



XAPP264 (v1.2) July 2, 2004

## Building OPB Slave Peripherals using System Generator for DSP

Author: Jonathan Ballagh, James Hwang, Phil James-Roxby, Eric Keller, Shay Seng, Brad Taylor

### Summary

The inclusion of embedded processor cores in Xilinx FPGAs opens new doors for high-throughput digital signal processing applications. System Generator for DSP is a high-level modeling environment for designing custom DSP data paths with performance and efficiency comparable to hand-crafted designs. Because System Generator for DSP is tightly integrated with the Simulink® and MATLAB® tools from The Mathworks, Inc., FPGA designs are implemented by users in a familiar setting without being overly concerned with underlying hardware details.

A model can be extended to create a CoreConnect® On-Chip Peripheral Bus (OPB) compatible peripheral using the libraries provided in System Generator for DSP. These peripherals are used in conjunction with the MicroBlaze™ and PowerPC™ processor cores, bringing unprecedented throughput and control to DSP embedded systems designers.

This application note shows how to model a slave OPB peripheral in the System Generator for DSP and how to include the peripheral in an embedded systems platform compatible with the Xilinx Embedded Development Kit (EDK). As an example, simple System Generator for DSP constructs are used to connect a reloadable DA FIR filter to the OPB. An embedded (PowerPC or MicroBlaze) processor is used to control filter coefficient reloading. Primary attention is paid to connecting the DSP data path and the OPB. To illustrate how a processor might be used to exchange data with the DSP peripheral, the steps needed to incorporate the peripheral in a platform consisting of a processor and UART are described. Similar interface logic built using System Generator makes it straightforward to implement far more sophisticated signal processing peripherals.

### Introduction

High-performance DSP data paths modelled in System Generator for DSP (System Generator) can be used as CoreConnect peripherals by extending them with an appropriate interface. The Xilinx BlockSet provides the components necessary to model a DSP peripheral and OPB interface. Although at present (v6.2 release), there are no intrinsic software models for either the PowerPC or MicroBlaze processors, sufficient subsets of PowerPC and MicroBlaze processor functionality, i.e., basic bus transactions, can be modeled within the same environment. This results in a robust simulation and debug environment suitable for DSP embedded systems design. When the software translates the model into hardware, the same vectors used in the Simulink simulation are used as golden test vectors in the hardware test-bench simulation. By ensuring correct peripheral behavior in the Simulink tool, the designer can be confident the peripheral will function correctly in hardware.

This application note discusses the techniques needed to extend a System Generator signal processing data path into a slave peripheral for use on the OPB. These techniques are illustrated using an example platform comprised of a PowerPC or MicroBlaze processor, a UART Lite peripheral for communication with a host PC and a reloadable distributed arithmetic (DA) FIR filter DSP peripheral modeled in System Generator. The principles described in this application note provide a sufficient understanding of the System Generator peripheral modeling process to promote similar techniques for use with other user models. In fact, a

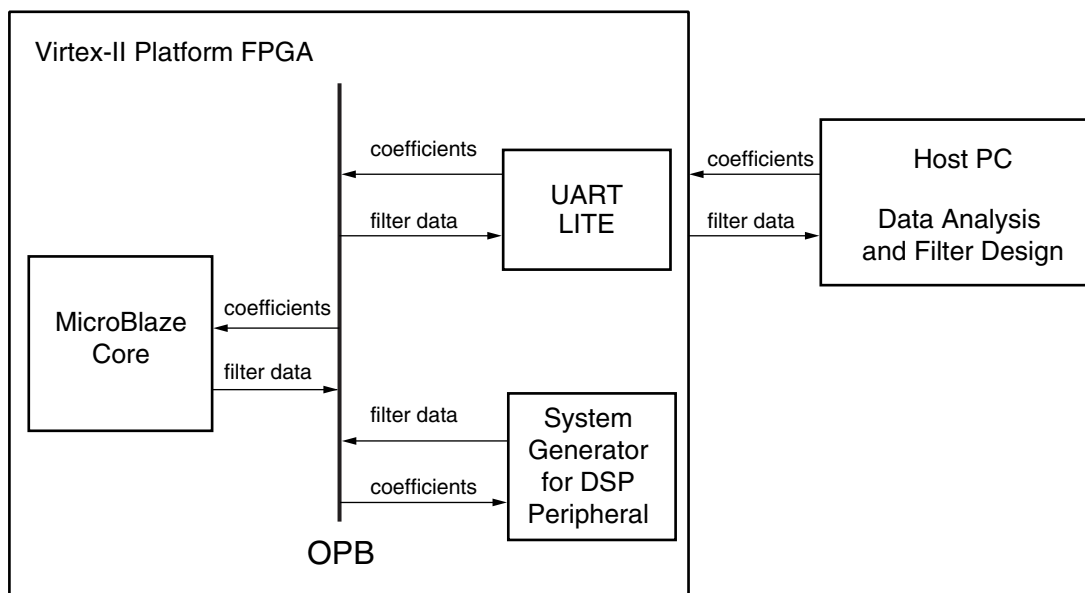
© 2003-2004 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at <http://www.xilinx.com/legal.htm>. MATLAB and Simulink are registered trademarks of The MathWorks. CoreConnect is a registered trademark of IBM. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

significant portion of the bus interface logic used in the example peripheral model is applicable and reusable with other models. This application note assumes the reader is comfortable with System Generator for DSP as well as the Simulink and MATLAB tools. It also assumes the reader has a basic understanding of OPB bus transaction protocols.<sup>[1]</sup>

## Example Platform

The example platform shown in Figure 1 explains how a System Generator model can be extended to become a peripheral. It includes a filter peripheral modeled in System Generator, an embedded processor (either a PowerPC or MicroBlaze processor) for controlling the peripheral, and a UART Lite for bidirectional communication through a serial cable with an external host PC. The primary focus is on the implementation of the peripheral itself.



x264\_01\_112002

Figure 1: FPGA Platform: MicroBlaze, UART, and System Generator DSP Peripheral

The peripheral consists of a System Generator reloadable DA FIR filter augmented with a small amount of control logic. The processor and UART use a serial cable to direct data between the peripheral and a host PC. The PC uses MATLAB to analyze the filter output and design new filters. The PC also initiates filter reloading and transfers new filter coefficients to the processor. Upon receiving new coefficients from the PC, the processor controls the filter reloading from within the FPGA.

The platform operates under two modes: filter reloading and filter frame data transfer. When the filter is not being reloaded, frames of filter output are transferred over the OPB to the processor. From there they are sent to the PC for analysis. On the PC the user can use a MATLAB filter design tool to construct a new filter. After a new filter is constructed, the coefficients are automatically transferred across the serial cable to the UART and then to the embedded processor. Upon receiving the coefficients, the processor transfers the coefficients to the peripheral.

## DSP Data Path

System Generator is ideal for modeling high-performance custom signal processing data paths. In particular, the ease of modeling filtering applications in the software, makes them useful, instructive examples. To extend a data path into an OPB peripheral, an example DSP data path is used. It incorporates a reloadable FIR filter block from the Xilinx BlockSet. Included with the filter is a small amount of control logic to manage coefficient reloading, adjust data rates, and control filter output frame buffering.

A reloadable DA FIR filter block lies at the heart of the data path. The block supports parameterization of coefficient precision, coefficient binary point, number of taps, and filter oversampling rate. The block used in the example datapath is configured with 32 taps, 12-bit coefficient precision, and reloadable coefficients (Figure 2).

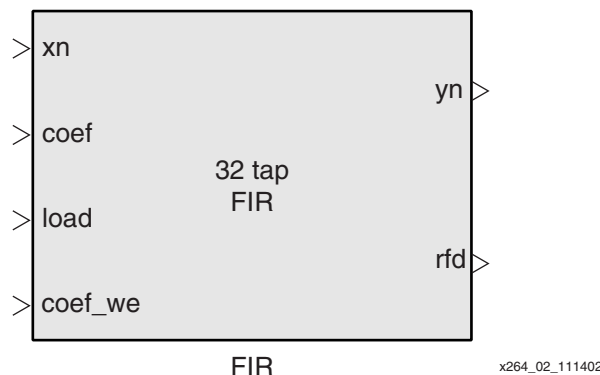


Figure 2: DA FIR Filter Block from the Xilinx DSP BlockSet

Operation of the filter block is straightforward. When the filter is not being reloaded, input values drive the *xn* port and filter output values drive the *yn* port. Filter reloading is initiated with a pulse on the *load* port, *load*. During reload the *rfd* port outputs zeros to indicate the filter is busy. Following the load pulse, new coefficients are written to the *coef* port. Asserting *coef\_we* identifies the current value on the *coef* port as valid. After all coefficients are written, the filter comes back online some number of cycles later and resumes processing data. The block signals when coefficient reloading is complete by reasserting the signal driven by *rfd*. For a detailed description of the block, please refer to the [DA FIR filter data sheet](#)<sup>[2]</sup>.

The filter block is augmented with control logic to allow the data path to communicate with the memory-mapped interface of the peripheral. The data path is implemented in the subsystem shown in Figure 3.

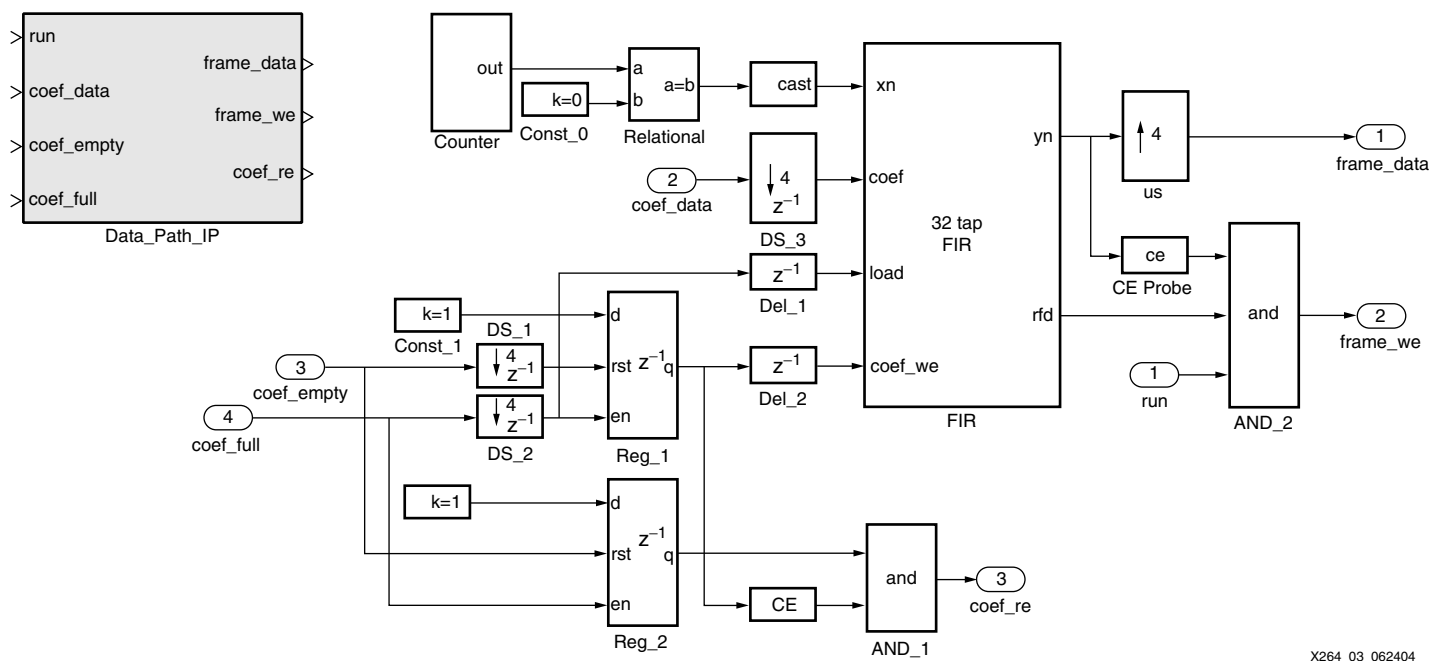


Figure 3: Example DSP Data Path Subsystem

The control logic enables the following in the data path:

- The data path monitors the status of a 1-bit *run* control register in the memory-map interface of the peripheral. The value of this register is driven to the subsystem through the port labeled *run* in Figure 3. When the register is set to "1", contiguous filter output values are written to a FIFO residing in the memory-map interface. The FIFO write-enable signal is driven by the *frame\_we* port of the data path. When the register is set to "0", no values are written to the FIFO. This control register allows the processor to manage data flow from the peripheral to the bus. A full FIFO constitutes one frame of filter output data.
- The data input port of the filter is driven with an impulse train. The impulse train is generated using a counter/comparator pair (blocks "Counter" and "Relational" in Figure 3) to produce a pulse each time the counter rolls over. The maximum count value is chosen to be larger than the number of filter taps. The cast block converts the Boolean (1-bit) output of the relational block into a 12-bit input value driving the data input port of the filter.
- New filter coefficients written to the peripheral by the processor are stored in a second memory-mapped FIFO. It is the responsibility of the data path to monitor the coefficient FIFO signals driven on input ports *coef\_empty* and *coef\_full*. When the FIFO is full, indicating all coefficients have been written, the data path initiates a filter reload sequence and issues read requests to the FIFO to obtain the new coefficients. The coefficient FIFO read request is driven on the *coef\_re* output port.

To conserve hardware, the filter is configured to oversample at a rate of four. The oversampled filter runs at the system rate (i.e., the same rate as the OPB clock), and therefore the filter data rate is four-cycles per sample. To compensate for the rate change, up and down samplers are used (Figure 3 blocks "US", "DS\_1", "DS\_2", and "DS\_3") at places where the data path connects to the bus. Clock enable probes extract the clock enable pulses used in multi-rate designs, and are used to ensure the FIFO Read/Write transactions align to the filter input sample frame.

## Extending the Data Path into a Peripheral

A System Generator data path can be extended into a CoreConnect peripheral through a custom interface constructed using the Xilinx BlockSet. A typical peripheral model requires the following components in addition to the DSP data path.

- Interface to the OPB signals
- Address decoding logic
- A memory-mapped register interface to the I/O ports of the data path
- Logic to manage bus transaction handshaking

To make the peripheral as modular as possible, each of the above components are encapsulated in their own Simulink subsystem. Using subsystems also allows each component to be designed and debugged individually, and then to be added to a library for future reuse. A general case System Generator peripheral consisting of subsystems is shown in Figure 4. The following sections focus on the implementation of each subsystem with an example using the reloadable DA FIR filter data path.

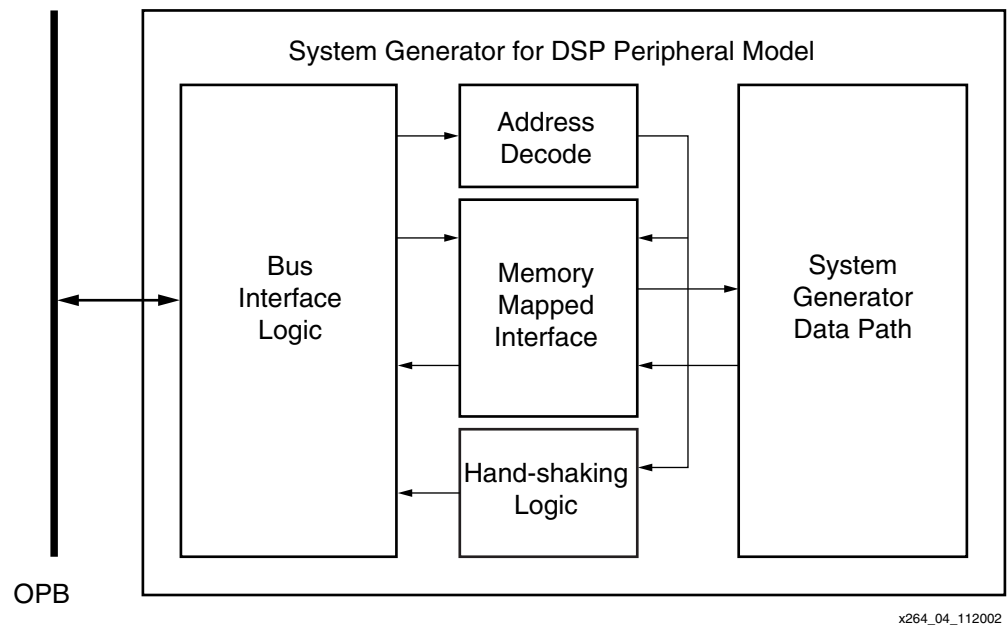


Figure 4: A Peripheral Modeled in System Generator for DSP

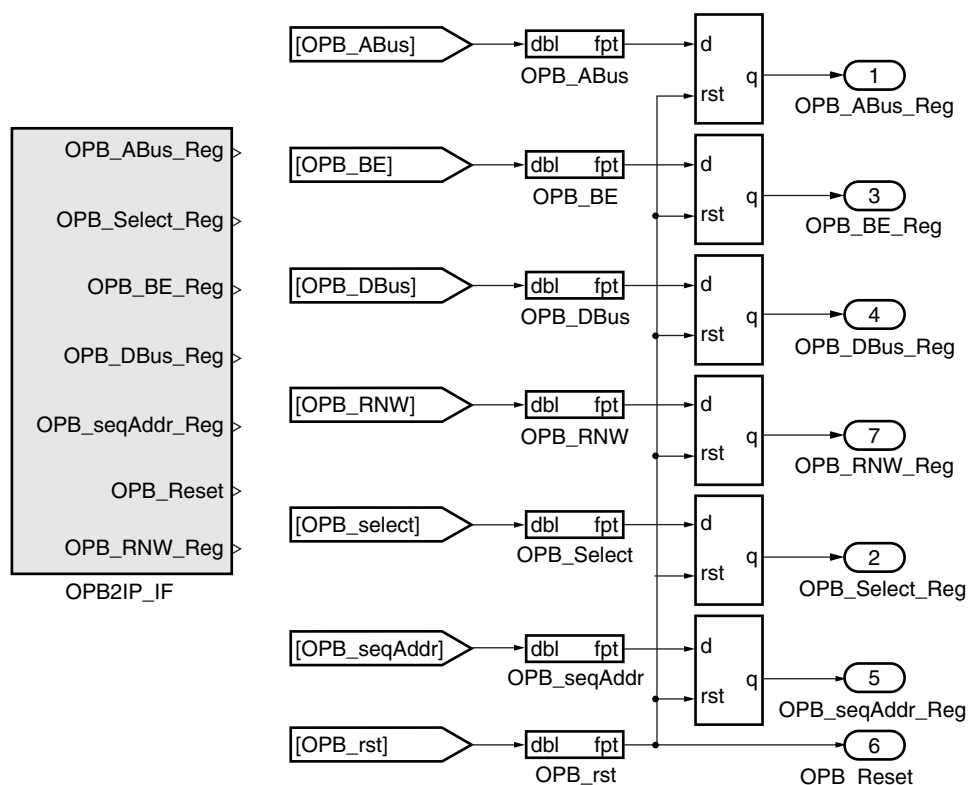
## Bus Interfacing

Bus interface logic is needed to bridge the gap between the I/O ports on the peripheral and the OPB. A Simulink subsystem is a natural parking place for this logic. The benefits of encapsulating the bus interface logic in a subsystem are two-fold. First, placing this logic in a subsystem results in a convenient abstraction of the OPB. Users can easily tap-off signals as needed from the bus interface subsystem. Second, coupling Xilinx input gateway and output gateway with the subsystem logic ensures the necessary ports are instantiated on the top-level peripheral VHDL when the model is translated into hardware.

The names given to the gateway blocks reflect the corresponding OPB signal names. Following this guideline allows the designer to easily identify and associate OPB signals in the microprocessor peripheral description (MPD)<sup>[3]</sup> file with the corresponding top-level ports on the VHDL model description.

Separating the bus interface logic into two subsystems, one for signals driven to the peripheral by the OPB, and one for signals driven by the peripheral to the OPB, results in a more natural depiction of the left-to-right data flow within the model. This is done in the example by implementing the interface logic in two separate subsystems, *OPB2IP\_IF* and *IP2OPB\_IF*. *OPB2IP\_IF* contains the interfacing of signals driven by the OPB to the peripheral. *IP2OPB\_IF* connects signals driven by the peripheral to the OPB. Wherever possible, register these signals to improve timing.

The interface and subsystem logic for the *OPB2IP\_IF* component is shown in Figure 5. A best practice design registers the signals read from the OPB. These registers can be removed if the peripheral timing constraints can be relaxed. Each register in the subsystem has an explicit reset port exposed. Routing the *OPB\_rst* signal to the reset port of each register ensures the contents of these registers are reset to an initial value if the OPB reset is asserted.

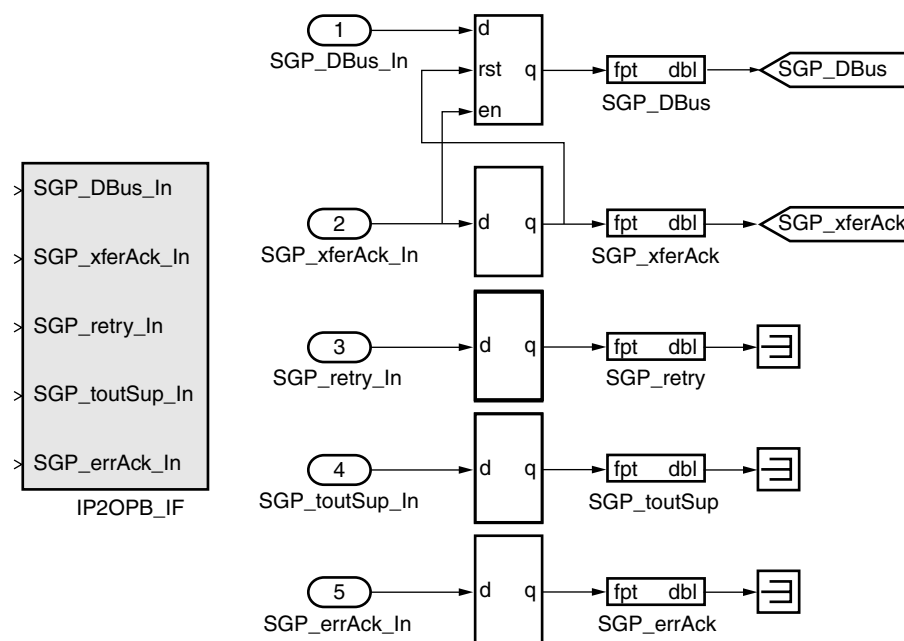


x264\_05\_040903

Figure 5: OPB2IP\_IF Subsystem

Of note is the use of global from blocks as sources for the input gateways. Global from blocks allow gateway blocks to be driven without needing explicit ports on the subsystem interface. This has advantages for simulation, as shown when processor code is encapsulated into a separate processor model subsystem. During simulation, the processor subsystem drives these from blocks using global goto blocks.

The *IP2OPB\_IF* subsystem is shown in Figure 6. All output signals are registered before being written to the OPB. Again, these register blocks can be removed if peripheral timing is relaxed. Global goto blocks follow the output gateways and allow the processor subsystem to monitor the output signals of the peripheral without explicit wiring. The reset port on the *SGP\_DBus* register is driven by the registered acknowledge signal. This wiring ensures the peripheral data output register resets to zero on the cycle immediately following the assertion of the acknowledge signal. This satisfies the requirement to have the peripheral drive zeros to the OPB when the acknowledge is Low. The terminated OPB signals are not used in this example.



x264\_06\_112002

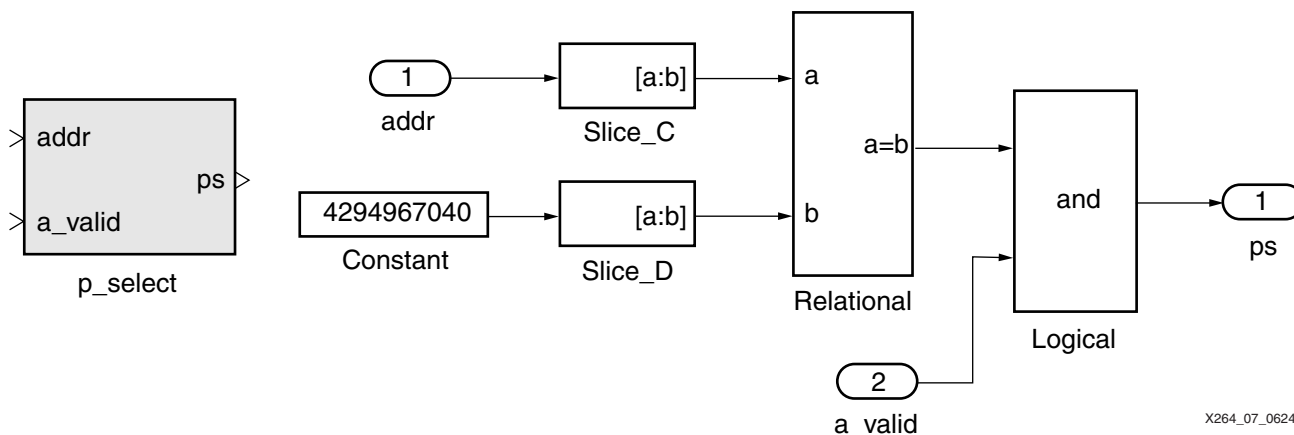
Figure 6: IP2OPB\_IF Subsystem

The *IP2OPB\_IF* and *OPB2IP\_IF* subsystems are placed in the top level of the peripheral hierarchy. Every gateway is named after a corresponding OPB port and is assigned a matching width.

## Address Decoding

When a processor (or other OPB master) attempts to read or write to a peripheral, it writes an address to the bus. It is the responsibility of the peripheral to decode the address and decide if the current address value is within the memory-mapped allocation space of the peripheral. The OPB master indicates a valid address value by asserting the *OPB\_select* signal. The peripheral needs only to decode the current address when *OPB\_select* is high.

In this example, the address decoding subsystem is implemented with behavior matching the *p\_select.vhd* <sup>[4]</sup> component distributed with the Xilinx EDK. For reuse, the subsystem is made as generic as possible. The subsystem and subsystem logic are shown in Figure 7. The *p\_select* subsystem has two input ports, *addr* and *a\_valid*. Port *addr* is driven by the bus address signal, *OPB\_ABus*. The *a\_valid* port of the subsystem is driven by the *OPB\_select* signal from the bus. The *ps* output port drives the peripheral select signal for the model.

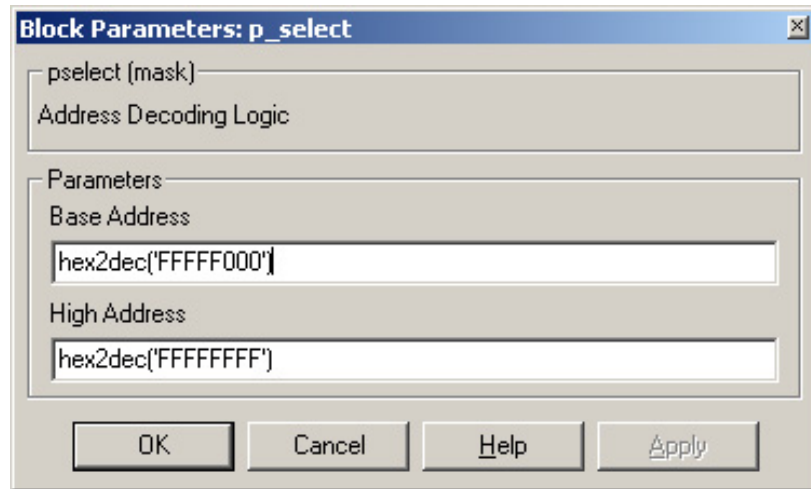


X264\_07\_062404

Figure 7: *p\_select* Address Decoding Subsystem

Two slice blocks, *Slice\_C* and *Slice\_D* (Figure 7) extract the relevant bits of the address signal. The slice blocks are configured with a range defined as an offset from the MSB. The constant block stores the entire base address of the peripheral. The relational block tests for equality between the outputs of the two slice blocks. Finally, a logical block configured to perform an *AND* operation ensures that the peripheral select output *ps* is only asserted when the address is valid, as indicated by the *a\_valid* signal. Note, the *p\_select* subsystem implementation assumes the memory-map allocation range is an even power of two.

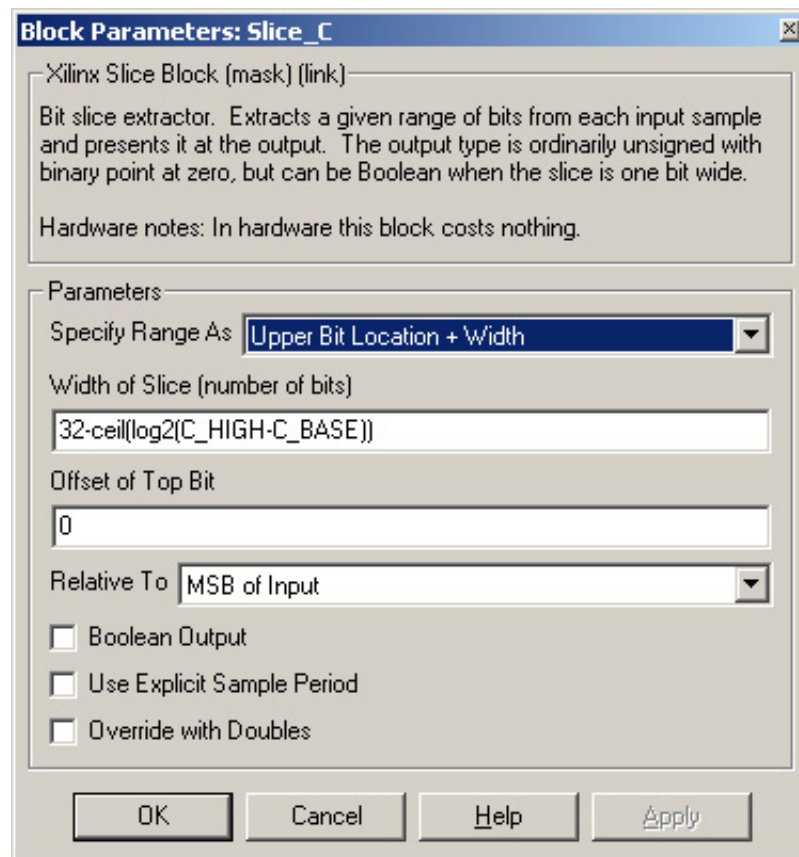
The usefulness of the *p\_select* subsystem is further extended by converting it into a masked subsystem. The subsystem is parameterized in terms of the desired base and high address values for the peripheral model (Figure 8). The base and high address values are passed to mask parameters *C\_BASE* and *C\_HIGH*, respectively. The *p\_select* block should be placed be at the top level of the Simulink model to pass the mapped address to the OPB Export Tool.



x264\_08\_111402

Figure 8: Mask Parameterization GUI for the *p\_select* Subsystem

The close integration of System Generator with MATLAB allows blocks to be parameterized using MATLAB expressions. This flexibility allows the constant and slice blocks to be parameterized using the *C\_BASE* and *C\_HIGH* parameters. The slice blocks are identically parameterized and the corresponding mask GUI is shown in Figure 9.



X264\_09\_062204

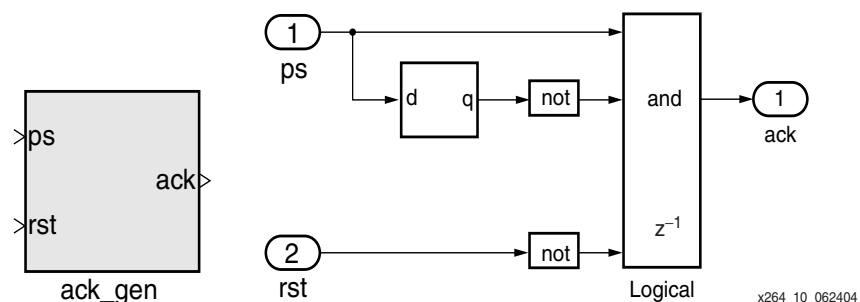
Figure 9: Mask Parameterization GUI for the *Slice\_C* Block

## Generating the Acknowledge

A slave peripheral either read or written to on the OPB must generate an acknowledge pulse once it has completed the transaction. This acknowledge must be accompanied by valid peripheral output data during a read. This pulse is driven to the *SGP\_xferACK* signal of the OPB.

In the example peripheral, each read and write has a fixed and equal latency. Although a state machine is an equally valid alternative, this example uses a registered AND gate to produce the pulse. This is the technique used in the tutorial "Designing Custom OPB Slave Peripherals for MicroBlaze"<sup>[4]</sup> to generate the acknowledge. The input to the registered AND gate is driven by the peripheral select with extra logic to ensure the register output is asserted for one cycle, thereby producing a pulse when the peripheral select is asserted High.

The AND gate is registered to ensure the acknowledge pulse is correctly aligned with the peripheral output data. Figure 10 shows the logic needed to generate the acknowledge pulse.



x264\_10\_062404

Figure 10: Acknowledgement Generation Subsystem

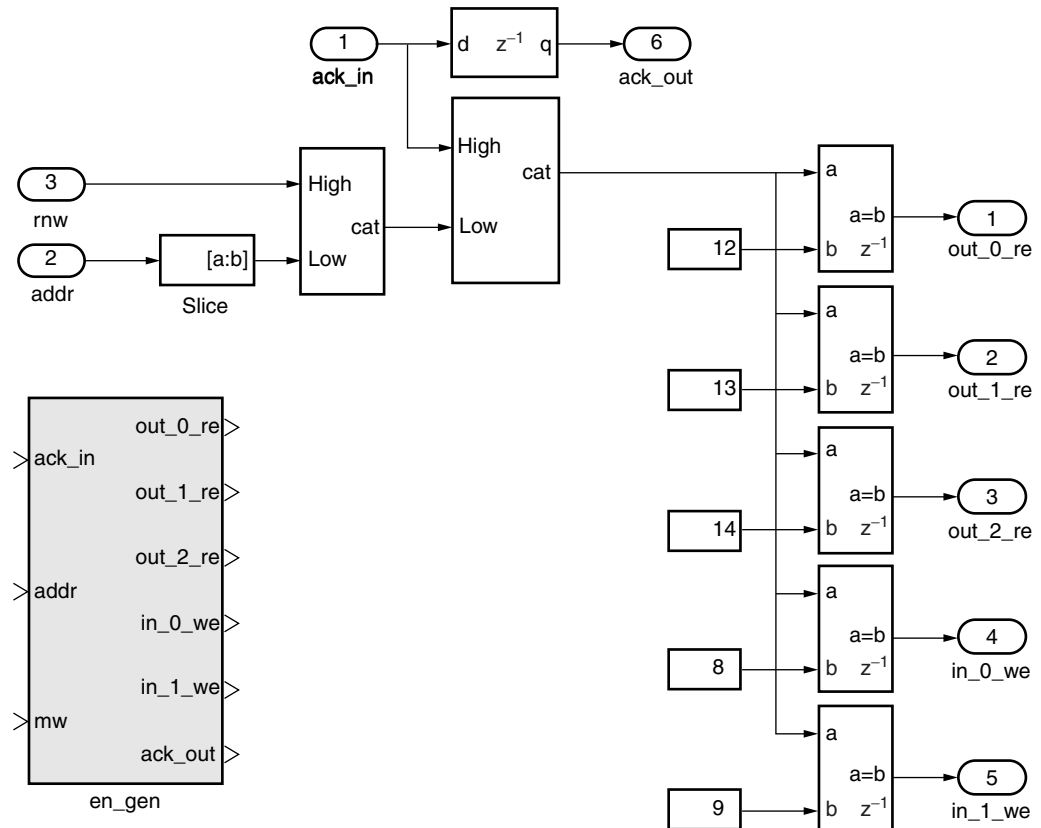
### Defining a Memory-Mapped I/O Interface

The peripheral communication interface with the OPB is defined in this section. This is typically realized through a memory-mapped I/O interface where each port on the example data path is assigned an offset from the base address of the peripheral. There are five I/O ports of interest in the design's data path. The ports are assigned the offset values shown in [Table 1](#).

**Table 1: I/O Mapping for the DA FIR Filter Peripheral**

Signal	Description	Transfer	Offset
out_0_re	Data	Read	0 x 0
out_1_re	Buffer Full	Read	0 x 4
out_2_re	Buffer Empty	Read	0 x 8
in_0_we	Run/Stop	Write	0 x 0
in_1_we	Coefficient	Write	0 x 4

Additional decoding logic is included in the peripheral to generate the enable signals for each memory-mapped register/FIFO component. The subsystem used to generate these enable signals is shown in [Figure 11](#). A slice block extracts the relevant bits from the address signal. In the example peripheral, the addresses are aligned to the full 32-bit word boundaries. Therefore, ignore the two least significant bits of the address signal as they are not needed for data steering. The *OPB\_BE* signal is not used in the model. The *OPB\_rnw* signal and peripheral acknowledge signals are concatenated with the extracted address bits. The resulting signal drives the first input port of a series of comparators. A constant block drives the second input port of each comparator. The constant value is derived using the offset value of the memory-mapped element along with the read/write status, and assumes an asserted acknowledge. The enable signals can now be wired to the enable ports of their respective memory-mapped components.



X264\_11\_062404

**Figure 11: Enable Generation Subsystem**

Having assigned peripheral I/O ports to addresses, a memory-mapped interface is constructed using the Xilinx BlockSet. The memory-map interface is partitioned into two subsystems, one for the peripheral inputs and the other for the peripheral outputs. The peripheral input memory map is considered first.

A standard memory-mapped input interface is comprised of register and FIFO blocks; both are naturally modeled and available in the Xilinx BlockSet. The input memory map for the example peripheral is implemented using a register block for the *run* control register and a FIFO block for filter coefficient buffering. Both the register and FIFO data inputs are driven by the OPB data input signal. Slice blocks are placed on both data input signals before the block inputs. For the *run* control, only a single bit for the control is required. This eliminates the need to use a 32-bit register to store the bus data. Instead, the slice block extracts the LSB from the data bus and generates a Boolean output signal. Likewise, the slice block for the FIFO extracts the 12 bits needed to store each filter coefficient. However, the precision produced by the slice block is incompatible with the filter coefficient precision required by the DA FIR filter block. The slice block generates an unsigned 12-bit number with zero fractional bits. The parameterization of the DA FIR filter block requires signed 12-bit values with 11 fractional bits. To make this conversion, a force block is placed immediately after the *Slice\_B* block. The force block does not require additional hardware resources and is only used to allow Simulink to correctly interpret and scale the coefficient data value.

Many System Generator blocks provide explicit enable and reset controls. These ports are mapped to the enable and reset ports in the synchronous hardware elements when the model is translated into hardware. These ports are used in the example memory map to control when the register and FIFO blocks are written to. An explicit enable signal is exposed on the register block and is driven by its respective *we* signal *in\_0\_we* from the address decoding logic. Similarly, the *we* port of the FIFO block is driven by the corresponding *we* signal, *in\_1\_we*. The corresponding input memory-map subsystem is shown in Figure 12.

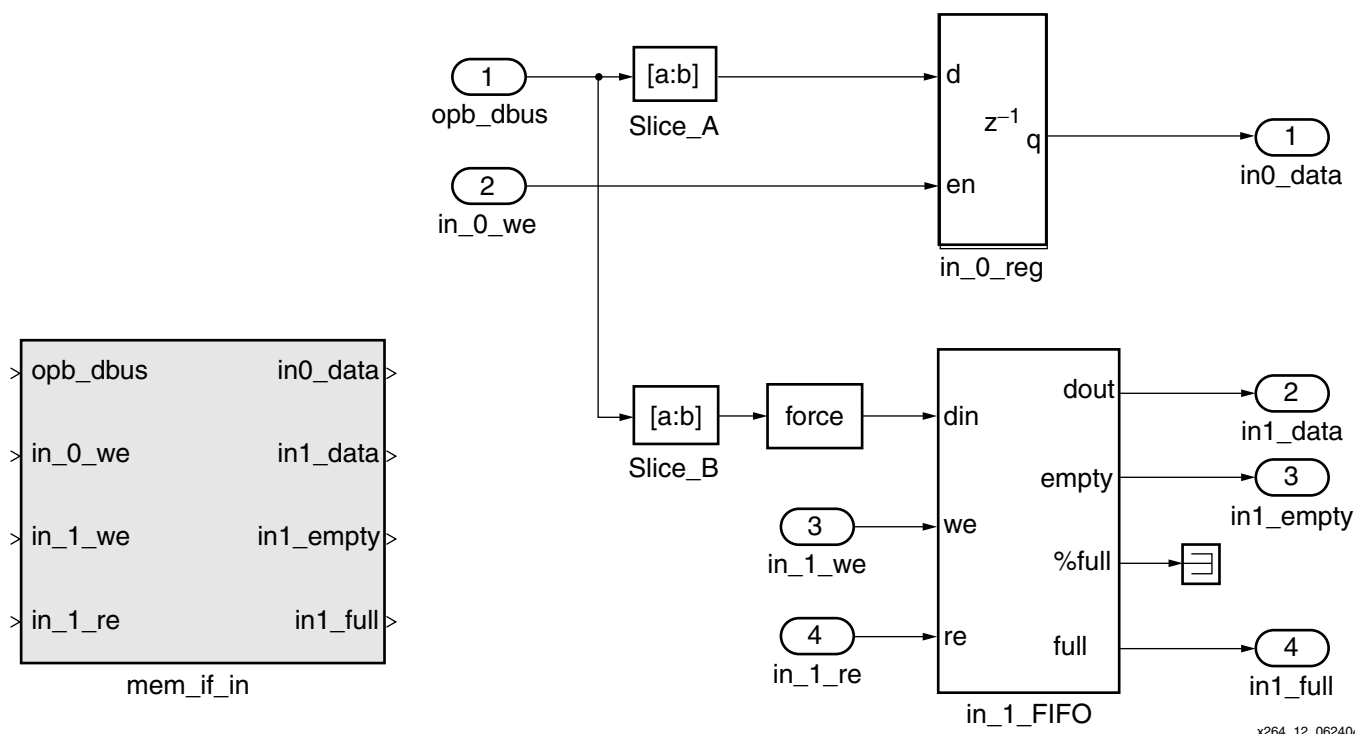


Figure 12: Example Peripheral Input Memory Map

The output memory-map interface is comprised of a FIFO with multiplexing logic to switch between FIFO output signals. As shown in Table 1, three outputs from the FIFO: data, full, and empty are the predominant concern. In addition, the output memory map must drive zeros to

the OPB data output signal if the master is not attempting to read from one of these three signals.

When a read request is issued to the peripheral and the address corresponds to one of these three signals, valid data must be driven to the bus data signal. The bus must have the valid data driven to it in the same cycle as when the acknowledge signal is asserted. The peripheral drives zeros at all other times to avoid bus contention.

The output memory-map subsystem and logic are shown in Figure 13. A MUX block configured with four inputs is used to switch between constant zeros and the FIFO outputs. The FIFO outputs are all different widths, however, this is compensated for by using cast blocks to convert the output widths to 32 bits. The input to the subsystem are the three output read enable signals. These signals are concatenated together and drive the input of a ROM block. The ROM block is parameterized to decode the signal and drive the MUX select line with an appropriate value. If none of the read-enable signals are asserted, the MUX selects the constant 32-bit zero input.

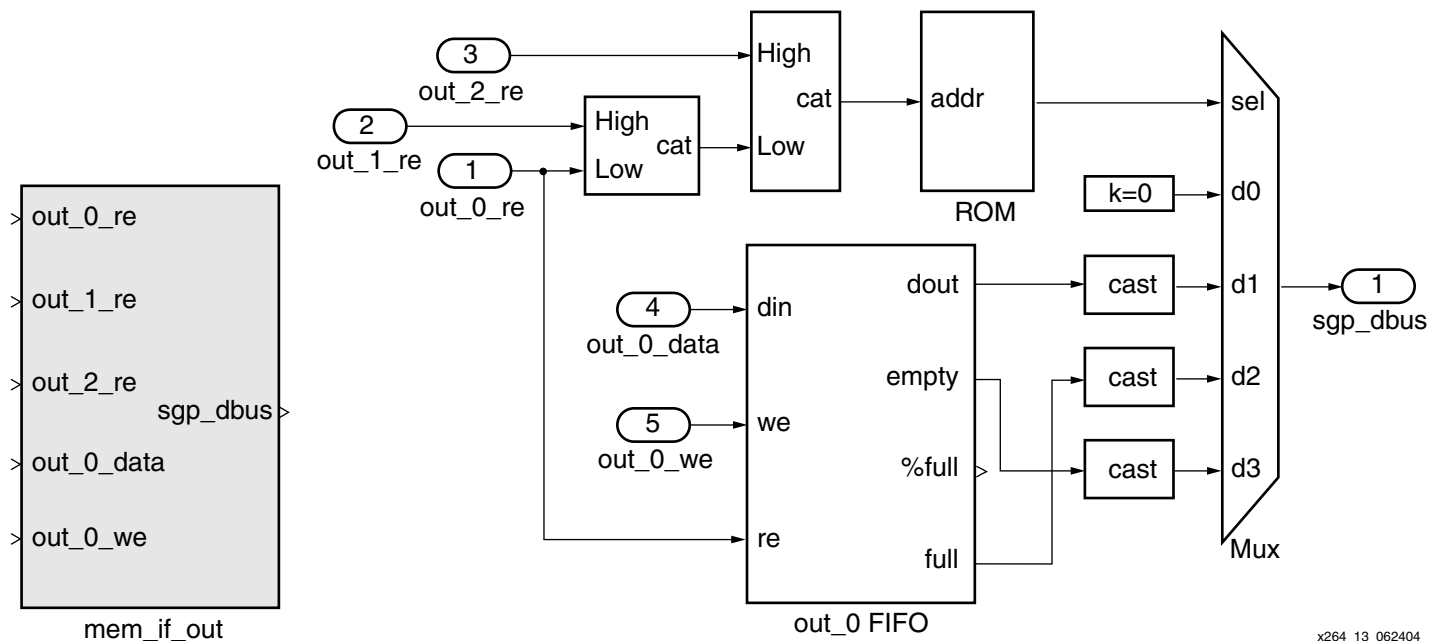


Figure 13: Example Peripheral Output Memory Map

x264\_13\_062404

## Simulating the Peripheral

The complete peripheral implementation is shown in Figure 14.

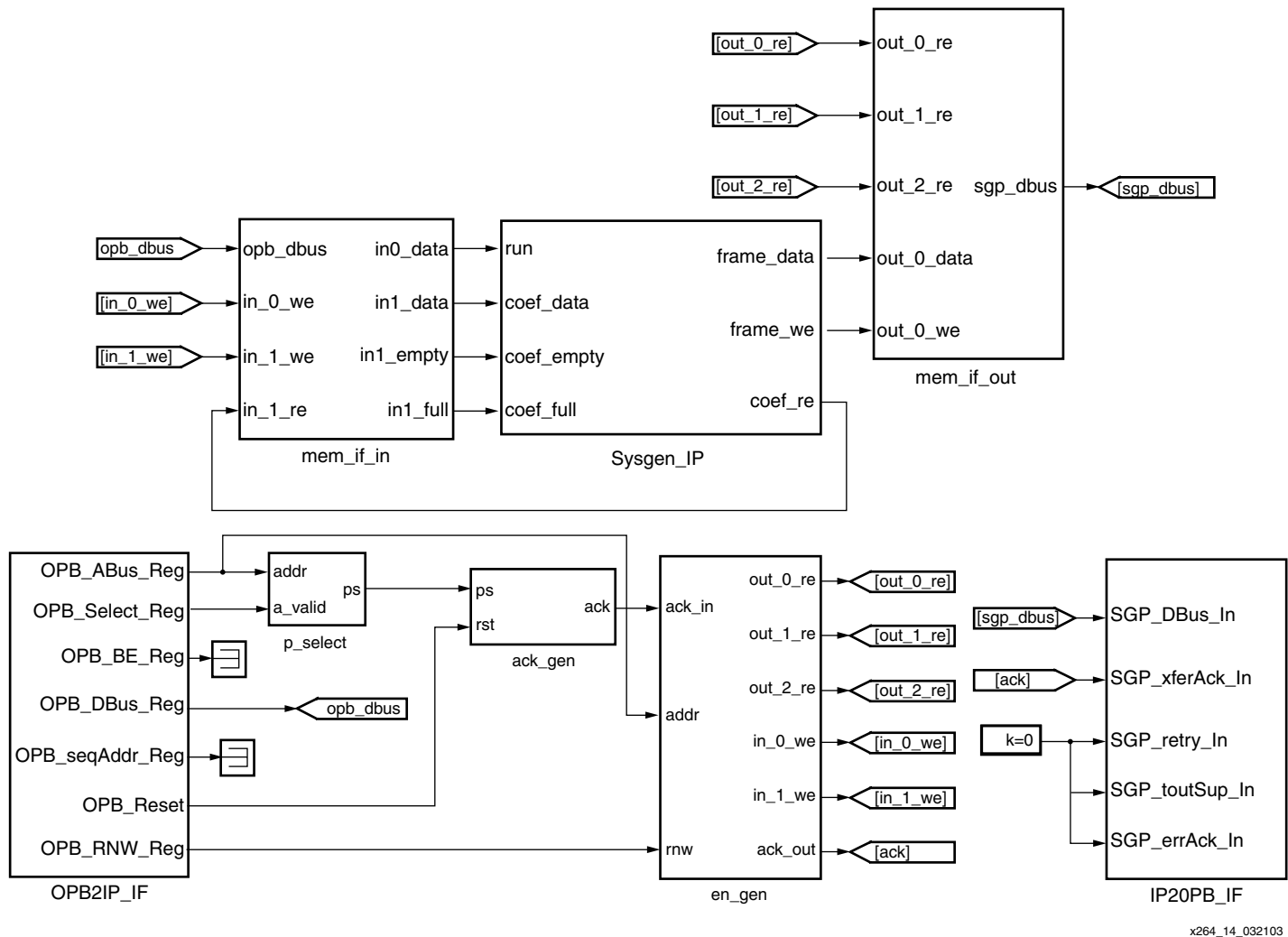


Figure 14: Example Peripheral Implementation

Simulink offers a variety of tools for simulating and debugging the peripheral model. These tools can be used by coupling the peripheral model with a bus stimuli model. StateFlow™, an event-driven interactive modeling and simulation tool from MathWorks, can be used as a tool to model basic processor behavior. By simulating subsets of the processor code using state transition diagrams, the user can better visualize peripheral model behavior under realistic stimuli. Simulating subsets of the processor code in Simulink is also advantageous as most analysis tools from existing Simulink libraries can be used in the peripheral debugging process. When the model is translated to hardware, System Generator automatically produces a test bench using the bus simulation test vectors as golden test vectors in the hardware simulation. By running these tests, the hardware representation is both bit and cycle accurate when compared to the behavior of the model.

For the example peripheral, a StateFlow diagram is implemented to model a MicroBlaze code stub. Each StateFlow diagram output drives a corresponding OPB input port of the input bus interface subsystem of the peripheral. Similarly, every StateFlow input is driven by an OPB output port from the output bus interface subsystem of the peripheral. Abstract connections to the bus interface subsystems are realized by the input and outputs of the StateFlow diagram driving or reading global from or goto blocks, respectively. This approach allows encapsulation of the StateFlow diagram, source, and syncs into a single subsystem. Because the input and

outputs are wired to global from/goto blocks, bus signals can be tapped off accordingly with additional from/goto blocks. The model monitors the bus signals driven by the peripheral via global from blocks.

The tags on the global from/goto blocks match the from/goto tags found in bus interface blocks, *OPB2IP\_IF* and *IP2OPB\_IF*. The processor model in Figure 15 accepts a trigger condition; where the triggering is on the rising edge of the clock. A clock probe block extracts the system clock and drives the trigger port of the StateFlow diagram. The resulting StateFlow model with sample state transitions modeling a processor code stub are shown in Figure 15.

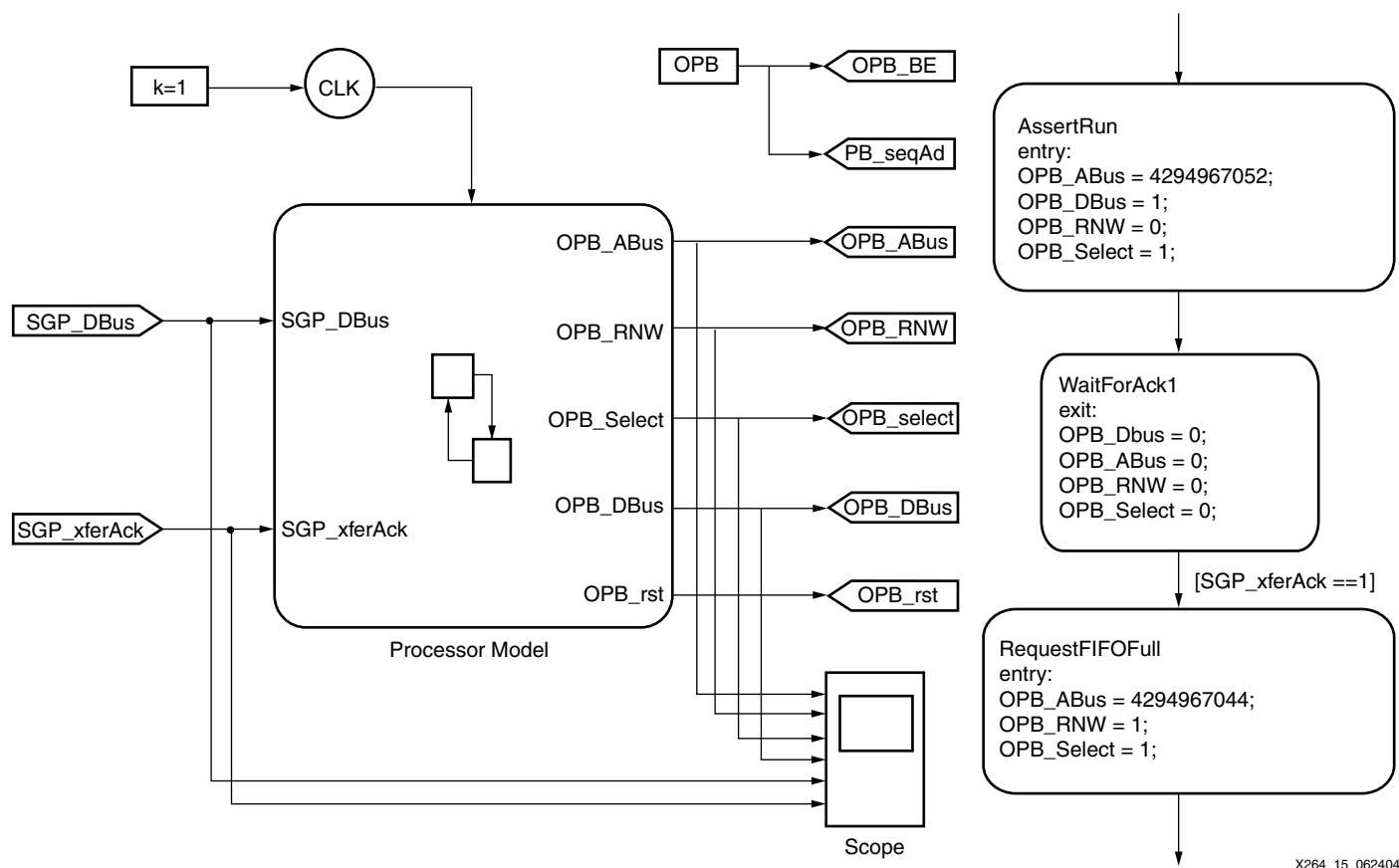


Figure 15: StateFlow Diagram Block with Example State Transitions

Included with each state transition in the diagram is a set of signal assignments producing a corresponding bus transaction (Figure 15). Using StateFlow allows easy, reproducible behavior of a processor code stub. In this case, the stub is focused solely on testing the functionality of the DSP peripheral, and not on the other components in the platform. The model is used only during simulation and is not translated in the hardware implementation.

## The OPB PPC Demo and Test Programs

The OPB filter peripheral is designed to be included in an FPGA system containing a PowerPC 405 or MicroBlaze microprocessor and an OPB. The XAPP264 reference design includes the following:

- An example system that utilizes the peripheral in a PPC configuration
- C code to run on the PPC to illustrate how to access the OPB filter
- A C/Matlab executable running on the host PC that demonstrates how to exercise the filter from C and Matlab environments.
- A ChipScope™ block to demonstrate real-time debugging of the OPB filter.

- The OPB Export Tool, a System Generator plug-in, which is used to generate an EDK peripheral from the System Generator Simulink model
- A more complex System Generator model called OPB\_2D\_FILTER. This model implements a reloadable 16 x 16 pixel 2D video rate filter<sup>(1)</sup>. The filter can be attached to a real-time video stream or used by a PPC or MicroBlaze processor as an accelerator. The design runs at 2X the OPB clock rate and demonstrates how to use an asynchronous video clock.

## Installing the OPB Export Tool in System Generator

An optional compilation target is available for System Generator v6.2 that generates the files necessary to import the System Generator design as a peripheral in the Xilinx Embedded Development Kit (EDK). The OPB Export Tool plug-in file, `opb_export_tool.zip`, is included with the XAPP264 reference design files. To install it, save the plug-in file to a temporary directory, and then type `xlInstallPlugin('opb_export_tool.zip')` from the Matlab command window. Alternately the file can be unzipped and moved to the `plugins` directory at:

```
$MATLAB\toolbox\xilinx\sysgen\plugins\compilation\OPB Export Tool
```

where `$MATLAB` indicates the current Matlab install directory.

After installing the OPB Export Tool, re-open the Simulink model and click on the System Generator dialog box. Select the `OPB Export tool` for compilation.

## Generating the EDK Peripheral

It is useful to begin with the EDK project into which the System Generator peripheral is imported. If an EDK project does not currently exist, a new system may be created using the EDK Base System Builder tool included in the Xilinx Platform Studio (XPS). An example base system that supports the System Generator peripheral may be created using the following options in Base System Builder:

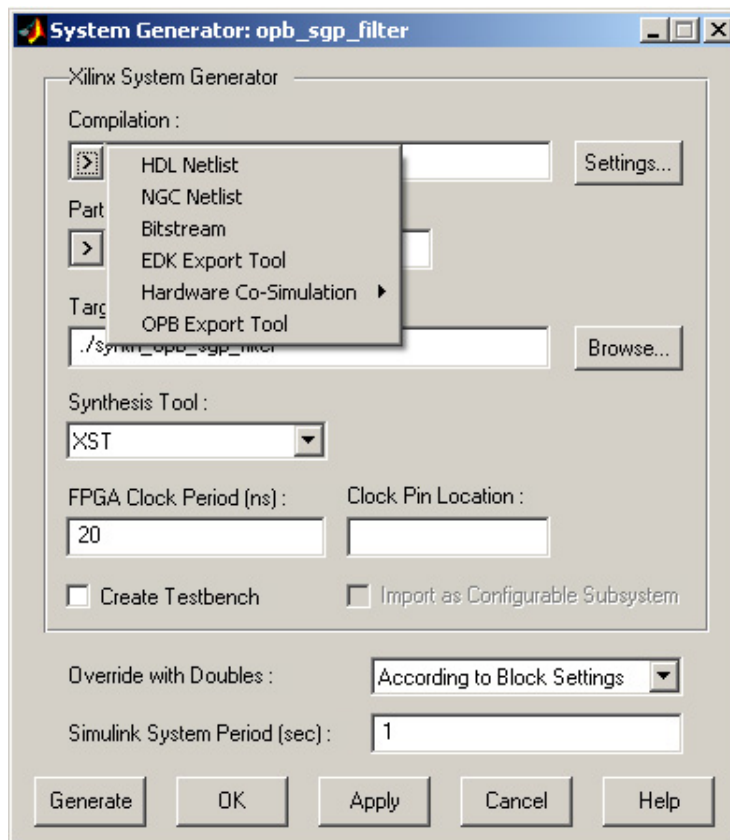
- MicroBlaze or PowerPC option
- Bus clock frequency at 50 MHz
- At least 8 KB of instruction RAM and 8 KB data RAM
- RS232 peripheral as 'OPB UARTE' option (set the baud rate to 57600)

Once the EDK system is properly constructed, System Generator can implement your design as a peripheral that may be imported in this system.

To generate an OPB peripheral from the filter design described earlier, open the filter model named `sg_opb_sgp_perhipheral.mdl`, which is located in the `\xapp264_ref_design_v1.2\sysgen_model\` directory. From the Compilation menu on the System Generator dialog box, select the `OPB Export Tool` option (see [Figure 16](#)).

---

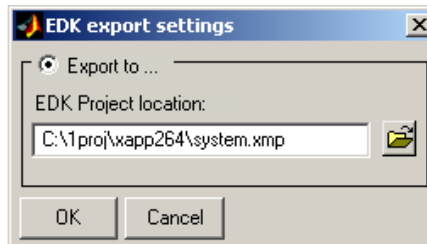
1. The authors wish to acknowledge the contributions of Catalin Baetoni in the design of this filter.



X264\_16\_062204

Figure 16: OPB Export Tool

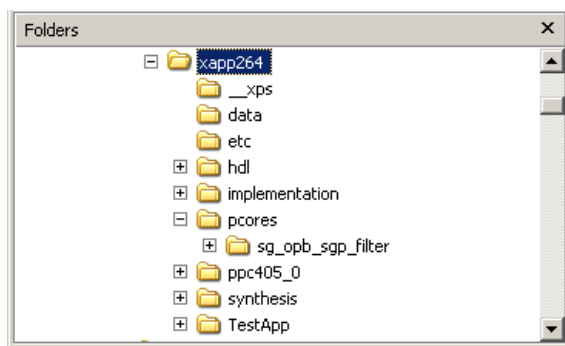
To create the EDK peripheral, version 6.2i of the ChipScope tool must be installed. To avoid creating an unnecessary testbench, uncheck the `Create Testbench` checkbox in the dialog box. Select the `Settings` button (located next to the `Compilation` menu) to specify the EDK project directory that contains the Xilinx Microprocessor Project file (XMP) for the system in which you are including the peripheral (see Figure 17).



X264\_17\_062204

Figure 17: EDK Project Directory

Select the `Generate` button to create an OPB peripheral. After generation, a peripheral named `sg_opb_sgp_peripheral` is created in the target directory and optionally copied into the `pcores` directory of the EDK project (see Figure 18).

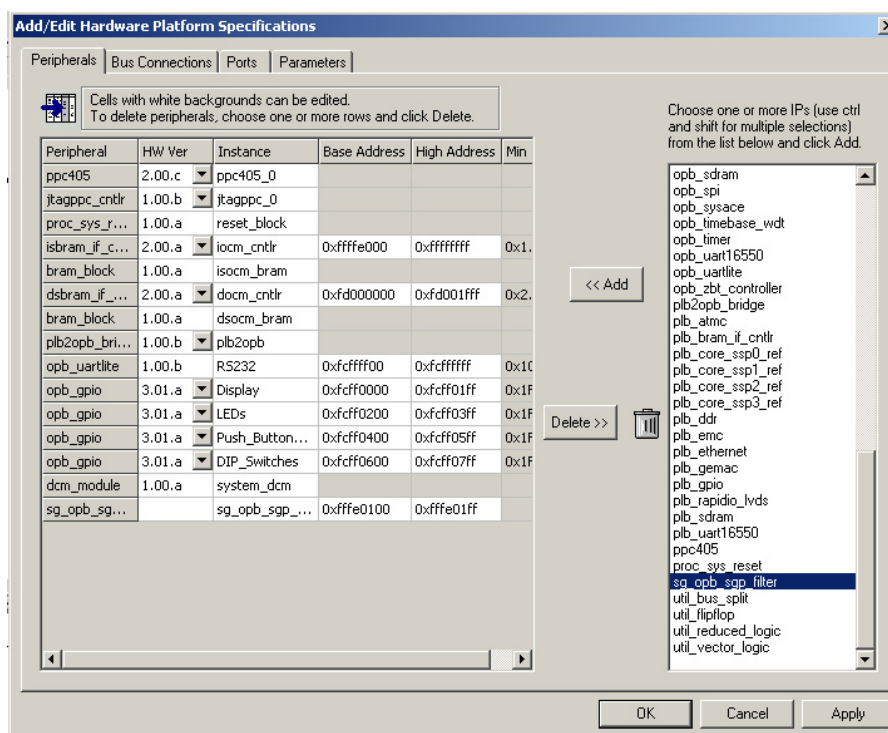


X264\_18\_061704

Figure 18: Target Directory

## Including the New Peripheral into a EDK Project

If the XPS is open, close and re-open the EDK project to which you are adding the peripheral. This procedure allows XPS to detect the new directory under `pcores` containing the files for your System Generator peripheral. Select **Add/Edit cores** from the Project menu in XPS. Select the **Peripherals** tab and add the `sg_opb_sgp_filter` peripheral to the project (see Figure 19).



X264\_19\_061704

Figure 19: Peripherals Tab

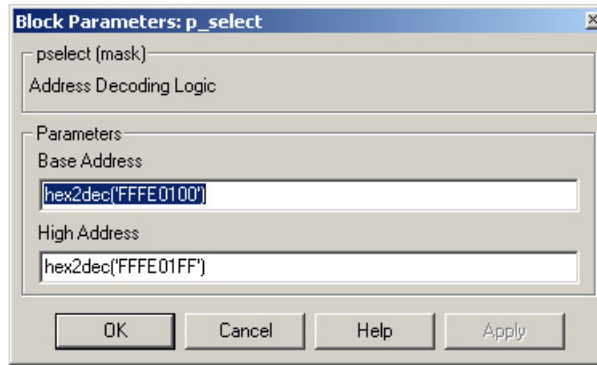
The OPB Export Tool for System Generator v6.2 does not support a relocatable address space. Therefore you must ensure that there are no address conflicts.

The `sg_opb_sgp_peripheral` appears in the left window. Note the address space allocated to the `sg_opb_sgp_peripheral` and make sure it does not conflict with another peripheral's address space. If there is a conflict, select an unused address and enter it into the dialog box of the `p_select` block in `opb_sg_filter.mdl`. Alternately, you can remap the conflicting

EDK peripheral to an address space that does not conflict with the `sg_opb_sgp_filter` peripheral.

## Assigning a New Address Space to the OPB Peripheral

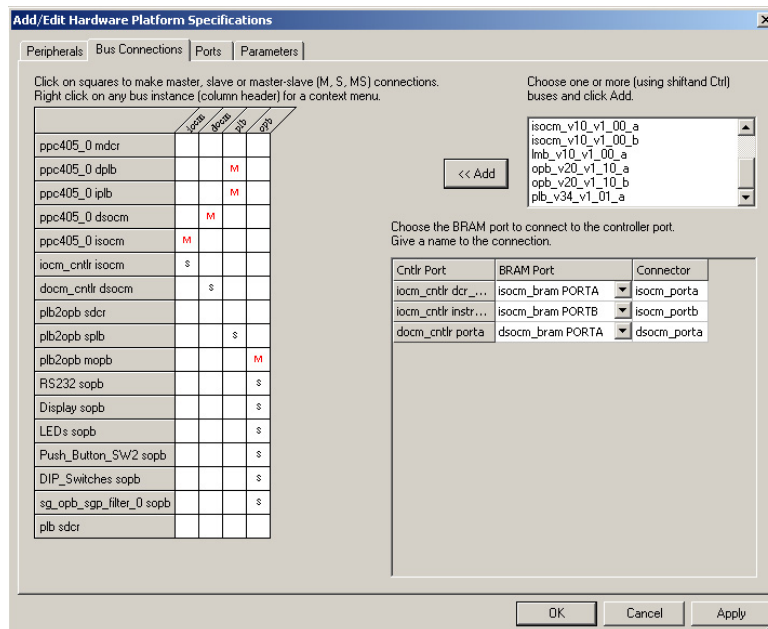
Open the Simulink `opb_sg_filter` model included with the XAPP264 reference design files. Open the parameters dialog box for the `p_select` block and enter the new address space as shown in Figure 20. Note that the OPB Export Tool requires that the `p_select` block be at the top level of the Simulink model.



X264\_20\_062404

Figure 20: Address Space Entry

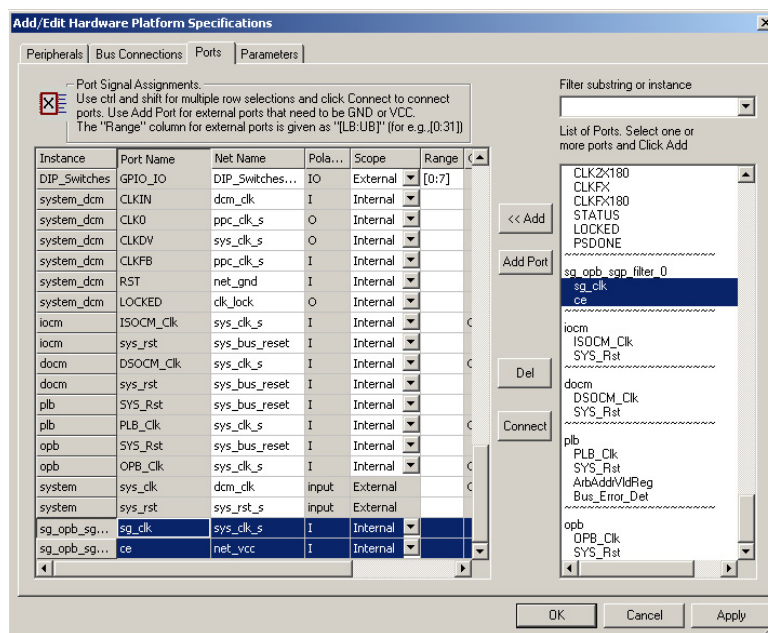
Use System Generator to regenerate the filter peripheral, close the EDK project, and then re-open it.



X264\_21\_061704

Figure 21: Bus Connections Tab

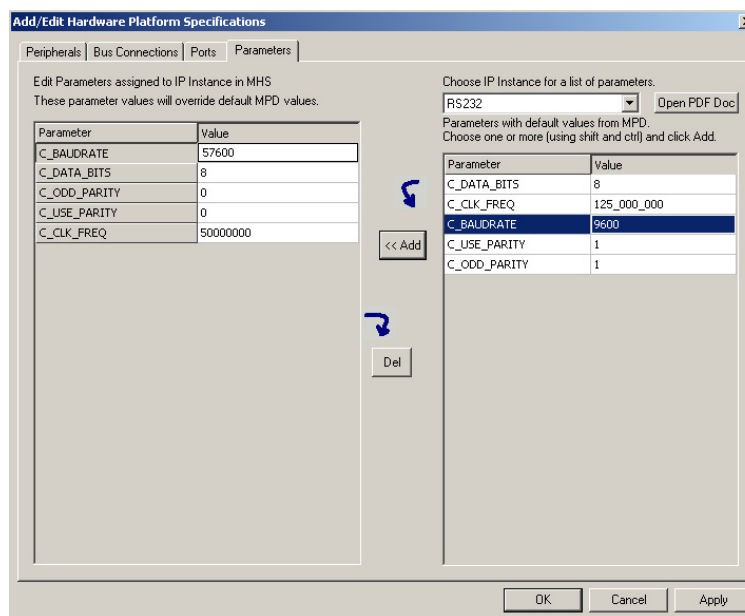
In the `Bus Connections` tab (see Figure 21), connect `sg_opb_sgp_filter_0` to the OPB by clicking on the square at the intersection of the `opb` column with the `sg_opb_sgp_filter_0` row. An 's' appears indicating that the filter's OPB ports are now connected to the OPB.



X264\_22\_061704

Figure 22: Ports Tab

In the Ports tab (see Figure 22), connect the filter peripheral's remaining ports. Select the `sg_opb_sgp_filter_0 sg_clk` and `ce` ports in the right List of Ports window, and click << Add. After the filter ports appear in the left window, change the scope for the two ports to Internal. Next, edit the Net Name fields for `sg_clk` to `sys_clk_s` and `ce` to `net_vcc`.



X264\_23\_061704

Figure 23: Parameters Tab

Finally select the Parameters tab (see Figure 23) and change the RS232 baud rate to 57600.

After configuring the EDK system, the project can be built for a target FPGA by selecting the Generate Bitstream option from the XPS Tools menu. If the address space of the peripheral is changed, it might be necessary to delete the old peripheral before generating a

new bitstream. From the Tools menu in XPS, select Clean->All, then select Generate Bitstream.

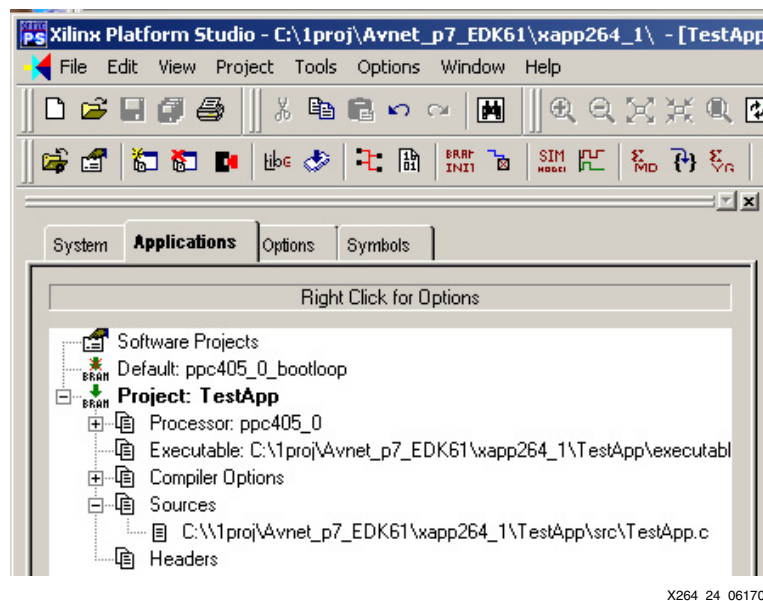
## Adding the PPC Source Code and Compiling the PPC Program

A new EDK project contains a default C program called `TestApp.c`. On boot, this program sends the following messages to the serial port and toggles any LEDs that are supported on the test board:

```
-- Entering main() --
-- Exiting main() --
```

You may test this application code using your FPGA test board. To do so, attach a serial cable from the FPGA test board to COM1 on the host PC and open a Microsoft HyperTerminal window (found in the Start/Accessories/Communications/ folder). Configure the HyperTerminal window to use COM1 with a baud rate of 57600, 8 bits of data, no parity, and one stop bit. To set the UARTs Flow Control setting to None, select File>Properties>Configure under Flow Control.

Download the bit file to the board by selecting Tools->Download and ensure the FPGA program is configured with the appropriate text being displayed in the HyperTerminal window.



X264\_24\_061704

Figure 24: Applications Tab

To include a custom C program in your EDK system, right-click on Sources in the Applications tab (see Figure 24), and select Add. Then select the C files and header files to include in the project.

## Running the OPB Filter Demos

The OPB filter project contains two C files that can be added to the test application. To run the demo, remove `TestApp.c` from the project and copy the `sysgen_peripheral_demo.c` and `sysgen_peripheral_monitor.c` files to the `TestApp/src` directory. Add their files to the project and recompile the project. Download the resulting design to the test board.

## Running the Monitor Demo

After you resolve any address space issues and include the example C source files in the test system, download the bitstream to the test board. The `sysgen_peripheral_monitor.c` file

contains a very simple monitor program. When the board boots, you should see the following text in the HyperTerminal window:

```
>-----
      DA Filter test
-----
Help
-----
<?> print this menu
<c> exit monitor and jump to filter I/O loop
<l> load filter with coef based on i*data
<i> display filter impulse
<[,<]> select reg
<+,<-> select data
<w> write data to reg
<r> read from addr
```

The monitor program allows single reads and writes of registers mapped into the System Generator filter peripheral. To test the filter, set the data value to 1 using the '+' and '-' keys. Load the filter coefficients using the 'l' key, which loads the coefficients as an array incrementing by the 'data' value. The data input of the filter is connected to an impulse generator, and the filter output therefore reflects the value of the loaded coefficients. To view the filter output, type 'i'. To exit the monitor program, type 'c'. Upon exit, the filter monitor jumps to an I/O loop contained in `sysgen_peripheral_monitor.c`. This I/O loop is used to send filter data to a Matlab program.

For this design, the registers are mapped according to [Table 2](#).

**Table 2: Address Map**

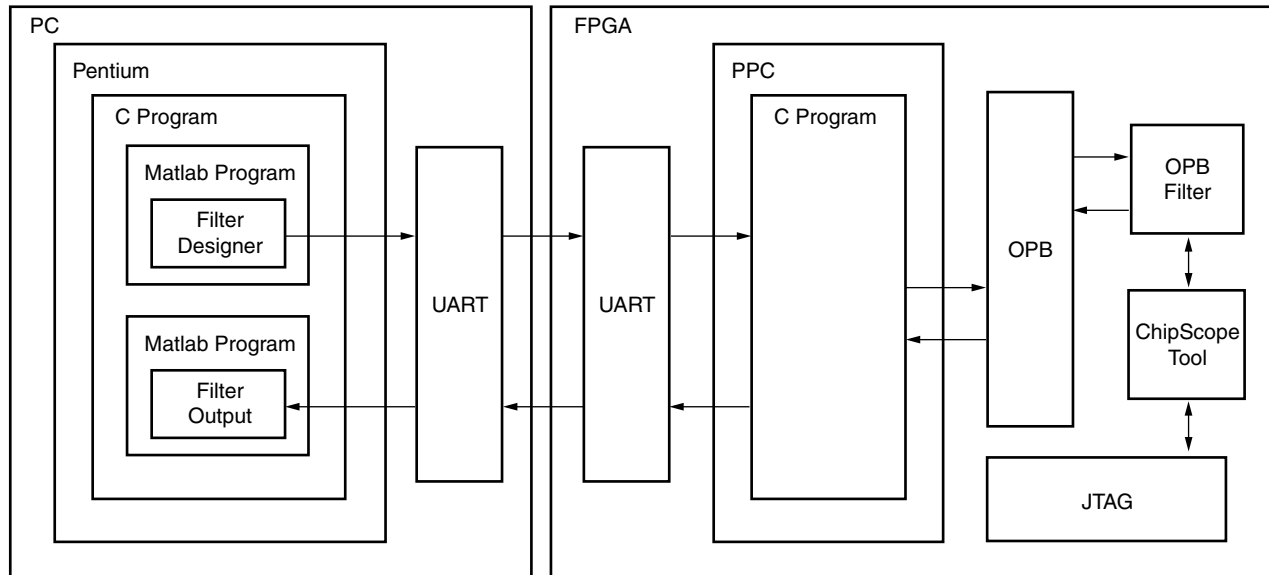
reg	addr	mode	Description
dout	0	Read-only	Read output of filter
full	4	Read-only	Indicates filter FIFO is full
empty	8	Read-only	Indicates filter FIFO is empty
run	0	Write-only	Write to control filter operation
coef	4	Write-only	Write to load filter coefficients

Note that because the writable registers are write-only, you cannot read back the contents of a register after writing it.

By design, the filter is continuously driven by an impulse. It is tested by reading back the filter output to return the filter coefficients. Since the impulse phase is not configurable, the resulting filter coefficients exhibit a random alignment when read out.

## The OPB Filter Program

The structure of the complete demo system is illustrated in Figure 25.

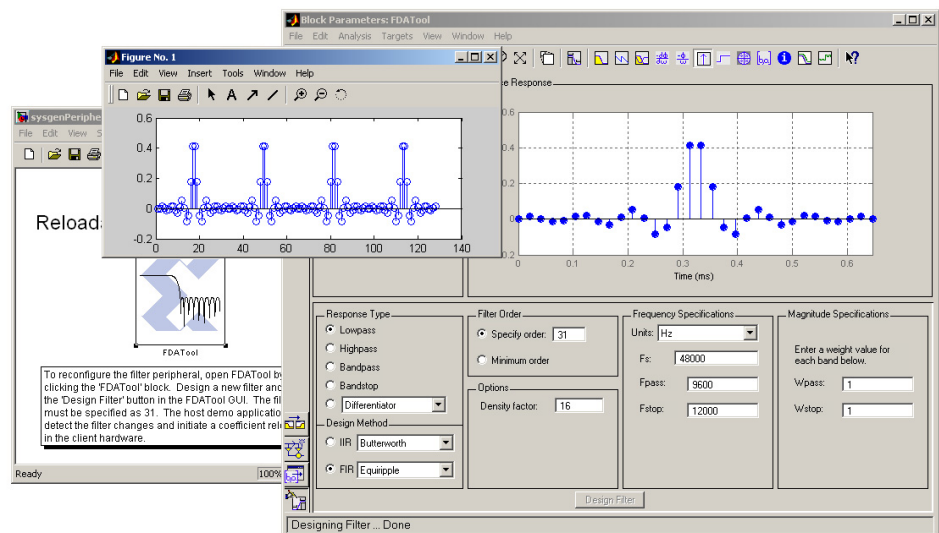


X264\_25\_061704

Figure 25: OPB PPC Demo System

## Running the Matlab Filter Demo

A program called `sysgenPeripheralDemo.exe` is included with the XAPP264 reference design files. This program uses the serial port to talk with the PPC program running the I/O loop program. Running this program from the command window launches Matlab's Simulink-based filter designer GUI. After you design a new filter, the coefficients are sent over the serial port to the PPC and then to the OPB filter via the `filter_io()` C program loop. The impulse response is then returned back to the C program and displayed in a Simulink display window (see Figure 26). To run the demo, type `sysgenPeripheralDemo.exe` and make certain that HyperTerminal is closed to allow the C program to connect to the serial port. To quit the demo, type 'q' in the command window.



X264\_26\_061704

Figure 26: Simulink Display Window

A Microsoft Visual C project directory is included if you desire to rebuild or alter the demo. To rebuild the demo, the C compiler Include and library paths must be extended to include the \$MATLAB\extern\include and \$MATLAB\extern\lib directories. \$MATLAB is an environment variable that points to the base directory of your MATLAB software tree (for example, c:\MATLABR13).

## Using the ChipScope Tool to Debug the OPB Peripheral

The ChipScope tool from Xilinx is very useful for debugging FPGA-based microprocessor peripherals. The ChipScope block included with System Generator v6.2 automatically inserts the ChipScope circuitry in the design when it is compiled (the ChipScope block is included in the lower-right corner of the design). Any signals intended to be visible to the ChipScope tool are connected to the ChipScope block. Clicking on the ChipScope block opens a dialog box that allows you to select the number of signals to make visible (see Figure 27).

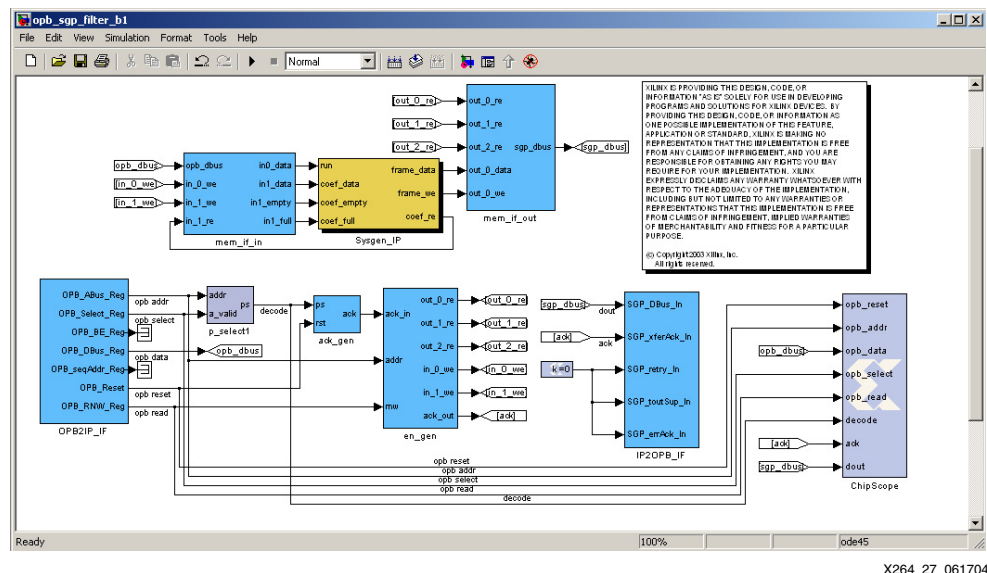


Figure 27: ChipScope Dialog Box

To use the ChipScope tool to debug the OPB filter:

- Connect the host PC to the FPGA test board using a serial cable
- Open HyperTerminal using COM1 with a baud rate set to 57600
- Make sure a suitable Xilinx programming cable (e.g., Parallel Cable IV) is connected to the FPGA test board via the JTAG port
- Download the bitstream to the test board
- Launch the ChipScope tool

When the ChipScope GUI comes up, select Get JTAG Cable Information and open the device (FPGA) containing the OPB filter and PPC.

To import System Generator signal names into the ChipScope signal display, select Import from the File menu. Navigate to the target directory used to generate the OPB filter (\xapp264\_v2\xapp264\_ref\_design\_v2\_0\sysgen\_model\synth\_opb\_sgp\_filter) and select the opb\_sgp\_filter\_chipscope.cdc file. The signal names should now show up in the left ChipScope window. You can then drag the signals of interest into the display window.

Set the ChipScope tool to trigger on the decode signal when it is a logic 1.

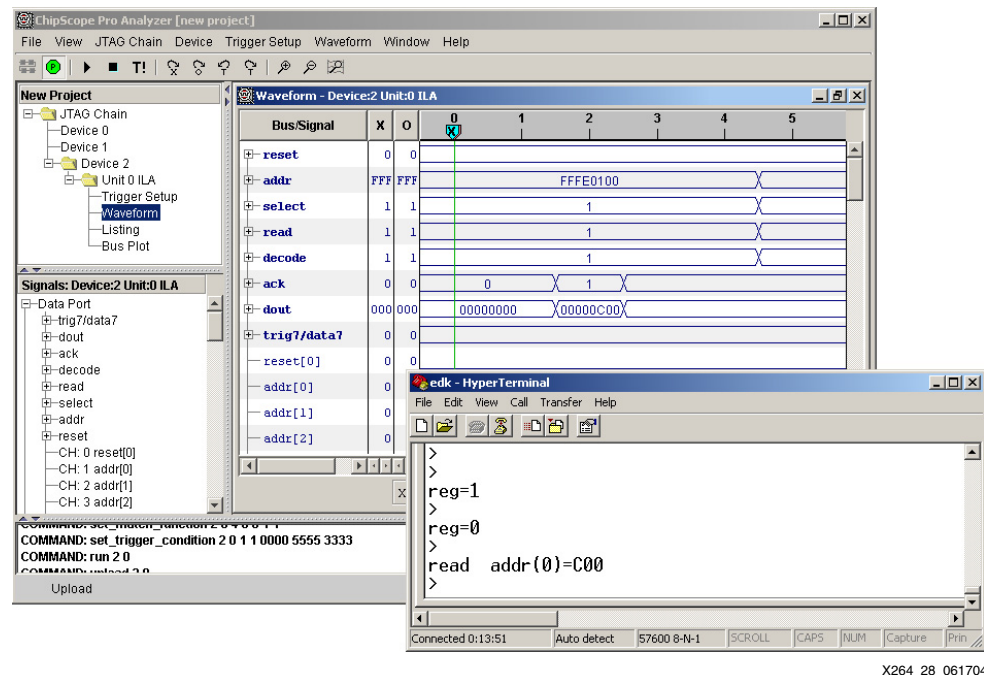


Figure 28: OPB Read Cycle Result

Now reboot the FPGA to enter the monitor program. Using the HyperTerminal window and keyboard, select an address and data value. Then execute a read or write cycle to the OPB filter, which triggers the ChipScope display, enabling the inspection of internal OPB signals. The example screen capture in Figure 28 shows an OPB read cycle returning 0x0C00 on both the HyperTerminal screen and the ChipScope screen.

## Reference Design

The reference design files are available on the Xilinx FTP site in both VHDL and Verilog.  
<http://www.xilinx.com/bvdocs/appnotes/xapp264.zip>

## References

1. IBM, Inc. On-Chip Peripheral Bus: Architecture Specifications Version 2.1,  
<http://www-3.ibm.com/chips/techlib/techlib.nsf/techdocs/9A7AFA74DAD200D087256AB30005F0C8>
2. Xilinx, Inc., Distributed Arithmetic FIR Filter V8.0, Product Specification.  
[http://www.xilinx.com/ipcenter/catalog/logicore/docs/da\\_fir.pdf](http://www.xilinx.com/ipcenter/catalog/logicore/docs/da_fir.pdf)
3. Xilinx, Inc., Embedded System Tools Guide: Embedded Development Kit, EDK (v6.2i) January 30, 2004  
[http://www.xilinx.com/ise/embedded/est\\_guide.pdf](http://www.xilinx.com/ise/embedded/est_guide.pdf)
4. Xilinx, Inc., Tutorial: Designing Custom OPB Slave Peripherals for MicroBlaze, February 8, 2002  
[http://www.xilinx.com/ipcenter/processor\\_central/microblaze/doc/opb\\_tutorial.pdf](http://www.xilinx.com/ipcenter/processor_central/microblaze/doc/opb_tutorial.pdf)
5. Xilinx, Inc., Xilinx CORE Generator System,  
<http://www.xilinx.com/ipcenter/coregen/updates.htm>

## Revision History

The following table shows the revision history for this document.

Date	Version	Revision
11/26/02	1.0	Initial Xilinx release.
04/09/03 & 04/18/03	1.1	Revised <a href="#">Figure 5</a> , <a href="#">Figure 10</a> , <a href="#">Figure 11</a> , <a href="#">Figure 14</a> , and <a href="#">Table 1</a> . Added section on “ <a href="#">Reference Design</a> ” files.
07/02/04	1.2	Replaced "Including the Peripheral in a Platform" section with the “ <a href="#">The OPB PPC Demo and Test Programs</a> ” section.