XRC Board Level Application Library v2.3c
(For Simulink and System Generator)

Alpha Data Parallel Systems Ltd.
4 West Silvermills Lane
Edinburgh EH3 5BD
Scotland, UK

Phone: +44 (0) 131 558 2600
Fax: +44 (0) 131 558 2700
Email: support@alpha-data.com

1 Introduction

The Alpha Data XRC Board Level Application Library 2.3 is a collection of embedded IP, VHDL, Simulation Models, and a top level Wrapper Builder application, all designed to ease the design of FPGA applications on Alpha Data Embedded PMC and PCI cards. These modules are designed to work with Xilinx System Generator 8.1, and Matlab/Simulink 7.1(sp3) (R14sp3) from the Mathworks.

1.1 Whats New

Updates from 2.3b to 2.3c.

Support for non-normalised sample periods. (i.e. Simulink Period now no longer required to be 1). Clock wrapper used in synthesis.

Updates from 2.3a to 2.3b.

Memory blocks now fully supported with co-simulation.
Tri-state port instantiation.

Updates from 2.2 to 2.3.

The wrapper builder now automatically adds user defined pin locations specified in Gateway blocks to the UCF file.

2 UCF files are generated: one with XST style bus bit delimiters : <> and one with Synplify style bus bit delimiters ().

An additional DEBUG port has been added to the PCI interface blocks to allow debugging modules to be added such as the user generated co-simulation block.

The wrapper now supports single step clocking from within the PCI interface, primarily to support Co-simulation.

Co-simulation support has been added. This allows user definition of a set of "Gateways" to define an interface to the Simulink environment. A GUI Wizard is used to specify the ports which are for cosimulation. These connections appear as gateways in Simulation, but are remapped to PCI transfers when building the bitstream. A Simulink block for running the cosimulation in hardware is also generated, and this can control the hardware.

An auto-build Matlab script has been added to allow bitstream generation from the VHDL without using the ISE GUI. This is however optional, and advanced users can Synthesize and Implement the design using their own choice of design tools such as Synplify, PlanAhead and/or ISE.

This blockset now has limited support for the ADP-XPI board. PCI Interface is supported. 32bits of DDR-II SSRAM is supported with ADLOCB dual port. Co-simulation is supported. DDR SDRAM DIMMs are supported with 64-bit wide models, but no ADLOCB dual port.

2 Installation Instructions

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

xrc_application_blockset.zip
xrc_application_blockset_plugins.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

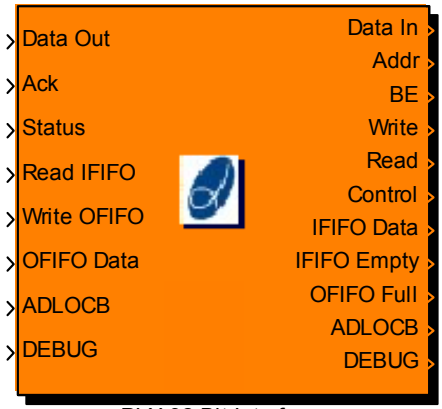
4) Quit Matlab

3 XRC Simulink Blockset

The XRC Blockset provides simulation and synthesis models to handle the most common I/O tasks in the ADM-XRC Series of cards. Communication with the host is through a Local Bus bridge and PCI. The modules provided have control and status registers (for simple control; tasks), a slow, asynchronous, memory mapped, slave interface (for more sophisticated control and small data transfers), and input and output demand mode DMA FIFO's for large high speed data transfers to and from the host. The other modules provided interface to the off-chip memory on each card (ZBT, DDR-SDRAM, DDR-II SSRAM). Other I/O is generally unidirectional and synchronous to the System Generator clock domain, and this can be implemented using the standard System Generator I/O ports. All the modules contain ADLOCB ports. This allows the components to be chained together on a single bus controlled by the host, and allow fast background access to the off-chip memory.

3.1 Local Bus Interface Modules

3.1.1 LBIF_PLX32



This interface connects to the PLX9656 PCI bridge chip used on the ADM-XRC-II. It provides the following ports:

Control : (UFix_32_0) -- control register value – set by host at address 0x40000
Status : (Ufix_32_0) -- status register – read by host at address 0x40040

The slave memory mapped interface uses the following ports:
Data_In: (Ufix_32_0) -- data written by host
Addr: (UFix_16_0) -- 32 bit address for read or write
BE: (Ufix_4_0) -- byte enables

Write: (Boolean) -- host wants to write data (Data_Write) to Address (Addr)
Read: (Boolean) -- host wants to read data from Address(Addr)
Data_Out: (UFix_32_0) -- data to be returned to host
Ack: (Boolean) -- indicates that Data_Read is valid – must be asserted in response to Read.

The slave memory is mapped to local bus addresses 0x000000-0x03FFFC

The Demand Mode DMA FIFOs are connected to the PLX9656 DMA channels, with writes from the host on DMA channel 0 resulting in Data Being pushed into IFIFO, and reads from the host on DMA channel 1, pops data out of OFIFO. Since Demand Mode DMA is used the host process will block if IFIFO is full or OFIFO is empty.

IFIFO Ports are:
IFIFO_Data: (UFix_32_0)
IFIFO_Empty: (Boolean)
Read_IFIFO: (Boolean)

OFIFO Ports are:
OFIFO_Data: (UFix_32_0)
OFIFO_Full: (Boolean)
Write_OFIFO: (Boolean)

The following C-Code for accessing these can be used. The Demand Mode DMA application in the ADM-XRC SDK should also be consulted to see how to set up Demand Mode DMA transfers. With this module the DMA counters at Local Bus locations 0x040080 and 0x0400C0, and these should be set before starting the DMA transfer.

```
void sendData(int offset, int size)
{
  ADMXRC2_STATUS              status;
  HANDLE                             event;

  event = CreateEvent(NULL, TRUE, FALSE, NULL);
  /*
  ** Program the PCI-to-local transfer counter.
  */
  fpgaSpace[65568] = size;
  fpgaSpace[65568];

  /*
  ** Perform the DMA transfer.
  */
  status = ADMXRC2_DoDMA(card,sendbufHandle,offset,size,0x40,
                         ADMXRC2_PCITOLOCAL,0,mode,0,NULL,event);
  CloseHandle(event);
}

void recvData(int offset, int size)
{
  ADMXRC2_STATUS              status;
  HANDLE                             event;
```

```
  event = CreateEvent(NULL, TRUE, FALSE, NULL);

  /*
  ** Program the local-to-PCI transfer counter.
  */
  fpgaSpace[65584] = size;
  fpgaSpace[65584];

  /*
  ** Perform the DMA transfer.
  */
  status = ADMXRC2_DoDMA(card,recvbufHandle,offset,size,0x80,
                         ADMXRC2_LOCALTOPCI,1,mode,0,NULL,event);
   CloseHandle(event);
}
```

Two parameters are provided for the Local Bus Interface Block, the first "Local Bus script Program" is used to simulate Local Bus transactions. The second parameter "Clock Ratio" sets the ratio used in simulation between Local Bus Clock Cycles and the System Generator System Clock e.g. if this is set to 1.5 then the Local Bus program will update its outputs every 1.5 System Clock cycles.

The Local Bus Script Program can be specified as a string or as a Matlab variable. If either of these is specified as load <filename.txt> then the file <filename.txt> will be loaded and parsed in place of the string.

The following commands are parsed:

slave_read <address>
reads the contents of a local bus address and displays the result

slave_write <address> <data> [be]
write 32 bit integer <data> to address with optional byte enable signal

dma_read <length> [matlab variable]
reads <length> words from OFIFO and either displays them or stores them in a a [matlab variable]

dma_write <length> [matlab variable]
writes <length> words into IFIFO. If a [matlab variable] is specifed then this is used as data otherwise integers 1..<length> are sent

sleep <count>
this pauses the simulation program for <count> LCLK cycles

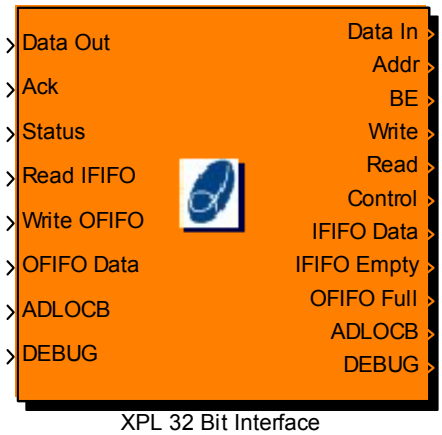
Example Program:

```
slave_write 0x40000 0xabcd1234
slave_read 0x40000
slave_read 0x40040
slave_write 0x0 0x1234abcd
dma_write 8 myfifo_in
dma_read 8 myfifo
slave_read 0x0
```

N.B. There is a 32K limit on the size of the program.

The ADLOCB ports provide a ring bus for connecting up external memory components and accessing the over PCI.
The DEBUG ports are used for co-simulation.

3.1.2 LBIF_XPL32

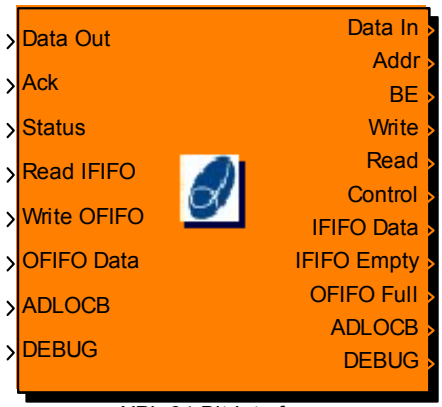


XPL 32 Bit Interface

This interface connects to the XPL PCI-Local Bus Bridge used on the ADM-XPL and ADM-XP. Functionally this module performs identically to the LBIF_PLX32 module. In implementation there are minor differences in the VHDL top level ports produced, as the data and address pins are multiplexed with the XPL bridge, unlike the PLX9656.

N.B. Due to the default clock frequency ratio between the sysgen clock domain and the Local Bus clock domain, special consideration has to be taken when generating Data Out. The asynchronous transfer register used to sample Data Out, will sample Data Out between ½ and 1 Local Bus clock cycles after Ack is asserted. Since the Sysgen clock frequency is by default twice the Local Bus clock frequency, Data Out should be held for 1 clock cycle after Ack is asserted to ensure correct transfer.

3.1.3 LBIF_XPL64



XPL 64 Bit Interface

This module is similar to LBIF_PLX32 and LBIF_XPL32, however the data widths are 64 bits wide. The ports therefore are

Control : (UFix_64_0)
Status : (Ufix_64_0)
Data_In: (Ufix_64_0)
Addr: (UFix_16_0)
BE: (Ufix_8_0)
Write: (Boolean)
Read: (Boolean)
Data_Out: (UFix_64_0)
Ack: (Boolean)
IFIFO_Data: (UFix_64_0)
IFIFO_Empty: (Boolean)
Read_IFIFO: (Boolean)
OFIFO_Data: (UFix_64_0)
OFIFO_Full: (Boolean)
Write_OFIFO: (Boolean)

The host application should set the Local Bus Space up to allow 64 bit access. The DMA Mode bit for 64 bit access should also be set.

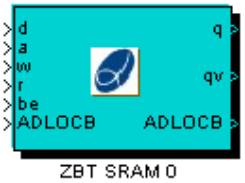
A similar format is used for the Local Bus Script Program but with one minor changes to deal with 64 bit values:
slave_write <address> <data0> [data1] [be]
will write data0 to the lower 32 bits and data1 to the upper 32 bits if specified

N.B. the address input does not change and is still refers to a 32 bit word address. So bit 0 may be ignored if 64 bits transfers are being used.

3.2 Memory Modules

Each card has a number of memory interface modules to allow access to the off-chip memory. With the SRAM modules, the memory is accessed directly. With the DRAM modules, assess is via a cache. For simulation, it is possible to initialise the memory from Matlab by specifying the contents of the variables: *sram0, sram1, sram2* etc. for the SRAM banks, and *dram0, dram1* for the DRAM banks. Each element is converted into a 32 bit integer. For 64 bit memories, 2 elements are written to each location with the lower 32 bits written first. When the simulation finishes, the memory contents are written back into the Matlab variables, however the size of the variable is used to determine how much data is transferred back to Matlab.

3.2.1 ZBT SRAM (32 bit)

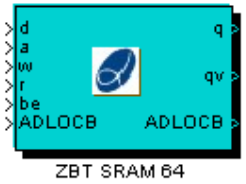


These modules implement the 32bit ZBT SRAM on the ADM-XRC-II.
Ports are:
d : UFix_32_0
a : UFix_20_0
w : Boolean
r : Boolean
be : UFix_4_0
q : UFix_32_0
qv : Boolean

If "w" is asserted then data of "d" is stored in the SRAM at address "a".
If "r" is asserted then SRAM address "a" is read and the output appear on "q" 4 clock cycles later, with "qv" going high to indicate that it is valid.

The modules for the 32 bit ZBT SRAM on the ADM-XRC-4LX and ADM-XRC-4SX are almost identical. The only minor difference is that the address input 'a' is 21 bits wide rather than 20.

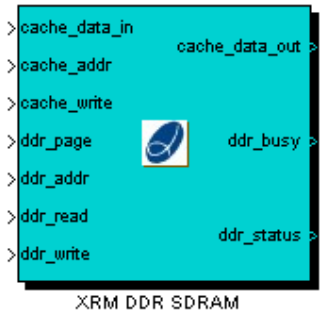
3.2.2 ZBT SRAM (64 bit)



This module implement the 64bit ZBT SRAM on the ADM-XPL.
Ports are:
d : UFix_64_0
a : UFix_20_0
w : Boolean
r : Boolean
be : UFix_8_0
q : UFix_64_0
qv : Boolean

If "w" is asserted then data of "d" is stored in the SRAM at address "a".
If "r" is asserted then SRAM address "a" is read and the output appear on "q" 4 clock cycles later, with "qv" going high to indicate that it is valid.

3.2.3 XRM-DDR SDRAM (64 bit)



This module implements and interface to the XRM-DDR SDRAM on the XRM-DDR module. The data interface to the System Generator module is via an 8K cache memory block RAM, with a 1 clock cycle latency.

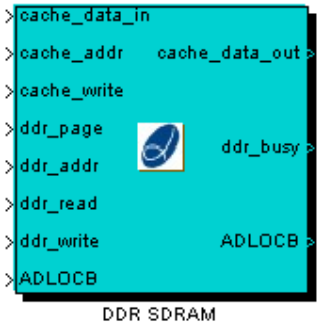
Ports are:
cache_data_in : UFix_32_0
cache_data_out : UFix_32_0
cache_addr: UFix_11_0
cache_write: Boolean

Control signals are used to burst the data from the 8K cache to the DDR
ddr_page : UFix_14_0 is used to select an 8K page in the DDR
ddr_addr : UFix_4_0 Selects a 512 byte block within the cache
ddr_read : Boolean – when asserted the 512 byte block is copied from DDR to Cache
ddr_write : Boolean – when asserted the 512 byte block is copied from Cache to DDR
ddr_busy : Boolean -- Indicates whether a burst is in operation or not. While ddr_busy is high, ddr_read and ddr_write will be ignored.

The time for a burst will vary as it may have to wait for a refresh to finish, however each burst should take between 36 (usual) and 43 (worst case) clock cycles.

N.B. The XRM DDR SDRAM module does not currently support ADLOCB connection.

3.2.4 DDR SDRAM (32 bit)



These modules implement the interface to the DDR SDRAM on the ADM-XPL and ADM-XP. The data interface to the System Generator module is via a 4K cache memory block RAM, with a 1 clock cycle latency.

Ports are:
cache_data_in : UFix_32_0
cache_data_out : UFix_32_0
cache_addr: UFix_10_0
cache_write: Boolean

Control signals are used to burst the data from the 4K cache to the DDR
ddr_page : UFix_15_0 is used to select an 4K page in the DDR
ddr_addr : UFix_4_0 Selects a 256 byte block within the cache
ddr_read : Boolean – when asserted the 256 byte block is copied from DDR to Cache
ddr_write : Boolean – when asserted the 256 byte block is copied from Cache to DDR
ddr_busy : Boolean -- Indicates whether a burst is in operation or not. While ddr_busy is high, ddr_read and ddr_write will be ignored.

The time for a burst will vary as it may have to wait for a refresh to finish, however each burst should take between 36 (usual) and 43 (worst case) clock cycles.

3.2.5 DDR-II SSRAM

These modules implement the interface to the DDR-II SSRAM on the ADM-XP. The 32 bit DDR ports on the devices are mapped to single clock 64 bit wide ports. The SSRAM devices have a burst length of 4, and therefore the minimum transfer to the

RAM will take 2 clock cycles. This creates some operational limitations of the use of these devices.

Ports are:
d : UFix_64_0
a : UFix_21_0
w : Boolean
r : Boolean
be : UFix_8_0
q : UFix_64_0
qv : Boolean

For writes, the DDR-II controller waits for a second write "w" before bursting both data elements to the SRAM, in successive locations. The addresses used will be "a" when the first write is asserted and "a"+1. The byte enables can be specified separately for each write. Write bursts can be continuous, but should contain an even number of writes. "a" should be incremented every write.

Read "r" should not be asserted after an odd number of writes. It should also not be asserted one clock cycle after the last write.

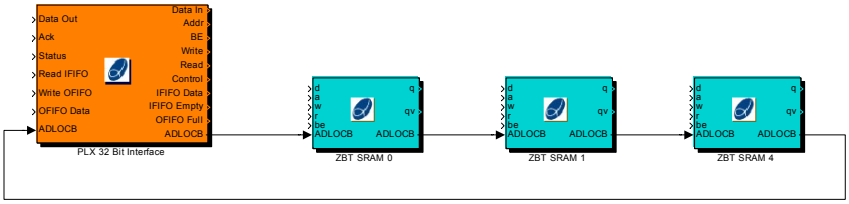
Reads to an even address "a(0)"= 0 will generate two reads 3 and 4 clock cycles after "r" is asserted. "qv" will indicate that the data in "q" is valid. For reads to an odd address "a(0)" =1, one read will be generated and the data will be valid after 4 clock cycles.

Read bursts can be continuous, for as long as "r" is asserted, and "a" is incremented every clock cycle.

For correct operation the DDR-II-SSRAM must be clocked at over 75MHz and less than 133MHz.

3.3 ADLOCB

The Alpha Data Local to On Chip Bus interface provides a fast, simple to use mechanism for high speed data transfers between the host and off-chip memory with minimal impact on the System Generator design logic.



The ADLOCB requires a Local Bus Interface module to act as the master. RAM modules are then connected in a ring as slaves. Finally the last RAM output should be connected back to the Local Bus Interface to provide a data path for reading data.

From the host, the ADLOCB is mapped into the FPGA into a 2MB window at addresses 0x200000 to 0x3FFFFF. A page/device register is provided at address 0x040100. The lower 16 bits are used to identify the off chip memory device. The SRAM devices are indexed first with the DRAM device numbers depending on the number of SRAM devices on the board. On the XRC-II, ZBT SRAM banks 0 to 5 have device numbers 0 to 5. On the XPL, the ZBT SRAM is device 0 and the DDR SDRAM is device 1. On the XP, the DDR SRAM banks have device numbers 0 to 3 and the DDR DRAM banks have device numbers 4 and 5.

The upper 16 bits (0x040102) of the register are used to select the upper address bits of the memory if the RAM bank is larger than 2MB.

The ADLOCB system does not include any mutual exclusion logic. Host accesses will have higher priority than System Generator Logic accesses, however simultaneous access to any memory may produce unexpected results. The application should implement the appropriate mutual exclusion logic, using the status and control registers to indicate when it is safe for the host and System Generator to access memories.

The Cache in the SDRAM also introduces a twist in the data on a 16 bit boundary. 32 bit elements D0 and D1 at addresses A0, A0+1, appear on the Local bus in the order:
LA(0) : LD(31:0) = [D1(15:0) D0(15:0)]
LA(1) : LD(31:0) = [D1(31:16) D0(31:16)]

3.4 Special Clock Ports

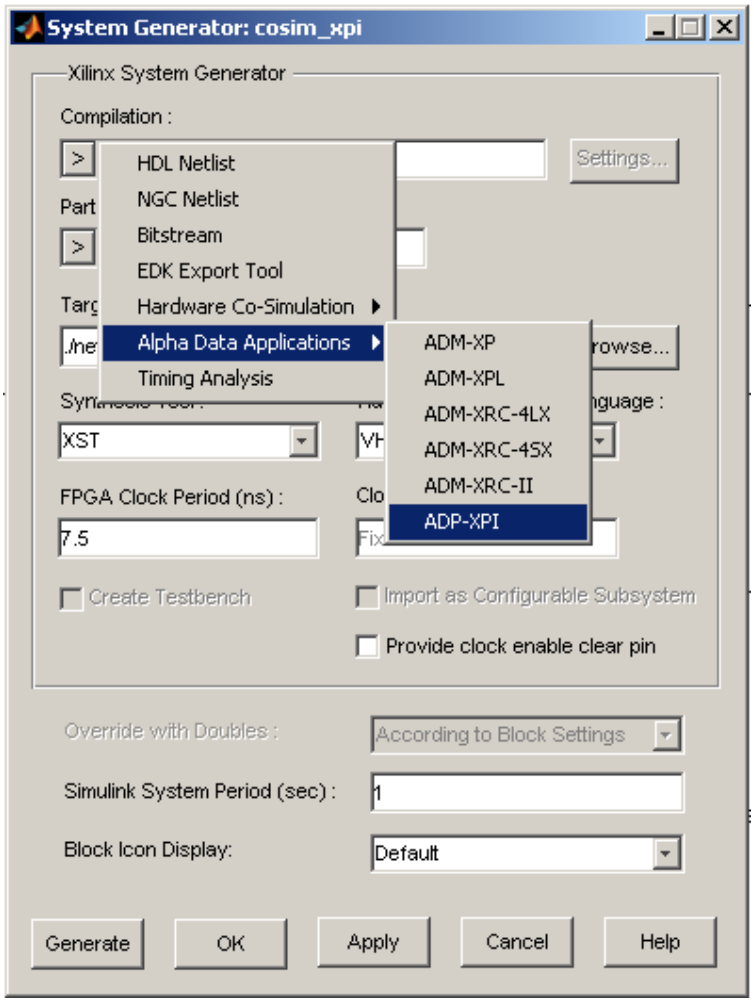
It is possible to bring LCLK and MCLK directly into design for connecting to black box logic, possibly running asynchronously from the rest of the design. Naming a single bit Boolean Gateway in as “usr_lclk” or “usr_mclk” will connect to the LCLK or MCLK global clock signal.

3.5 Tri-State Ports

It is now possible to instantiate user tri-state ports in the top level VHDL. The wrapper generation program reads the user input and output gateways and if it finds 3 ports with the names <name>_t, <name>_o, <name>_i (where <name> is a user specified name), and <name>_t and <name>_o are output ports and <name>_i is an input port, and all 3 ports have the same size and type specification, then the wrapper will instantiate IOBUFs for these ports, and generate an external inout port <name> at the top level. The pin constraints for the port should be put in the <name>_t output gateway block.

4 Wrapper File Generation

The VHDL produced by System Generator cannot handle all the interfacing requirements for the ADM-XRC series cards. Therefore the VHDL module produced must be wrapped before synthesizing and generating a bitstream.



A new compilation option has therefore been added to the System Generator Token. When Generate is run with this option selected this will run through the System Generation process to produce VHDL. The wrapper builder will the run and create a vhdl file, a ucf. (<design>_top.vhd, <design>_top.ucf).

This wrapper builder will identify all the external ports associated with the embedded IP modules. It will then connect up the appropriate clocking circuits and tri-state buffers. It will however not identify any user defined ports (in Gateway blocks). The pin constraints for these will appear in the System Generator output constraint file (the .xcf file with version 8.1). The top level .ucf constraints and the user defined

LOC constraints should be combined into a single UCF file using a text editor before running ISE Project Navigator.

If the wrapper builder detects that the wrapper files have already been generated it will provide the option to not rebuild them. It is only necessary to rebuild them if the ports in the System Generator design have changed. (i.e. after the addition or deletion of any PCI, RAM or Gateway blocks.) Selecting Yes will rebuild the wrapper files, and will also back up the old files; selecting No, will leave the wrapper files unchanged.

The wrapper building stage can be avoided completely by changing the Compilation Option back to HDL netlist, which will limit the changes to the System Generator VHDL.

In many cases the default generated wrapper VHDL and UCF might not quite match the users requirements. However these can be modified, (e.g. to change the system clock pin, or add in timing or area group constraints) with any text editor or within the Project Navigator environment. In these cases after each generation the user should select not to rebuild the wrapper files, or change the Compilation Option back to HDL netlist.

Two UCF files are generated: <design>_top.ucf and <design>_top_curly.ucf.
These files are identical except for the the bus delimiter symbol used.

<design>_top.ucf used the XST default style <>

<design>_top_curly.ucf used the Synplify default stye ()

N.B. if you open the auto-generated System Generator ISE project (<design>.ise), you should add the file <design>_top.vhd and set it as the top level. You should then add the file <design>_top_curly.ucf as that ISE project has XST configured to use the Synplify style bus delimiters.

5 Auto-Build Script

After building the wrapper VHDL and UCF files, the design can be loaded into ISE to build the bitstream.

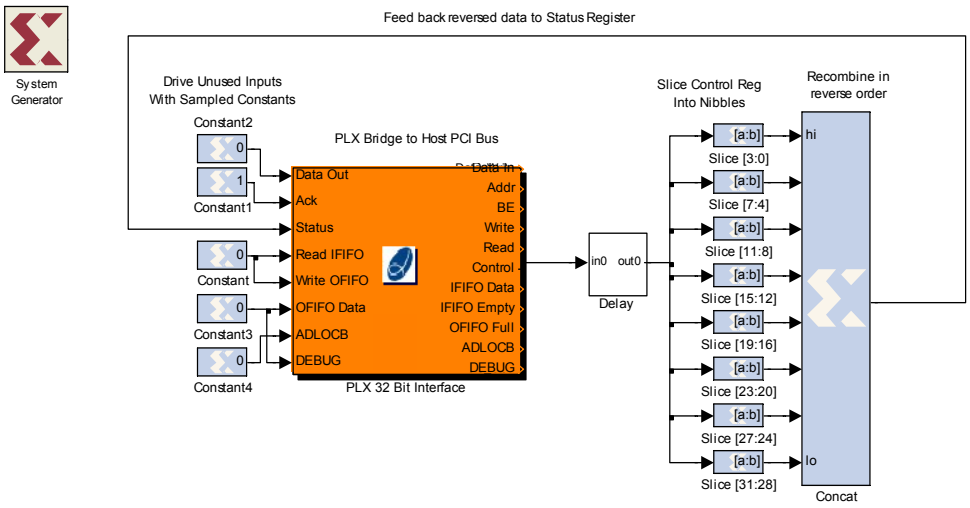
Another option, for simpler designs, is to use the auto-build Matlab script ad_build_bitstream.m. This only requires the netlist directory to be provided as a parameter and it will create <design>_top.bit in that directory. i.e.

>> ad_build_bitstream('netlist')
will run XST etc. and build a bitstream in the directory ‘netlist’, assuming that netlist was the System Generator target directory.

6 Example Applications

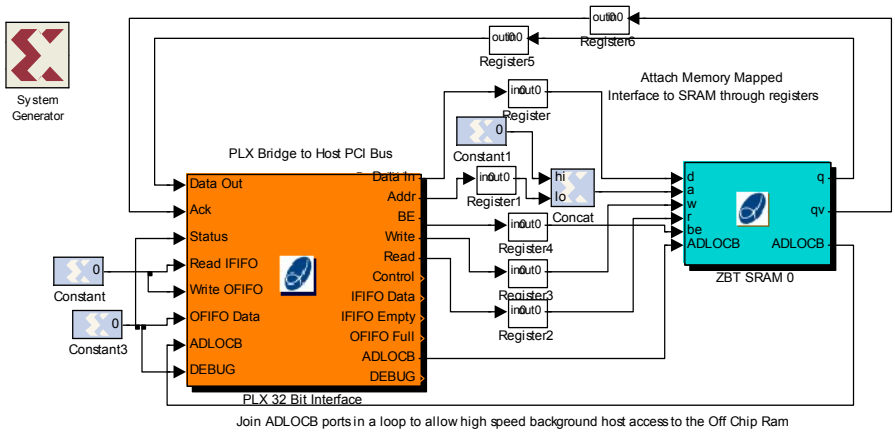
Three simple applications are provided for each board. These applications demonstrate basic connections to the Local Bus Interface, basic I/O and a simple memory application. Separate applications for each board are provided in examples/admxrc2, examples/admxpl, examples/admxp, , examples/admxrc4, examples/adpxpi. Matlab scripts test_simple.m and test_memory.m are provided in the examples directory to run these applications from within Matlab.

6.1 NibRev



This example here shows the simplest possible interface using the PLX/XPL Interface Module. Data from the control register is nibble reversed and then output to the status register. This example formerly named simple has been renamed nibrev.mdl to avoid confusion with the functionally similar simple.vhd example in the ADM-XRC SDK.

6.2 Memory



This example demonstrates how to connect up a simple memory mapped interface. It also shows how to connect up the ADLOCB ports in the modules.

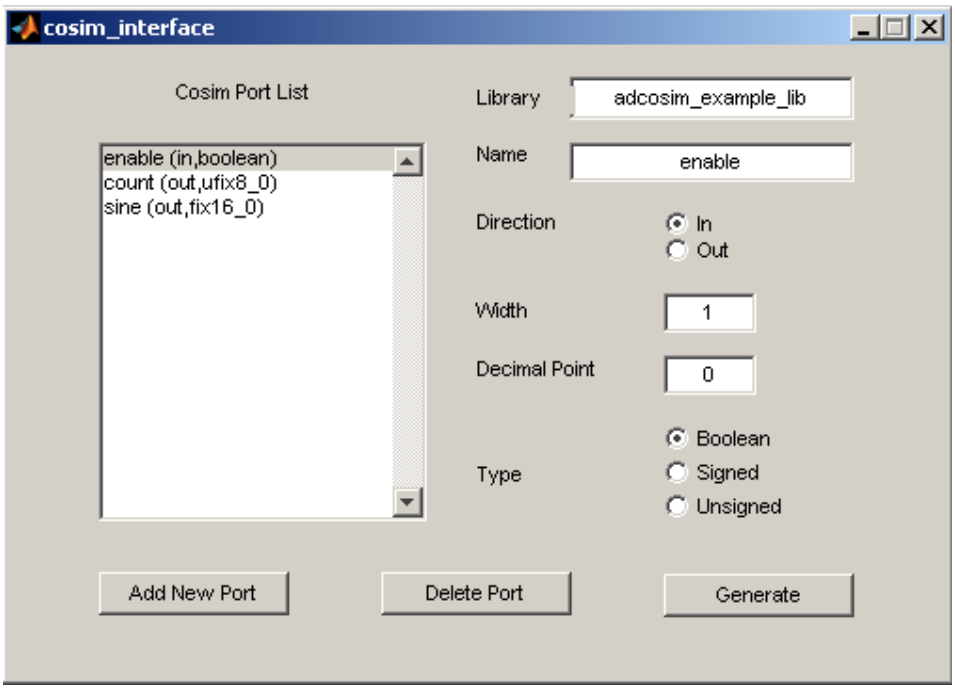
7 Co-Simulation

Co-simulation using the ADM-XRC Applications blockset operates slight differently from that of previous ADM-XRC Cosim blocksets and that of other cosimulation targets such as the Xilinx ML402.

The main reason for this is to allow easy user specification of Ports (Gateways) which are actually external pins, and not connect everything to the co-simulation interface.

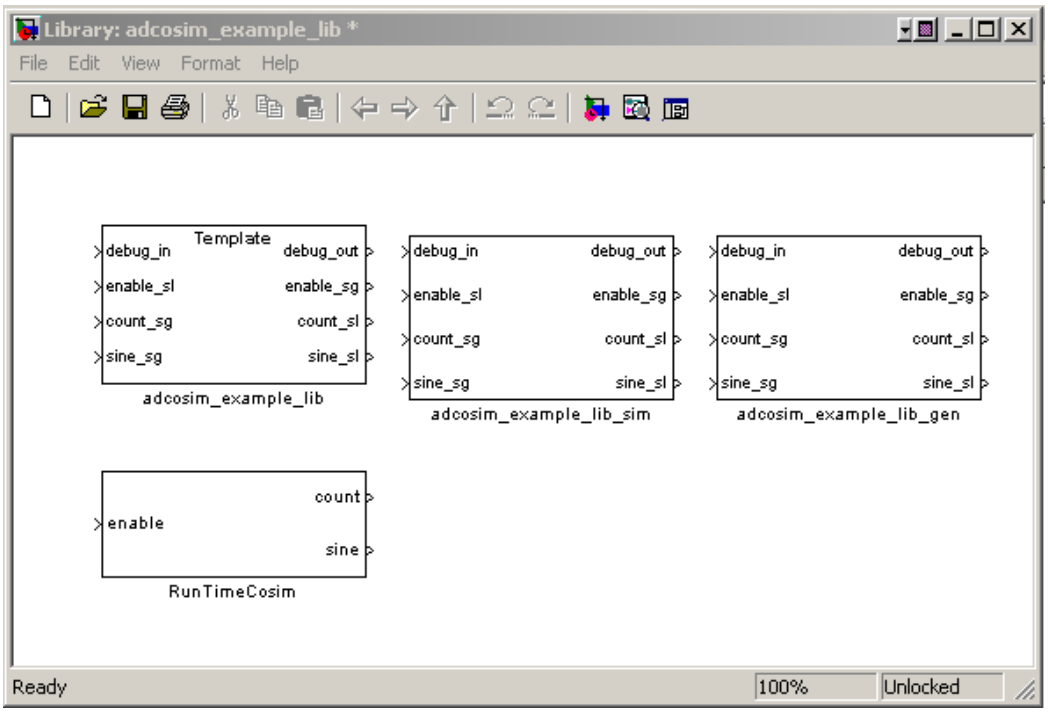
7.1 Cosim Interface Wizard

The first stage in using cosimulation is to build a user cosim library which specifies all the ports. Typing cosim_interface in the Matlab command line brings up the GUI Wizards:

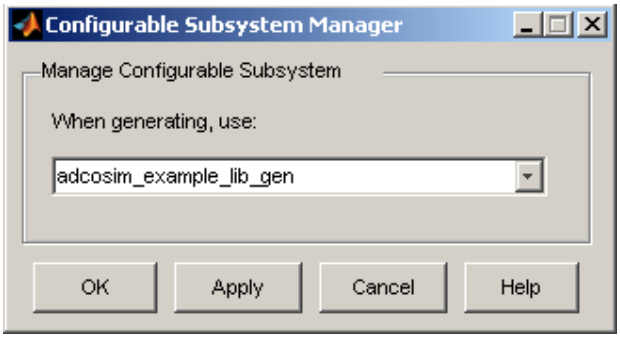


This Wizard allows a list of ports to be specified. A user defined library name should be set. Ports can be added and deleted. The Generate button can then be used to create a library which contains the drop in cosimulation component.

7.2 Cosim Interface Library



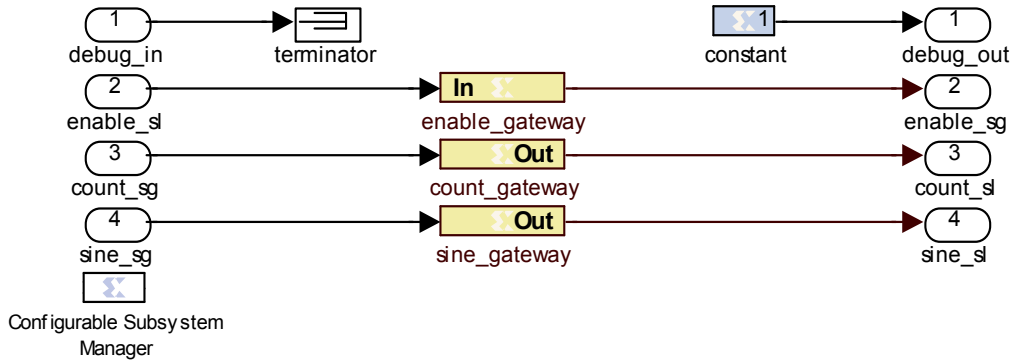
The library contains a Simulation and a Generation version of the port interface. The Xilinx configurable Subsystem Manager will open a dialog, to specify that the generation subsystem should be used: click OK to dismiss this window.



In the design, the "template" block should be dragged and dropped into the design. This is a configurable sub-system, with one subsystem used for Simulation and one subsystem used for the design.

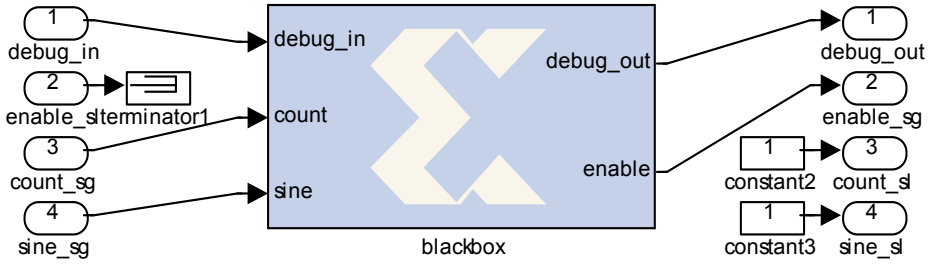
The GUI saves information about the library in 2 Matlab workspace variables: ad_cosim_libname and ad_cosim_ports
These can be saved and reloaded from the Matlab commaned line:
e.g. >> save ad_cosim_data.mat ad_cosim_libname ad_cosim_ports

The simulation subsystem consists of an array of Gateways as specified in the Wizard GUI.



The debug ports are not used in Simulation. Ports ending sl should be connected to the ordinary Simulink signal domain, and ports ending sg should be connected to the System Generator domain.

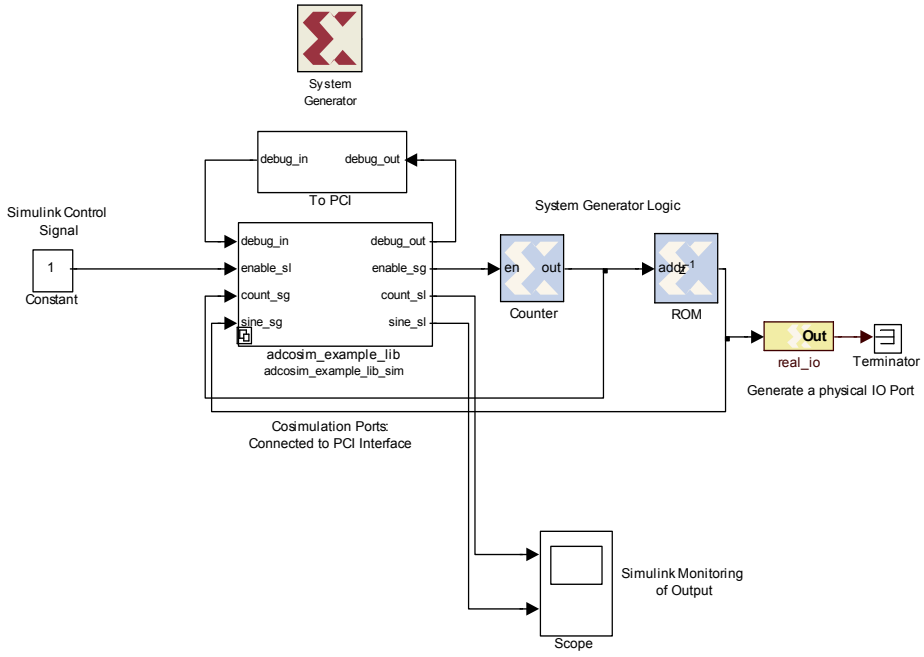
The generation sub-system connects the debug ports (and therefore the PCI bus) to the used defined signals. In generation the sl signals have no function since they are outside of the System Generator scope.



The black box contains an auto-generated VHDL module with the same name as the library. A config.m Matlab function is also auto generated.
This connection of the PCI bus to System Generator signals is similar to the operation of the Xilinx Co-simulation, however it appears internally in the design as a black box, rather than being built externally into the wrapper.

7.3 Cosim Interface Example

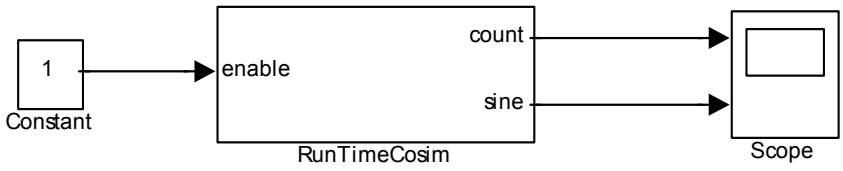
Dragging and dropping the "Template" model into the design will use the Simulation model for simulations and use the co-simulation generation model when generating the bitstream:



The example cosimulation design consists of the cosim example library interface block built by the GUI. Which has an enable port coming into the System Generator design and a count and sine ports coming out of the design. The "To PCI" module is a wrapped up PLX or XPL interface module with all unused ports connected to constant zeros. In this case, one advantage of the drop in cosim interface approach can be seen: the "real_io" Gateway does not turn into a Cosim port, it actually generates a real port in the hardware, and the pins (16 pins on XRM-IO146) can be monitored to see the signal when running the cosimulation.

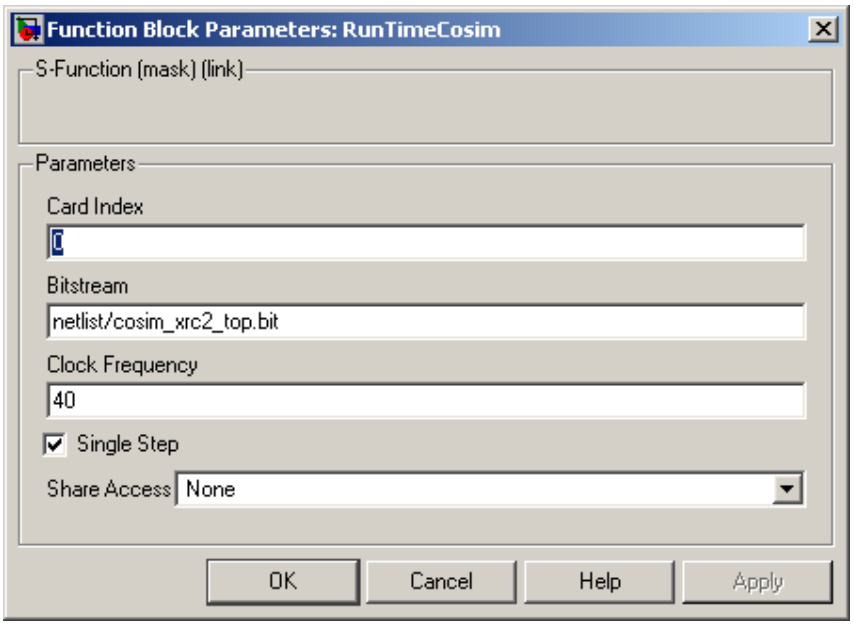
The design can be built using the normal Applications Blockset build option. This will generate the VHDL and UCF files required. The bitstream can then be built using the auto-generation script (ad_build_bitstream) or by using ISE.

7.4 Run Time Cosimulation



The RunTimeCosim block (which was built in the library) can be used to control the hardware. This block has the 3 simulink ports.

The configuration mask for this block has the following parameters:



Card Index selects the FPGA board in the system.

Bitstream selects the generated bitstream

Clock Frequency sets MCLK, which is limited to 100MHz on the XRC2, 160MHz on the XPL,XP and XPI and 200MHz on the XRC4.

The single step option is available, although the following should be noted:
1) The bitstream defaults to free running operation, so if the logic depends on its initial configuration, an input enable or start signal should be used to keep the logic in a know state until the free running clock is disabled.

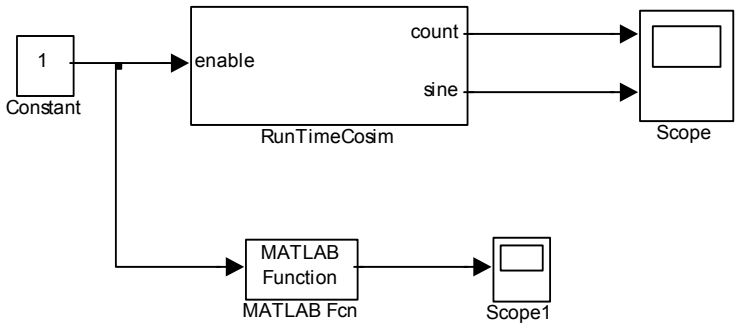
2) The single stepping is achieved through the use of the CLK enable infrastructure in the SysGen VHDL. Any black box VHDL model included in the design needs to respond appropriately to the 'CE' signal.

3) The memory interfaces operate differently in single stepped mode to free running mode. They respond only to inputs when the CE signal is active, however since it is impossible to single step the external hardware, the interfaces continue to free run, and outputs are latched. Consequently in single step mode the qv signal is high and the data valid the clock cycle immediately after a read: in free running mode there is a 3 or 4 clock cycle latency between read and qv. With the SDRAM interfaces, the data transfers between the cache and the SDRAM always complete within a single stepped clock cycle, rather than taking approx. 40 clock cycles.

Shared Access

The Share Access option allows other Simulink Blocks or even other processes to access the FPGA while the simulation is running. In Matlab Share mode, a handle variable 'admxrc_handle<X>' (where <X>=0,1,2… is the card index) is written into the top level Matlab workspace, where it can be accessed by other Matlab functions and Simulink S-functions. When the card is closed at the end of the Simulation, the card handle is set to 0.

The following example has the Share option set to Matlab, and a Matlab function is used to access the FPGA:



The MATLAB function used is

```
function y=read_adregs(x)

if (evalin('base','exist(''admxrc_handle0'')'))
    admxrc_handle0 = evalin('base','admxrc_handle0');
    if (double(admxrc_handle0) >0 )
        y=double(admxrc_read32(admxrc_handle0,hex2dec('40204'),1));
    else
        y=0;
    end
else
    y=0;
end
```

The first clause checks whether the handle exists. The second clause checks that it is valid (not 0). The code then calls the ADMXRC Matlab toolbox function to read Local Bus locations 0x040204. This location is actually the sine co-simulation port, as the cosimulation ports are mapped to addresses 0x040200 to 0x0402FF.
This Matlab script could however perform a much more useful function, such as accessing the Control or Status register of the design. Or writing data over the memory mapped port, emulating the operation of the final control application. Or even using the toolbox DMA functions to write or read data over the ADLOCB into or out of external memory. For more sophisticated interaction with the Local Bus interface M-file or C S-functions can be used.

The third shared option (System) causes Simulink to Open and Close the FPGA board every time it accesses it. This is slower than sharing the handle across the Matlab process, however it is safer and allows any process running on the PC to access the FPGA board between Simulink steps.