowing access to XRC ZBT SRAMS  XRCSRAM0 XRCSRAM1 XRCSRAM2 XRCSRAM3 XRCSRAM4 XRCSRAM5 XRCSRAM7	Other Miscellaneous Functions  HEX Display HEX Constant ADMXRC Board Implementation XRC Scope	

The second method of accessing the blocks is to open the Simulink browser by typing:

```
>> simulink
```

If the toolbox has been installed properly then the Alpha Data Library will appear in the browsers list of libraries.

3.2 Simple Hardware in the Loop

In this example, the running of a bitstream on an ADM-XRC, ADM-XRC-II or ADM-XPL card is demonstrated.

First change directory to the Blockset Examples directory:

```
>> cd c:\matlab\toolbox\admxrc_sb\examples
```

Open the example

```
>> open simple_on_card
```

This will bring up the model shown in figure 2:

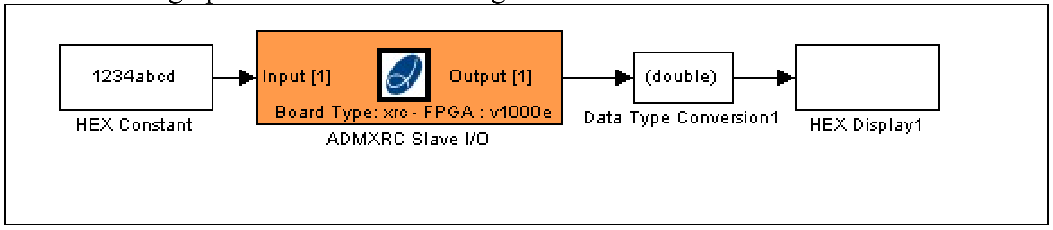


Figure 2: Simple On Card Model

In this example a hexadecimal constant (8-digit) is fed into the FPGA application using Slave mode programmed I/O. The FPGA nibble reverses the data and it is read out using Slave mode programmed I/O, and displayed in the Hex Display box.

To set the parameters for the FPGA, double click on the ADMXRC Slave I/O block (orange). This will bring up a window into which parameters can be entered.
In the field: **Filename>** you need to enter the filename for the FPGA bitstream.

A number of bitstreams are included in the examples directory. In the Matlab window type the following to display all the available bitstream files:

```
>> dir simple-sb*.bit
```

to display all the available bitstreams.
These filenames are in the format simple-sb-<card>-<fpga>.bit

Type the bitstream filename for your card into the filename field of the parameters pop-up. e.g. if you are using an XRC-II with an Virtex-II 2v6000, you would enter `simple-sb-xrc2-2v6000.bit`

The rest of the fields determine the local bus clock speed (this should be limited to 33MHz on XRC cards and 66MHz on XRC-II, and the number of inputs and outputs. The operation of the simulink block is to transmit a vector of data across the local bus, poll an address on the FPGA, which is used to indicate completion of the algorithm, and then read data back to FPGA. The input and output fields determine the size of these data transfers. The terminate loop limit stops Matlab hanging if the FPGA never indicates that it is finished, and the card index can be used if multiple cards are installed in your PC. Click on OK to close the pop-up window.

You can now run the Simulink Model, by selecting Start in the Simulation Menu. The nibble revered output of the input HEX constant will appear in the HEX Display window. The nibble reversal is performed using Hardware in the Loop. You can edit the HEX Constant value and run the Simulation again to see the effects of your changes.

3.3 Creating a Simple Bit Stream

The above example is a trivial example of running hardware in the loop, using a pre-compiled bitstream. In this tutorial example, the steps required to create a bitstream will be performed.

To open the example type:
```
>> open simple
```

This will bring up the model as shown in figure 3. This is very similar to the hardware in the loop example, but the orange block has been replaced with a green template block. You can select Start in the Simulation menu to run this model. Double clicking on the template block brings up a window allowing parameters such as the number of inputs to be modified.

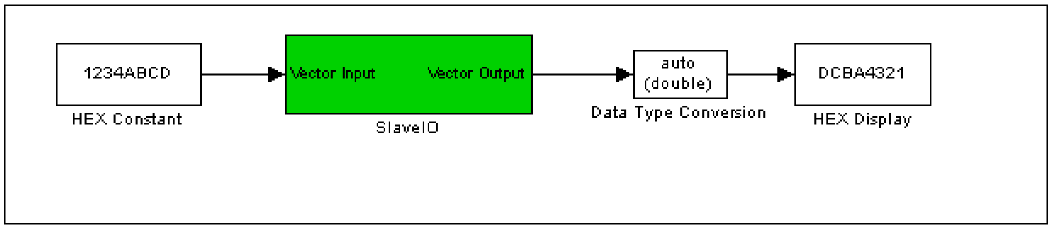


Figure 3: Simple Model

To access the template design, double click on the SlaveIO block This will open the subsystem in figure 4.

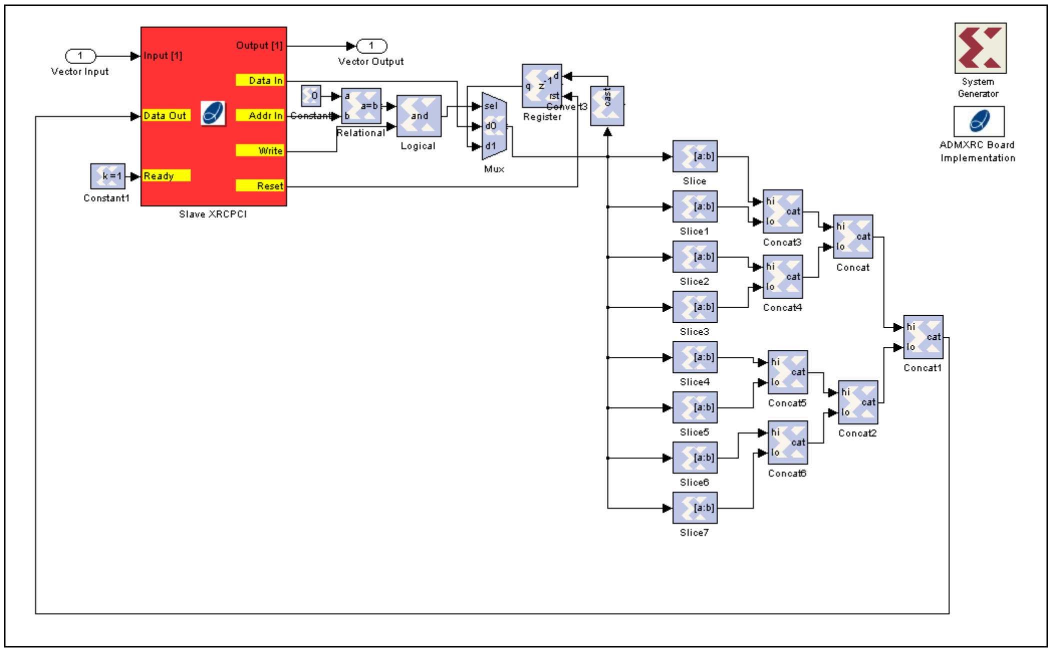


Figure 4: Simple Model using System Generator

The model contains a number of different blocks. Firstly there is the XRC PCI I/O block. This models the interaction between the Hardware in the Loop Simulink S-Function Inputs and Ouputs and the FPGA hardware subsystem. The PCI blocks in the library provide ports to which System Generator blocks can be attached.

Between these ports sits the System Generator Design, which can be any circuit implementible using the System Generator blockset. In this case, the simple design consists of a register which latches the input *data_in* if *write* is asserted and *addr_in* is zero. This value is then Sliced into 8 four-bit fields and concatenated to produce a 32 bit nibble reversed word at *data_out*. The *ready* signal is always asserted.

There are two other special blocks in the design. Firstly there is the Xilinx System Generator block which is used to turn the design into VHDL. Secondly there is a special Alpha Data block "ADMXRC Board Implementation" which takes the VHDL produced by System Generator, combines this with Alpha Data VHDL code which interacts with the PCI interface (the code interacts with the on board PLX PCI chip, the PCI interface is not on the FPGA), and also automatically generates a constraint file. If chosen, XST, Synplify or Synplify Pro, can be automatically run, followed by the Xilinx ISE tools to produce a bitstream file.

The first part of the process is to generate the VHDL using Xilinx System Generator. Double click on the System Generator Icon to open the pop-up window. Select the product family, device, package and speed grade for your system. Select your synthesis tool. Choose a target directory. The "create testbench" box need not be ticked. The simulink clock period should be 1s (for proper interaction with the PCI models). Specify the system clock period required for your design. The Compilation

Box should be left as 'HDL Netlist', and the Clock Pin box can be left blank. Click on Generate to create the VHDL, and click OK to close the Windows.

Now right click on the "ADMXRC Board Implementation" and select "Mask Parameters …" from the menu. This allows you to select your board (XRC, XRC-P, XRCII or XRCII-L). You can also tick the Run Synthesis box which will automatically run XST or Synplify and the ISE tools. If you do not tick this only the VHDL and UCF files will be created. The Make Clean option removes some large intermediate files when the bitstream generation finishes. Do not put anything into the Custom Template File box. Ignore the LVDS I/O Standard and Clock Source fields.

Click on OK to close the window.
Double click on the "ADMXRC Board Implementation" to run the VHDL conversion, UCF file generation and bitstream synthesis if you've selected this.

N.B. If you are using Leonardo Spectrum, you will only be able to generate the vhdl and ucf. The required files for the project can be found in the text file vhdlFiles. From these you can generate the bitstream, although the synthesis may remove some pins from the design requiring these to be removed from the UCF file before NGDBUILD will run correctly.

Close the simple model, and re-open the simple_on_card model.
You can replace the simple-sb-xx-xx.bit filename with your newly created file (x:/xx/xx/simple.bit) and test it on the hardware.

Finally, you can try rearranging the connections between the slices and the concat blocks in the simple example to create a different nibble rearrangement function. This can be tested as a model. Then rebuilt as a new bitstream and loaded into simple_on_card to be tested in hardware.

3.4 Memtest Example

The memtest example demonstrates the use of the SRAM blocks in the ADM-XRC simulink blockset.

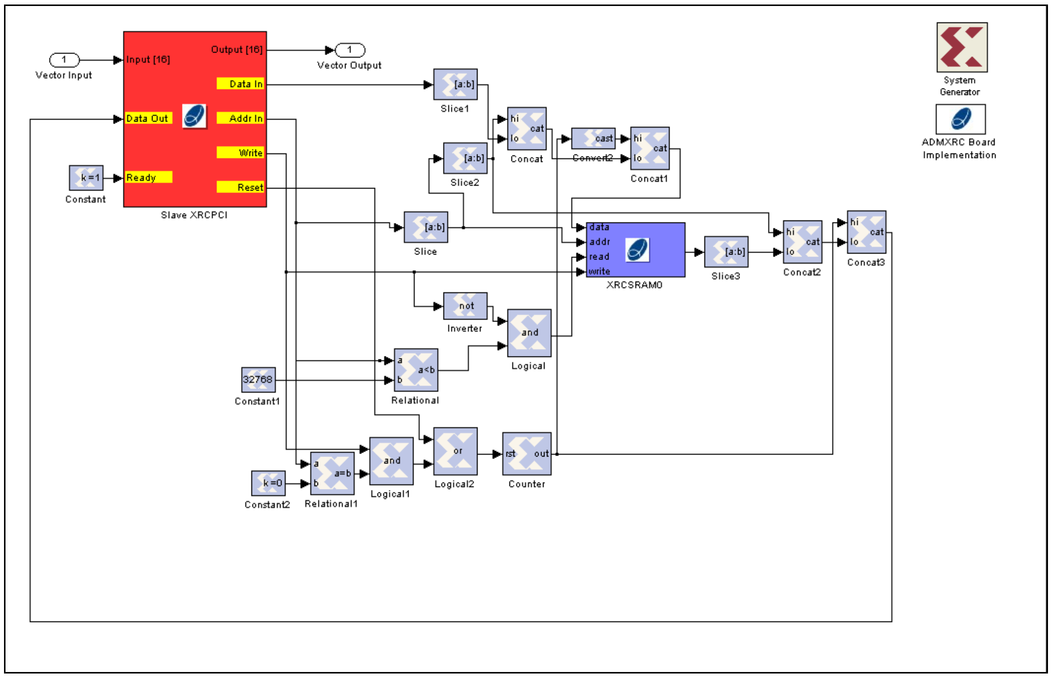


Figure 5: Memtest Example

In this case a counter is reset at the first write, the address data and the counter value are all written into memory. When the data is read out, the top 12 bits of the output are set to the counter value and the address value. This shows the relationship between the read address and the data coming out of the SRAM. There is a 4 clock cycle delay between *read* being asserted and the data from that address appearing at the output.

After running the simulation, the 32 bit hex output can be displayed by typing

```
>> s=sdec2hex(double(simout.signals.values(4,:)))
```

Bits 7:0 (the data read out of memory) can be displayed by

```
>> s(:,7:8)
```

Bits 11:8 (the address when the data were written) can be displayed by

```
>> s(:,6)
```

Bits 19:12 (the counter value when the data were written) can be displayed by

```
>> s(:,4:5)
```

Bits 23:20 (the address when the data is read) can be displayed by

```
>> s(:,3)
```

And bits 31:24 (the counter when the data is read) is displayed using

```
>> s(:,1:2)
```

If you build the bit file you can compare the results with simulation. In this case some discrepancy will be obvious. This is because the reads and writes across the PCI bus are not predictable. There may be several clock cycles between writes to successive addresses, although on some architectures, the write may be able to burst. The traffic generated by other hardware on the PCI bus may also interfere. Also the delay between writing and reading will be far longer than the single clock cycle, used in the simulation model.

These are some of the limitations of using slave mode programmed I/O. Another limitation is that the circuit is synchronous to the local bus clock which is limited to 33 or 66MHz. For some applications the simplicity of the programmed I/O interface may be sufficient, if only a few control signals are required. If large amounts of data are required to be transferred between the FPGA and the host then either the MMAP template or one of the Demand Mode DMA templates is more appropriate.

3.5 MMAP Example

The mmap example, demonstrates the use of the power memory mapped template. With this template, data is transferred between the host and FPGA application using buffers of RAM which can be accesed asynchronously from both sides.

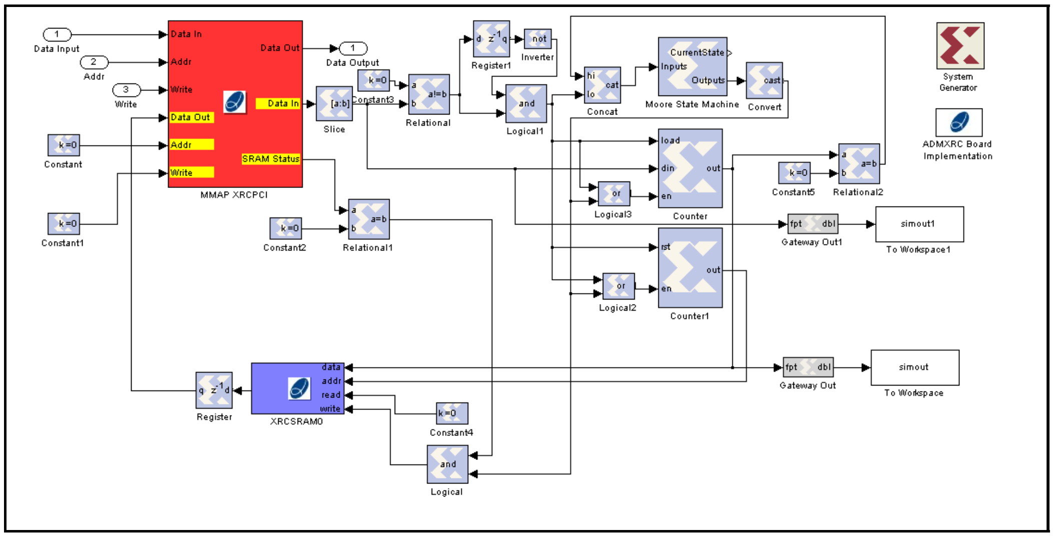


Figure 6: MMAP Example

From the host side, a control page register is used to select whether the host is reading or writing from either and asynchronous 512x32bit Dual Port Control RAM or from one of the banks of ZBT SRAM . From the FPGA application, the Control RAM and the 6 SRAM banks can be accessed simultaneously. However when the host is accesssing an SRAM bank, FPGA access to this bank is not permitted. This restriction does not apply to the asynchronous control RAM.

In the example, the MMAP XRCPCI block contains the model for the Dual Port Control RAM block, and access to this RAM from the FPGA is handled through ports on this block. This block also models the background transfer of data between the host and the SRAM. FPGA access to SRAM banks is provided using the standard SRAM blocks included in the library.

In the example, when a non-zero value is written into Control RAM address 0, this is latched into a negative counter which counts from this value down to 0. The output of this counter is written into SRAM bank 0 starting at address 0.

The control program (created using init_mmap_program.m) writes '16' into address 0 and then clears it, a few clock cycles later. After leaving time for the SRAM bank to be written, the program writes 8 to the status register (address '0x1FFFFC') to allow access to this bank from the host, and then reads out the 1st 20 values stored in the SRAM.

In the example the output from the SRAM is fed into input of the Control RAM through a register, however this is just to ensure the the ports are not optimised away by System Generator, and does not actually provide any functionality.

3.6 I/O VHDL Modules

Specification of I/O is highly application dependent. Simply connecting signals to I/O pins will be insufficient for most applications. Therefore I/O port specification for the XRC is handled by insertion of VHDL modules between the System Generator FPGA application and the I/O ports at the top level. The most compelling breason for using this approach is that it enables interfacing with external hardware running at a different clock domain, or the use of some of the flexible I/O pin options such as DDR, LVDS or Tri-State signalling.

3.6.1 Specifying the ports in System Generator

In the system generator model, extra ports can be added by including extra Gateway (yellow) blocks. In the top level System Generator generated VHDL entity these ports will appear as inputs or outputs.

3.6.2 Specifying the ports in the top level

Once extra ports have been specified in the System Generator model, the top level module must also be modified to route these to external pins. This must be specified in a script file with the following format:

This script file is a text file.

It can contain the line DIR=c:\mydirectory\
To specify the directory where the VHDL source for the I/O interface module is stored.
It must contain the line FILE=xxx.vhd
Where xxx.vhd is the required VHDL source, either in the currect directory or in the directory specified by DIR=
If the VHDL module requires the clock signal from the System Generator module, then the line
CLK=clk
Should be included (assuming that the clock input for the module has port name clk)
A reset input to the module can also be specified using
RST=rst

The system generator port names should then be specified:
The line SYSGEN PORT should be included, followed by lines of the following format:
port_name : direction : size
where
'port_name' should match the name specified in System Generator,
'direction' should be in or out, but this will be the opposite to System Generator specification and refers to the port direction on the VHDL I/O module.
'size' indicates the bit width of the port

After declaring the System Generator Port Names, the line
TOP_LEVEL_PORT should be inserted, followed by lines indicating the interface to the top level design. These ports will become output ports of the top level.
The same format is used, with the ports matching the direction of the VHDL I/O module. In this case the direction can also be specified as inout, to create tri-state ports.

After declaring all the top level port names, the line CONSTRAINTS should be added. This should be followed by the contraints defining the pins to be associated with the top level ports. All lines in the file after the line CONSTRAINTS will be added to the UCF file.

The CLK,RST lines, and all the SYSGEN PORT and TOP LEVEL PORT definitions are used to modify the top level VHDL, and instantiate a port based on these definitions. Therefore these definitions must match the VHDL file included.

A parameter ' User Defined I/O Modules' is provided in the ADMXRC Board Implementation block to allow script files specifying I/O modules to be inserted.

An example model containig 2 custom I/O modules is provided in *test_ddr_cache.mdl*. This model is for use with the XRM-DDR-IO module.
This model connects the Slave Mode Local Bus interface to a cached DDR-RAM controller. The cached DDR controller allows read and write access to a 2048x32 bit block RAM cache. Data can be transferred to and from this cache to the DDR RAM in 64 byte bursts, by asserting the read or write lines. Each burst takes approximately 11 clock cycles.

The files associated with this custom I/O module are *ddr_cache.txt* and *ddr_cache.vhd*
A simulation Matlab S-function *model_ddrram_cache.m* is also provided.

A second I/O module is included in the design to connect to the 38 pin Mictor connector on the XRM-DDR-IO.

This module provides 32 bi-directional data pins, an external clock output, and clocks the data in using an external clock input, and provides an input asynchronous FIFO. The files associated with this port are:
xrmddrio.txt and *xrmddrio.vhd*

3.7 Creating a new Design

To create a new design from scratch, it is necessary to drag one of the templates from the library (or from the Simulink Library Browser) into a new Simulink Model.

Right click on the block and select "Link options > Disable Link". This will allow you to modify the block. You may also want to select "Link options > Break Link"..

The template contains port specifications and the synthesis blocks. These should not be deleted or renamed, otherwise problems with synthesis will occur. You also should not add new ports, except as debugging aids to export data to Matlab. These should however be deleted before synthesis.

You can build your design between these ports using the Xilinx System Generator Blocks and the memory and I/O blocks provided in the Alpha Data library.

Other Examples
Two other examples are included in the example directory.

fft_example2.mdl is a 64 point FFT example using the SWDR template. This consists of a numerically controlled oscillator which produces a signal samples of a sine wave at the clock frequency. The frequency of the sine wave is controlled by input slave writes to the FPGA. 64 point windows of the signal are grabbed, multiplied by a hamming window function, an FFT performed, the power spectrum of the signal is then computed by squaring and adding the FFT outputs. The original signal and the power spectrum are then output as 16 bit fields across a DDMA FIFO to the host.

fft_example2_on_card.mdl runs the FPGA circuit generated by *fft_example2.mdl* on an FPGA.

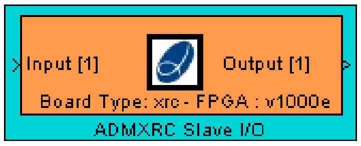
Note that the SWDR and DWSR templates are not supported for the ADM-XPL.

4 ADMXRC Simulink Blockset Reference

4.1 *Hardware in the Loop Blocks*

These blocks allow Simulink to control an FPGA application running on an ADM-XRC board.

4.1.1 Slave I/O



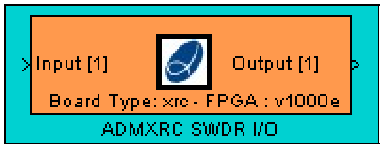
The Slave I/O hardware in the loop block is used to control FPGA applications which require only slave mode reads and writes.

Parameters:
Filename:
The name of the FPGA bitstream to be downloaded

Clock Frequency (MHz):
The frequency to set the local bus clock to. Should not be more than 33Mhz on an XRC and 66MHz on an XRC-II.
Width of Input Vector:
Defines the size of the input vector, minimum value 0, maximum value depends on memory.
Width of Output Vector:
Defines the size of the output vector, minimum value 0, maximum value depends on memory.
Terminate Loop Limit:
Number of loops, that Simulink waits for the FPGA to signal that it has completed.
Card Index:

The index of the FPGA card being controlled.

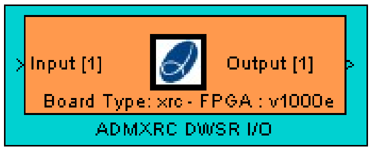
4.1.2 SWDR I/O



The SWDR I/O hardware in the loop block is used to control FPGA applications which require slave mode programmed I/O writes to the FPGA and demand mode DMA reads from the FPGA.

Parameters:
Filename:
The name of the FPGA bitstream to be downloaded
Clock Frequency (MHz):
The frequency to set the local bus clock to. Should not be more than 33Mhz on an XRC and 66MHz on an XRC-II.
M Clock Frequency (MHz):
The frequency to set the M clock on the ADM-XRC board to. This can be up to 100 MHz. The circuit implemented using the SWDR template will run at this clock speed, although the data transfers will be limited to running at the local bus clock frequency. N.B. in designs generated using System Generator for the XRCII and XRCII-L, this clock signal is doubled in frequency using a DCM before being used to drive the circuits, allowing the circuit to be clocked at up to 200MHz (you should specify the MCLK as 100MHz for this).
Width of Input Vector:
Defines the size of the input vector, minimum value 0, maximum value depends on memory.
Width of Output Vector:
Defines the size of the output vector, minimum value 0, maximum value depends on memory.
Card Index:
The index of the FPGA card being controlled.

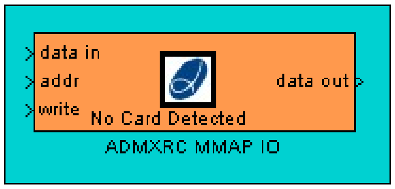
4.1.3 DWSR I/O



The DWSR I/O hardware in the loop block is used to control FPGA applications which require demand mode DMA writes to the FPGA and slave mode programmed I/O reads from the FPGA.

Parameters:
Filename:
The name of the FPGA bitstream to be downloaded
Clock Frequency (MHz):
The frequency to set the local bus clock to. Should not be more than 33Mhz on an XRC and 66MHz on an XRC-II.
M Clock Frequency (MHz):
The frequency to set the M clock on the ADM-XRC board to. This can be up to 100 MHz. The circuit implemented using the DWSR template will run at this clock speed, although the data transfers will be limited to running at the local bus clock frequency. N.B. in designs generated using System Generator for the XRCII and XRCII-L, this clock signal is doubled in frequency using a DCM before being used to drive the circuits, allowing the circuit to be clocked at up to 200MHz (you should specify the MCLK as 100MHz for this).
Width of Input Vector:
Defines the size of the input vector, minimum value 0, maximum value depends on memory.
Width of Output Vector:
Defines the size of the output vector, minimum value 0, maximum value depends on memory.
Card Index:
The index of the FPGA card being controlled.

4.1.4 MMAP I/O



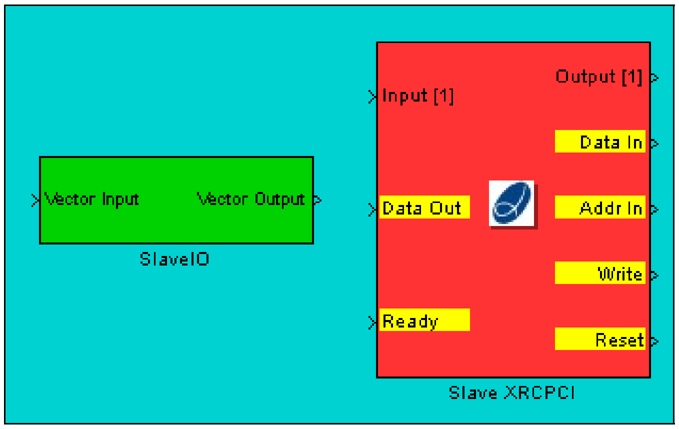
The MMAP I/O hardware in the loop block is used to control FPGA applications which require memory mapped I/O access to the FPGA.

Parameters:
Filename:
The name of the FPGA bitstream to be downloaded
Clock Frequency (MHz):
The frequency to set the local bus clock to. Should not be more than 33Mhz on an XRC and 66MHz on an XRC-II.
M Clock Frequency (MHz):
The frequency to set the M clock on the ADM-XRC board to. This can be up to 100 MHz. The circuit implemented using the MMAP template will run at this clock speed, although the data transfers will be limited to running at the local bus clock frequency. N.B. in designs generated using System Generator for the XRCII and XRCII-L, this clock signal is doubled in frequency using a DCM before being used to drive the circuits, allowing the circuit to be clocked at up to 200MHz (you should specify the MCLK as 100MHz for this).
Card Index:
The index of the FPGA card being controlled.

4.2 PCI Interface and Template Blocks

These blocks provide templates for FPGA design on ADM-XRC cards. They include one of 4 PCI interface configurations along with the system generator and board implementation tokens. They should be inserted at the highest level of the design and should provide a functionally equivalent block to the hardware in the loop blocks..

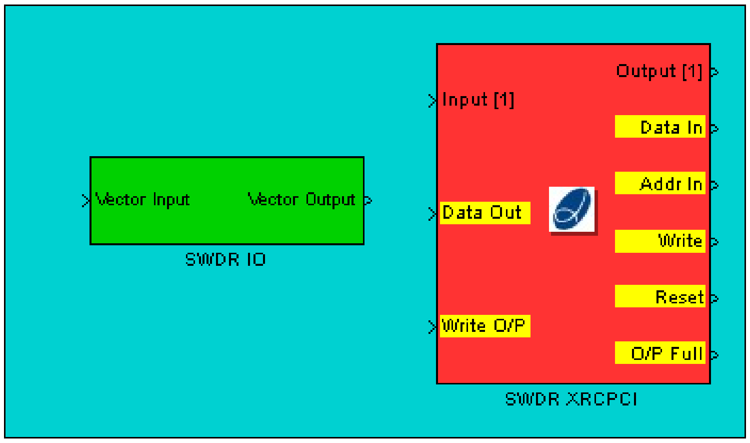
4.2.1 Slave I/O Template



The Slave I/O Template provides a template for building FPGA applications which use Slave I/O to communicate across the PCI bus. This block is suitable for simple examples and designs which require only small data transfers between the host and the FPGA. The parameters for the PCI block are:

Parameters:
Number of Inputs:
Defines the size of the input vector, minimum value 0, maximum value depends on memory.
Number of Outputs:
Defines the size of the output vector, minimum value 0, maximum value depends on memory.
Circuit Lag:
This defines how long the simulation waits between the last write and starting to read the data. N.B. In the simulation, Simulink does not wait on the ready signal.
Decode Delay:
This determines how many clock cycles after setting a read address are waited before the output is assumed to be valid.

4.2.2 SWDR Template (Slave Write DDMA Read)

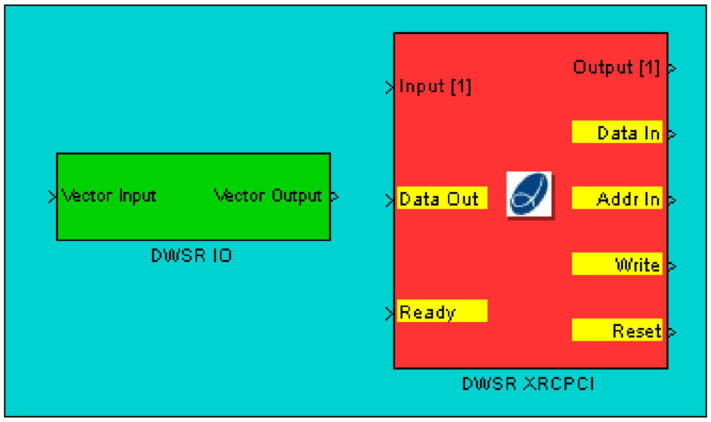


The SWDR I/O Template provides a template for building FPGA applications with Slave mode Writes (to the FPGA) and Demand mode DMA Reads (from the FPGA). The interaction with the local bus is asynchronous allowing the system generator circuit to run at a different clock rate. This template is ideal for applications which require only a few parameters to be controlled by the host, but which transfer a large amount of data back to the host (possibly data coming in from the other I/O ports).

The parameters for the PCI block are:

Parameters:
Number of Inputs:
Defines the size of the input vector, minimum value 0, maximum value depends on memory.
Number of Outputs:
Defines the size of the output vector, minimum value 0, maximum value depends on memory.
Circuit Lag:
This defines how long the simulation waits between the last write and starting to read the data.
FIFO Length:
This defines how large a FIFO is simulated. This does not affect the size of the implemented FIFOs.
Debug Flag:
Turns on some debug info. Should be left as 0.

4.2.3 DWSR Template (DDMA Write Slave Read)

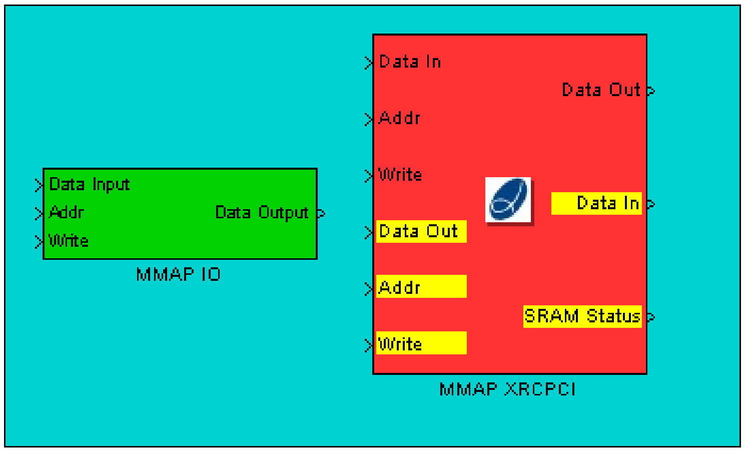


The DWSR Template provides a template for building FPGA applications with Demand mode DMA Writes (to the FPGA) and Slave mode Reads (from the FPGA). The interaction with the local bus is asynchronous allowing the system generator circuit to run at a different clock rate. This template is ideal for applications which process a large amount of data from the host, but require only a few result data to be read back to the host.

The parameters for the PCI block are:

Parameters:
Number of Inputs:
Defines the size of the input vector, minimum value 0, maximum value depends on memory.
Number of Outputs:
Defines the size of the output vector, minimum value 0, maximum value depends on memory.
Circuit Lag:
This defines how long the simulation waits between the last write and starting to read the data. N.B. In the simulation, Simulink does not wait on the ready signal.
Decode Delay:
This determines how many clock cycles after setting a read address are waited before the output is assumed to be valid.
FIFO Length:
This defines how large a FIFO is simulated. This does not affect the size of the implemented FIFOs.

4.2.4 MMAP Template (Memory Mapped)

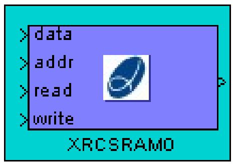


The MMAP Template provides a template for building FPGA applications with Memory Mapped I/O between the PCI host and FPGA applications. Communication occurs though a 512 word asynchronous dual port block RAM which can be accessed from either side. Unlike the Slave Mode template, the FPGA application can run asynchronously to the PCI bus. This template also provides the PCI host direct access to any SRAM block instantiated in the design. The 6 bit SRAM status is used to indicate if any of the 6 SRAM banks is being accessed by the host across PCI. In this condition, the FPGA aplpication will not be able to access that SRAM bank. There are no parameters for the PCI block. The host accesses the SRAM banks through a special register at address 0x1FFFFC. Bit 3 is used to select whether the host is accessing SRAM or the asynchronous dual port block RAM. Bits 0-2 select the SRAM bank. i.e. writing '8' to 0x1FFFFC, will allow accesses to addresses 0-1Mb top access SRAM bank 0, and will also set bit 0 of SRAM status to 1. Writing '0' will allow the host to access the block RAM. The FPGA application only accesses the block RAM through this interface, access to SRAM is through instantiating the individual SRAM blocks.

4.3 Memory Blocks

These blocks provide access to some of the external SRAM. These blocks should be inserted into the System Generator Design within the Simulink Model.

4.3.1 XRCSRAMx



This block models the SRAMs in the ADM-XRC and ADM-XRC-II boards.
data is written into *address* when *write* is asserted. The data at *address* is available at the output 4 clock cycles after *read* is asserted.

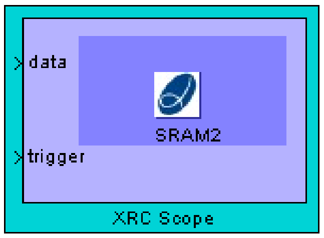
Separate models are supplied for each RAM bank. To include multiple RAM banks you should copy each individually from the Alpha Data library. Synthesis errors will occur if you simply change the last digit of a bank already in the design.

The SRAM memory is stored during simulation in the global array SRAM, which is a 6 by 1048576 array. This can be read after the Simulation has finished for debugging purposes.

With the ADM-XPL, the SRAM bank is 64 bits wide. A special SRAM block is provided for this card: XRCSRAM7. This block has two 32 bit wide data ports, but otherwise operates almost identically to the other SRAM modules. In simulation, the data is actually stored in SRAM(1,:) and SRAM(2,:) and can be read for debugging purposes.

When using the MMAP template with the ADM-XPL, the SRAM can be accessed from the Local Bus. When read from the local bus, bank 7 is mapped to bank 0 for the lower 32 bits and bank 1 for the upper 32 bits. However when writing from the local bus, data is only written into the lower 32 bits of the memory, additionally the upper 32 bits of each memory address are cleared by this action.

4.3.2 XRC Scope



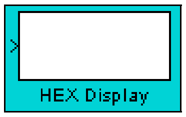
This block provides a debugging aid. When the *trigger* is asserted, 256k of data is recorded in an SRAM bank, 32 bits of *data* are recorded for 65536 successive clock cycles.

After the template simulation has finished, the results are recorded in Matlab variables XRCScopeData and XRCScopeState. After a hardware in the loop simulation has been run, the data can be read from the SRAM by using the Matlab function *read_rambank*

4.4 Miscellaneous Blocks

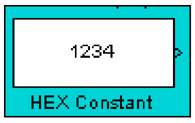
These blocks are provided to make the implementation of the example models easier. They provide simple methods for producing int32 data in Simulink which is easily handled by the FPGA.

4.4.1 HEX Display



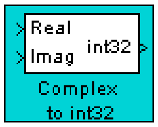
This block allows an int32 Simulink signal to be displayed as a HEX value.

4.4.2 HEX Constant



This block provides an easy way of generating int32 data from a constant HEX value.

4.4.3 Complex to INT32



This block provides efficient data conversion between 2 double signals and an int32 signal which has the lower 16 bits representing the real part and the upper 16 bits representing the imaginary part.

4.4.4 INT32 to Complex

int32
Real
Imag
int32 to
Complex

This block provides efficient data conversion from an int32 signal where the lower 16 bits represent the real part of a signal and the upper 16 bits represent the imaginary part, into two double signals.

4.5 Extra Matlab Functions

The simulink blockset also includes a number of useful Matlab functions.

4.5.1 sdec2hex

This function provides decimal to hexadecimal conversion which does not fail if presented with a negative number.
USAGE:

```
>> y=sdec2hex(x)
```

4.5.2 read_rambank

This function reads data from a RAM bank. It is a debugging aid to be used in conjunction with the XRC Scope Block.

USAGE:

```
>> y=read_rambank(card_index, bank)
```

This returns a 64k array of doubles containing a representation of the contents of the SRAM bank. N.B. This function assumes that the FPGA application has been created using System Generator and the XRC Board Application Library. This will not work on all FPGA bitstreams.

4.5.3 split_signal

This function breaks the data returned by read_rambank into a specified format.

USAGE:

```
>> x= split_signal(signal,format)
```

signal should be (part of) the array returned from *read_rambank*.

format should have the form {type,bits;type,bits; …}
where type is either 'u' or 's' to interpret the bits as signed or unsigned
and bits are interpreted in ascending order i.e. should be 0:7 (not 7:-1:0)

4.6 Custom Templates

Customisation of the template VHDL is also supported. There will be few applications where this is really necessary, however, this will allow the circuit to be driven using an I/O clock pin, or allow an application specific I/O block to be added which is more appropriate than the general I/O ports provided by the library.

There are 3 stages to creating a custom template.

4.6.1 Specifying the ports in System Generator

Again, in the system generator model, extra ports can be added by including extra Gateway (yellow) blocks. In the top level System Generator generated VHDL entity these ports will appear as inputs or outputs.

4.6.2 Modifying the VHDL

The VHDL for the top level module created when using the ADMXRC templates is generated by modifying a template file found in directory admxrc_sb/src.

The templates are:
ADMXRC_SLAVE_TEMPLATE.vhd,
ADMXRC_SLAVE_TEMPLATE_XPL.vhd,
ADMXRC_MMAP_TEMPLATE.vhd,
ADMXRC_MMAP_TEMPLATE_XRC2.vhd,
ADMXRC_MMAP_TEMPLATE_XPL.vhd,
ADMXRC_SWDR_TEMPLATE.vhd,
ADMXRC_SWDR_TEMPLATE_XRC2.vhd,
ADMXRC_DWSR_TEMPLATE.vhd,

The file corresponding to the XRC PCI I/O block being used in the design should be copied to a new directory and renamed. N.B. due to differences between the PLX 9656 (used on XRC-II) and PLX 9080 (used on the XRC, XRC-P and XRCII-L) there are different templates for demand mode DMA data transfer from the FPGA to the host PC.

The first modification that is required is to change the module entity name to ADMXRC_CUSTOM_TEMPLATE. This will be renamed to your design name when synthesized.

The extra ports added to the SysGen module have to added to the component declaration and instantiation of SysGen in this file. You can also add all the VHDL required to process these new outputs to this file. If you wish to use a different clock source for the SysGen module, you can connect a different clock. N.B. it may make more sense to connect your custom clock to the lclk port of the clocks module (DCM/DLL) as this will then also drive the SRAM modules in the design.

4.6.3 Including the custom VHDL in the Synthesis

The ADMXRC Library needs to be told to use your custom VHDL file and where to find it. This is done by modifying the mask parameter in the "ADMXRC Board Implementation" block.

Right click on the "ADMXRC Board Implementation" and select "Mask Parameters" Enter the full path name of your custom template file into the field "Custom Template File" and click on OK. (If this field is left blank, then the default template file is used)