Xilinx System Generator v1.0.1 for Simulink

Xilinx Blockset Reference Guide

—

Printed in U.S.A.

About This Manual

This guide should be used as a reference guide for system designers who are unfamiliar with the Xilinx Blockset elements. This is the Blockset provided with the Xilinx System Generator v1.0.1 for Simulink® software.

Note This Xilinx software release is certified as Year 2000 compliant.

Manual Contents

This guide covers the following topics.

- Chapter 1, “Introduction”
- Chapter 2, “Xilinx Blockset Overview” gives an explanation of Xilinx Blockset elements, how to instantiate them within your Simulink model, how to configure them through their Parameterization GUI, common options that can be used in several of the elements, and the nature of the signals used in the System Generator.
- Chapter 3, “Blockset Elements” describes the details of each Blockset element, including options, use of Xilinx LogiCOREs™, and filenames pointing to descriptions of the cores on your local computer.

Additional Resources

For additional information, go to `http://support.xilinx.com`. The following table lists some additional resources.

Resource	Description/URL
Tutorials	Tutorials covering Xilinx design flows, from design entry to verification and debugging. `http://support.xilinx.com/support/techsup/tutorials/index.htm`
IP Center	Information on Xilinx Cores and IP solutions. `http://www.xilinx.com/ipcenter/`
Xilinx DSP	Xilinx DSP product information. `http://www.xilinx.com/products/logicore/dsp`
Technical Tips	Latest news, design tips, and patch information for the Xilinx design environment. `http://support.xilinx.com/support/techsup/journals/index.htm`
The MathWorks	MATLAB® , Simulink® , DSP design, and other company information. `http://www.mathworks.com`

Conventions

This manual uses the following conventions. An example illustrates each convention.

Typographical

The following conventions are used for all documents.

- `Courier font` indicates messages, prompts, and program files that the system displays.

  ```
  speed grade: - 100
  ```

- **`Courier bold`** indicates literal commands that you enter in a syntactical statement. However, braces "{ }" in Courier bold are not literal and square brackets "[ ]" in Courier bold are literal only in the case of bus specifications, such as bus [7:0].

  `rpt_del_net=`

  `Courier bold` also indicates commands that you select from a menu.

  `File` → **`Open`**

- *Italic font* denotes the following items.

  - Variables in a syntax statement for which you must supply values

    `edif2ngd` *`design_name`*

  - References to other manuals

    See the *Development System Reference Guide* for more information.

  - Emphasis in text

    If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.

- Square brackets "[ ]" indicate an optional entry or parameter. However, in bus specifications, such as bus [7:0], they are required.

  `edif2ngd` *`[option_name] design_name`*

- Braces "{ }" enclose a list of items from which you must choose one or more.

  `lowpwr ={on|off}`

- A vertical bar " | " separates items in a list of choices.

  `lowpwr ={on|off}`

- A vertical ellipsis indicates repetitive material that has been omitted.

```
IOB #1: Name = QOUT'
IOB #2: Name = CLKIN'
.
.
.
```

- A horizontal ellipsis "…." indicates that an item can be repeated one or more times.

  `allow block` *block_name loc1 loc2 ... locn;*

Contents

Chapter 1

Introduction

Product Overview

The Xilinx System Generator™ enables the development of high-performance DSP systems for Xilinx FPGAs using the popular MATLAB® and Simulink® products from The MathWorks, Inc. This software tool automatically generates Hardware Description Language (HDL) code from a system representation in Simulink. The HDL design is optimized for synthesis and implementation in Xilinx Virtex™ and Spartan®-II FPGAs. To maximize predictability, density, and performance, the tool automatically maps the system design to Xilinx optimized LogiCORE™ modules. Because the HDL is automatically generated, you must verify only the system representation of the design. With only one design representation, risk of errors is minimized.

More information about the System Generator product and features can be found in the *System Generator Quick Start Guide*.

Bit true and Cycle true representation

The Xilinx System Generator supports *bit true* and *cycle true* modeling of hardware. The definitions of bit true and cycle true modeling are explained here.

Simulink is an event driven simulator for dynamic systems, including

- continuous time and space systems defined via state space equations (i.e. differential equations), and
- discrete time and discrete space (countable, and for all intents and purposes, finite index sets).

Discrete time and space simulation is important because it allows System Generator to model the evolution of hardware over time. In theory, it is possible to understand the bit and cycle behavior of the generated VHDL from Simulink, as opposed to from a behavioral VHDL simulator.

Signals in System Generator are represented as arbitrary precision fixed point data, which in VHDL corresponds to standard logic vectors. If you examine a System Generator signal in Simulink, you will see that its fixed point value consists of the same bits as the corresponding bits of the standard logic vector in VHDL. This is an example of how System Generator is bit true.

In addition to the fixed point value, every System Generator signal is sampled, and has an associated sample period. If you examine a signal in Simulink (using a Simulink Scope block), you will see that transitions occur only at multiples of the sample period for the block that drives the signal. In the VHDL generated by System Generator, the corresponding standard logic vector is driven by a block that is clocked (or if combinational, has an “inherited” clock period from its inputs) at a particular

clock rate. The corresponding sample period in Simulink is guaranteed to be a multiple of the hardware clock period. At the clock transitions that correspond to sample period multiples, the bits in the standard logic vector (VHDL) match the fixed point data in the Simulink signal (software). This is an example of how System Generator is cycle true.

You may sometimes refer to models that are bit true, but not cycle true. Whatever that may mean, with System Generator, there is no ambiguity: the bits in a standard logic vector are identical to the bits in the corresponding fixed point System Generator Simulink signal at the hardware clock transitions that correspond with multiples of the Simulink sample period.

Xilinx Blockset

The Xilinx Blockset is a collection of Simulink blocks that can be used to create and simulate designs within the Simulink environement. This Reference Guide explains the contents of the Xilinx Blockset and describes each element of the blockset in detail.

The Xilinx Blockset conforms to familiar Simulink idioms wherever possible, for example in propagating signal types through a model. Consequently, the Simulink documentation can often provide useful reference information and insight about the mechanics of models built from the Xilinx Blockset.

Chapter 2

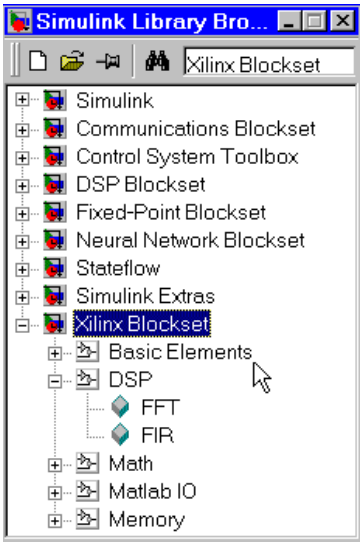
Xilinx Blockset Overview

This chapter gives an overview of the Xilinx Blockset, including background information on underlying blockset implementation, which will help you understand how each element can be used to create and simulate your designs.

This chapter contains the following sections.

- "What is a Xilinx Blockset Element?"
- "Instantiating Xilinx Blockset elements within a Simulink model"
- "The Parameterization GUI"
- "The Nature of Signals in the Xilinx Blockset"
- "Use of Xilinx Smart-IP Cores by the System Generator"
- "Common Options in Xilinx Blockset Element Parameterization GUI"

What is a Xilinx Blockset Element?



The Xilinx Blockset is a Simulink library, accessible from the Simulink library browser. It consists of building blocks that can be instantiated within a Simulink model, and like other Simulink blocksets, elements can be combined to form subsystems and arbitrary hierarchies. The Xilinx Gateway blocks (from the Xilinx Blockset's MATLAB I/O library) are used to interface between the Xilinx Blockset fixed-point data type and other Simulink blocks.

Every Xilinx Blockset element can be configured via a parameterization GUI, with few exceptions even during simulation. Many blocks share common parameters, which are described later in this document. Most also have parameters specific to the function computed.

System Generator has the ability to generate an FPGA implementation consisting of RTF VHDL and Xilinx Smart-IP™ Cores from a Simulink subsystem built from the Xilinx Blockset. The overall design, including test environment, may consist of arbitrary Simulink blocks. However the portion of a Simulink model to be implemented in an FPGA must be built exclusively of Xilinx Blockset elements, with the exception of subsystems denoted as black boxes.

Instantiating Xilinx Blockset elements within a Simulink model

Xilinx blocks can be dragged (from the Simulink library browser, or from an expanded sheet showing the blocks in the library) onto a Simulink model sheet. Double-clicking on a block will open its parameterization GUI and allow customization of that instance of the block. It is also possible to build user libraries of customized blocks and subsystems. Refer to the manual: *Using Simulink* from The MathWorks.

The Xilinx Blockset elements operate on fixed-point data, using an arbitrary precision arithmetic type. The Gateway blocks found in the Xilinx MATLAB I/O library comprise the interface between Xilinx blocks and other Simulink blocks, and enable Xilinx blocks to be freely instantiated within a Simulink model. Of course, the only blocks that System Generator will convert to hardware are those from the Xilinx Blockset.

The Parameterization GUI

Most Xilinx blocks have parameters that can be configured. The typical element has a parameterization GUI with several common parameters (common to most blocks in the blockset) and some specific parameters (specific to the particular block only). Double-clicking on any block on a sheet will open its parameterization GUI. Details of the use of each element's parameterization GUI may be found elsewhere in this document.

Each parameterization GUI contains four buttons: OK, Cancel, Help, and Apply. Apply applies your configuration changes to the block, leaving the GUI still visible on your screen. Help launches HTML help on the block. Cancel closes the GUI without saving any changes, and OK applies your configuration changes and closes the parameterization GUI window.

Figure: buttons common to each parameterization GUI

The Nature of Signals in the Xilinx Blockset

Simulink blocks use double-precision, floating point signals and arithmetic. However, when these signals pass through a Xilinx Gateway In block, they are converted to fixed point signals. Later, when passing through the Xilinx Gateway Out block, the signals are converted back into double-precision floating point signals.

The fixed point signals use arbitrary precision arithmetic internally. In an arbitrary precision system, there is no danger of overflow or rounding error, because the system computes the amount of precision necessary to perform the requested arithmetic functions, and uses the correct number of bits to represent the output. Xilinx blocks typically default to *full precision* which enables their arbitrary precision calculations to use the full number of bits necessary.

You have the option of changing blocks to *user defined* precision in which specific numbers of bits for precision can be specified. For example, shown below is the

Multiplier GUI with full precision chosen, then with user defined precision chosen. Note the additional options that you may set if you choose user defined precision.

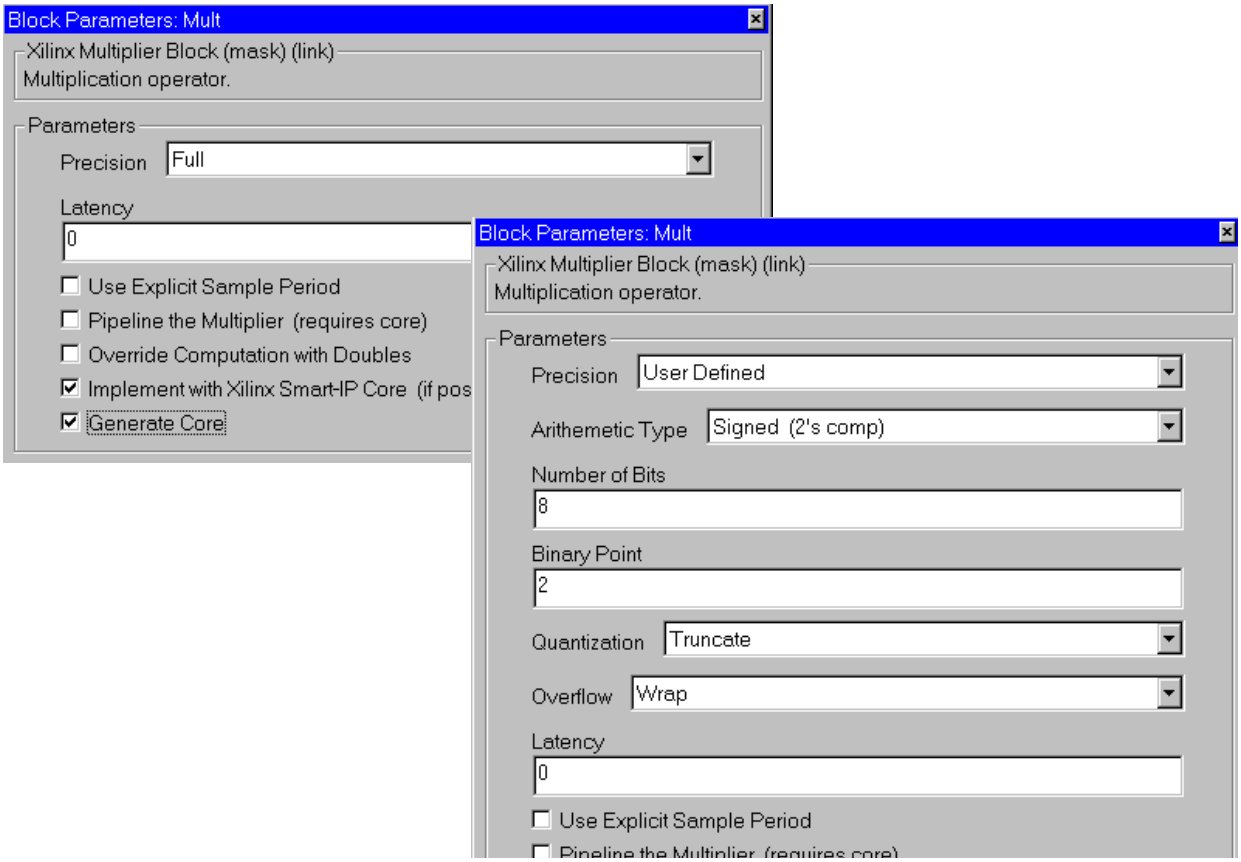


Figure: User-Defined precision options (available if selected instead of Full precision)

Valid and Invalid Data

In the Xilinx Blockset portion of a Simulink model, every data sample that flows through the model is accompanied by a handshake validation signal. In the corresponding hardware, every data-carrying bus has a companion net that carries a valid or invalid status indicator. This is a handshaking mechanism often seen in dataflow tools. There are different circumstances under which the status indicator may be set to invalid. For example, invalid data might mean that a pipelined dataflow hasn't yet filled up, or it may denote bursty outputs, as with an FFT. Elements in the Xilinx Blockset can use this valid bit signal to decide what to do with the input data. Some of the Xilinx blocks, for example the storage elements and the FFT, use the valid bit to determine when it is time to store input data.

Port Data Types

Selecting the Port Data Types option (under the Format menu in the Simulink GUI) shows the data type and precision of a signal. An example port data type string is **Fix_11_9**, which indicates that the signal is a signed 11-bit number with the binary point 9 bits from the right side. Similarly, an unsigned signal is indicated by the **UFix_** prefix.

Use of Xilinx Smart-IP Cores by the System Generator

All Xilinx blocks generate synthesizable VHDL code, and if requested, testbench code and testbench data vectors. Most blocks also generate a Xilinx LogiCORE™ using the Xilinx CORE Generator™. Xilinx LogiCOREs are particularly suited for optimal implementations of specific functions for Xilinx FPGA devices.

Some Xilinx blocks generate VHDL, but no Xilinx LogiCORE. For example, the Xilinx Constant block does not use a core.

Some Xilinx blocks generate a Xilinx LogiCORE if possible and if requested, or synthesizable VHDL otherwise. For example, the Xilinx Multiplier block generates a core if you specify inputs of up to 32 bits. If either input is larger than 32 bits, a Xilinx LogiCORE will not be used, and the multiplier will instead be created entirely in synthesizable VHDL.

Some Xilinx blocks require a Xilinx LogiCORE. For example, the Xilinx FFT block cannot be generated as synthesizable VHDL. You are required to use one of the supported parameter configurations for the Xilinx FFT, any of which will generate a configuration of the Xilinx FFT LogiCORE. The descriptions of the specific Xilinx Blockset elements note which Xilinx LogiCOREs are used by each.

For those Xilinx blocks that have associated Xilinx LogiCOREs, you also have options regarding when to generate the core and core files. You may want to defer invoking the Xilinx CORE Generator until you have finished debugging your design and are ready to implement. Or, you may already have a Xilinx LogiCORE previously generated, and may not wish to take the time to re-generate the core every time you invoke the System Generator code-generation software. The System Generator's optional invocation of the Xilinx CORE Generator can be configured as part of each Xilinx block's parameterization GUI, or from the System Generator token's parameterization GUI.

Xilinx LogiCORE™ Versions

The Xilinx LogiCOREs™ that are used in v1.0.1 of the Xilinx System Generator are listed below, with the version numbers being supported by the System Generator:

Xilinx Blockset element	Xilinx LogiCORE	Version
Accumulator	ACCUMULATOR	v2.0
Counter	COUNTER_BINARY	v2.0
Dual Port Ram	MEM_DP_BLOCK	v1.0
FFT	FFT and MEM_DP_BLOCK	v1.0, v1.0
FIR filter	DA_FIR	v3.0 (SysGen v1.0) v4.0 (SysGen v1.0.1)
Inverter	GATE_BUS_	v1.0
Logical	GATE_BUS	v1.0
Multiplier	MULT_VGEN	v2.0
Mutiplexer	MUX_BUS	v2.0
Negate	TWOS_COMP	v2.0
Relational	COMPARE	v2.0

Xilinx Blockset element	Xilinx LogiCORE	Version
Single Port RAM	MEM_SP_BLOCK and DIST_MEM	v1.0, v2.0
ROM	MEM_SP_BLOCK and DIST_MEM	v1.0, v2.0

Common Options in Xilinx Blockset Parameterization GUI

Each Xilinx block has several configurable parameters, seen in the Block parameterization GUI. Many of these parameters are specific to that particular block, and those parameters are described in the specific block documentation in the next chapter of this Reference Guide.

The remainder of the parameters in each block's parameterization GUI are common to several blocks. These common parameters are described below.

Arithmetic Type

In the Type field of the Xilinx Blockset parameterization GUI, you can choose unsigned or signed (two's complement) as the datatype of the output signal.

Implement with Xilinx Smart-IP™ Core (if possible)

This checkbox (sometimes referred to as the *use core* checkbox) specifies to the System Generator code-generation software to instantiate a core in the generated VHDL. If you do not select this checkbox, then you are specifying that the System Generator should create synthesizable VHDL, and should not use the Xilinx LogiCORE at all.

Selecting this option does not guarantee that a Xilinx LogiCORE will be used. If the parameters for your block are such that a core cannot be generated (for example if you have specified a multiplier that is larger than available Xilinx LogiCOREs), then synthesizable VHDL will be generated instead. The System Generator software makes this decision at code-generation time.

If you select this checkbox, the Xilinx CORE Generator software will also generate the behavioral VHDL model that models the simulation of the Xilinx LogiCORE.

Generate Core

When the Generate Core checkbox is selected, the Xilinx CORE Generator will be invoked during System Generator code-generation. If Generate Core is not selected, then a Xilinx LogiCORE will not be generated, and if the core doesn't already exist in your project directory, subsequently running the Xilinx Implementation tools will produce an error.

If you select "Implement with Xilinx Smart-IP Core" but do not select "Generate Core," you will be able to simulate your generated VHDL because (1) a core will be instantiated in the VHDL, and (2) the behavioral VHDL models will be available for a simulator to use. However, you will not be able to complete implementation into a Xilinx FPGA until you have also generated the core.

In some blocks, only the Generate Core option is available. If the Implement with Smart IP-Core option is not available, then there is only a core implementation available from the System Generator, no synthesizable VHDL implementation.

Latency

Many elements in the Xilinx Blockset have a latency option, which defines the number of input sample periods required for an input to affect a block output. This sample period may correspond to multiple clock cycles in the corresponding FPGA implementation, for example when the hardware is overclocked with respect to the Simulink model. System Generator does not perform extensive pipelining; additional latency is usually implemented as a shift register on the output of the block.

Number of Bits

If you have specified user-defined precision, then you will be asked to specify how many bits you would like the output to have.

Binary Point

You will also be asked to specify how many bits are to the right of the binary point (i.e., the size of the fraction). The binary point position must be between zero and the number of bits in the number's container.

Overflow and Quantization

When user-defined precision is selected, there is the possiblity of data error due to overflow or quantization. Overflow occurs if the data value lies outside of the representable range of the fixed point number. Quantization error occurs if the number of fractional bits is insufficent to represent the fractional portion of the data.

The Xilinx fixed-point data type supports several options for user-defined precision. In the case of overflow, the options are to saturate to the largest positive (or smallest negative) value when the data value exceeds the representation, wrap the value (i.e. discard any significant bits beyond the most-significant bit in the fixed-point number), or flag an overflow as a Simulink error during simulation.

In the case of quantization, the options are: (a) to round to the nearest representable value or to the values toward +/- infinity if there are two equidistant nearest representable values, or (b) to truncate the data (i.e. discard bits to the right of the least significant bit).

It is important to realize that whatever option is selected, the generated HDL model will have identical bit-true behavior as the Simulink model.

Override with Doubles

Regular Simulink blocks use double-precision, floating point signals and arithmetic. However, when these signals pass through the Xilinx Gateway In block (input to the FPGA portion of your Simulink design), they are converted to fixed point signals. Later, when passing out of the FPGA portion of your design through the Xilinx Gateway Out block, the signals are converted back into double-precision floating point signals.

For the purpose of simulation in the Simulink environment, the *override with doubles* option allows you to simulate the entire design in double-precision floating point.

This option is useful in selecting fixed point widths, or if you are not getting the simulation results that you expect when simulating the Xilinx portion of your design with fixed-point signals. If you find simulation errors with fixed-point signals, you can choose to simulate your entire design, or only specific blocks, using double-

precision floating point signals and arithmetic operations. This option will help you discover which part of your design has quantization error.

You may choose override with doubles on a particular block. You may also choose this option for an entire sheet, or an entire subsystem (the sheet plus underlying hierarchy) by instantiating a System Generator token on the sheet, and choosing override with doubles on the token's parameterization GUI.

When the output of one block with override with doubles set is connected to the input of another block where the option is also set, data samples will be transmitted between them in double precision. Thus, there will be no quantization effect due to the transmission of data between them.

You can easily identify which blocks are currently set to override with doubles. When this option is set, affected Xilinx blocks are displayed in gray, rather than in their normal blue or yellow colors.

Precision

The fundamental computational mode in the Xilinx Blockset is arbitray precision fixed-point arithmetic. Most blocks give you the option of choosing the precision, i.e. the number of bits in the number and the number of fractional bits (binary point position).

By default, the output of Xilinx blocks is *full* precision, that is, a sufficient number of total and fractional bits to represent the result without error. Most blocks have a *user-defined* precision option which fixes the number of total and fractional bits.

Sample Period

The data streams are processed at a specific sample rate, or clock period, as they flow through a dataflow system such as Simulink. Typically, each block detects the input sample rate and produces the correct sample rate on its output. Xilinx Blockset elements Up Sample and Down Sample provide a means to increase or decrease sample rates.

Use Explicit Sample Period

If you select Use Explicit Sample Period rather than the default, you may set the sample period required for all the block outputs. This is useful when implementing features in your design such as feedback loops. In a feedback loop, it is not possible for the System Generator to determine a default sample rate, because the loop makes an input sample rate depend on a yet-to-be-determined output sample rate. The System Generator therefore requires you to supply a hint to establish sample periods throughout a loop.

The following images (the Concat block's parameterization GUI) show the results with Use Explicit Sample Period selected and unselected.

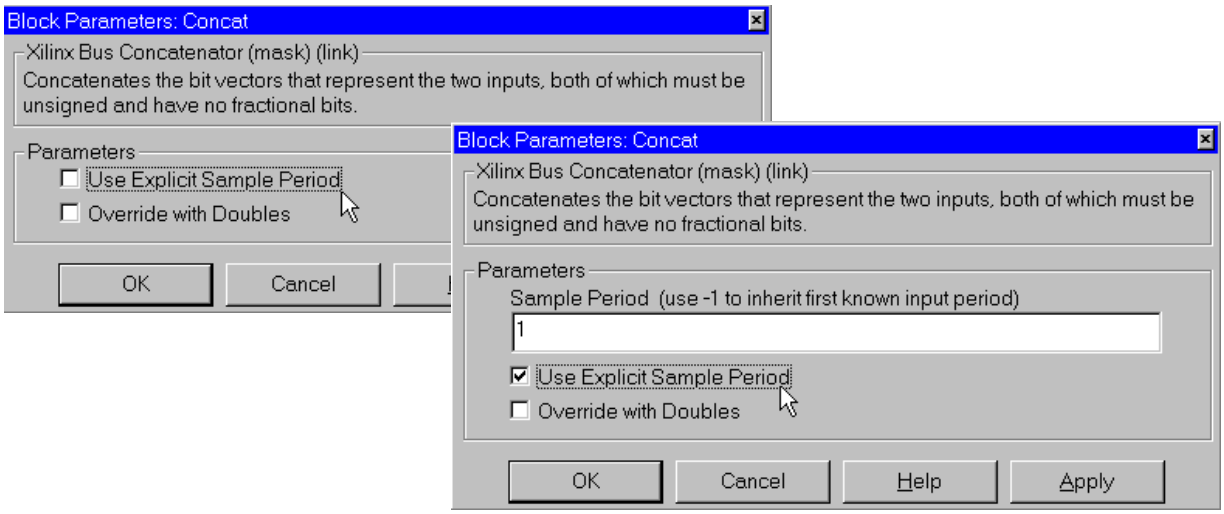


Figure: Use Explicit Sample Period options (available if selected)

Chapter 3

Blockset Elements

This chapter describes specific details of the each Xilinx Blockset element. Xilinx blocks are grouped within these five categories, also shown in the Simulink library browser:

- "Basic Elements"
- "DSP"
- "Math"
- "Matlab IO"
- "Memory"

Basic Elements

The Xilinx Basic Elements Library includes the standard building blocks for digital designs. Using these elements, you may insert delay, change the sample rate, and introduce constants, counters, multiplexers, etc.

In addition, there are two special elements in the Basic Elements Library: the System Generator and the Black Box.

System Generator

The System Generator is a special Xilinx block.

The System Generator invokes the tool's *Code Generation Software*. By placing the System Generator token on your Simulink project sheet, you can generate HDL and Xilinx LogiCOREs for all the Xilinx blocks on that sheet and on any sheets beneath it in the hierarchy.

Double-clicking on the System Generator token produces a customization GUI, which allows you to tailor your Simulink simulation and code-generation.

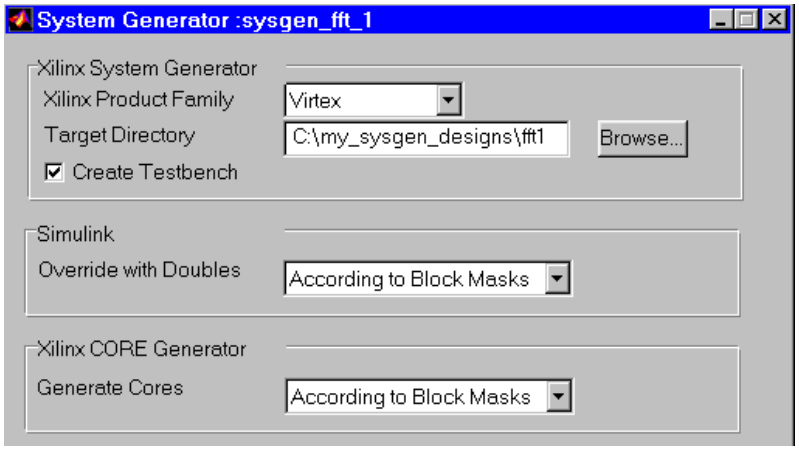


Figure: System Generator customization GUI

The System Generator token allows you to hierarchically override fixed point values with double-precision values for your Simulink simulation, which is particularly useful during design and debugging. The override with doubles directive from a System Generator token is applied to all Xilinx block on the same sheet and recursively through all subsystems on the sheet. Additional System Generator tokens can be inserted into the subsystems to selectively mask this effect. For an explanation of the *Override with Doubles* behavior, seethe Common Parameters section of the previous chapter.

You may determine whether Xilinx LogiCOREs should be generated (as edif) along with the HDL files, via the Generate Cores pulldown menu.

By checking the Create Testbench box, you instruct the tool to save test vectors that are imported and used during behavioral simulation.

Finally, selecting the Generate button invokes the code generation software, and your Simulink design is converted to VHDL and Xilinx LogiCOREs. Note that the Cancel button is active during code generation; if you wish to cancel the code-generation phase while it is running, you may do so by selecting Cancel during code-generation.

Black Box

You may wish to include functionality in your Simulink model that does not exist in the current blockset. Any Simulink subsystem may be treated as a black box if you so choose. You may want to build a model out of non-Xilinx blocks for an HDL representation of functionality that you wish to turn into a Simulink model.

The Xilinx Black Box token gives you the ability to instantiate your own specialized functions in your model, and subsequently into a generated design.

Like the System Generator token, the Black Box token may be placed in any Simulink subsystem, identifying the subsystem as a black box.

The black box customization GUI encapsulates the design information necessary for the compiler to create the correct instantiation interfaces. This black box support

allows you to abstract commonly used control signals and ports, and then infer them within the generated VHDL.

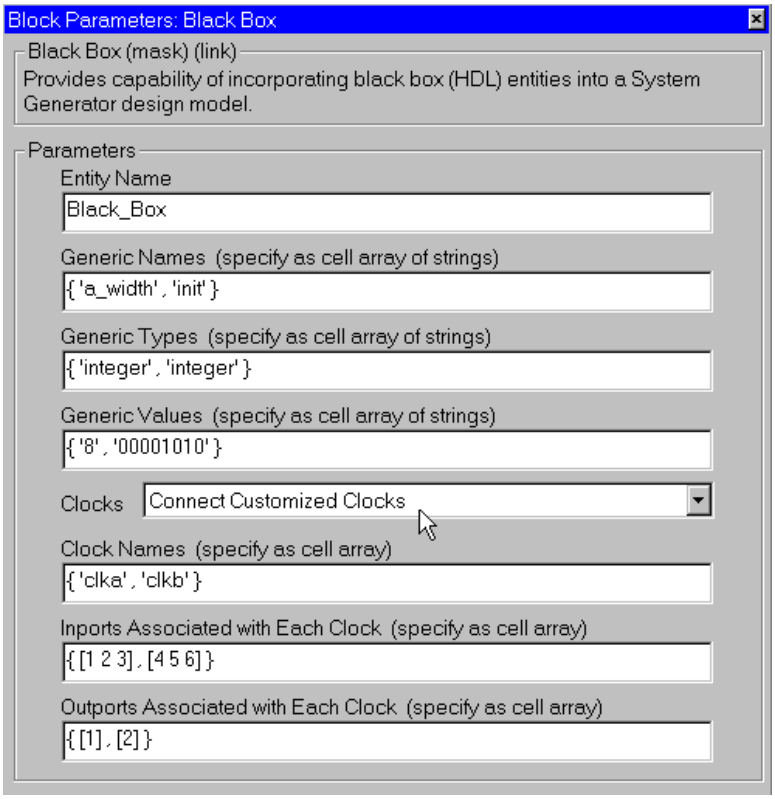


Figure: Black Box customization GUI

The parameters that are specified as a cell array (generic names, types, and values) permit several methods for entering data. You may specify your data directly in the GUI as shown. You may also specify the cell arrays as MATLAB expressions. This is useful if you have many elements in your cell arrays.

Generic types can be any VHDL type.

The black box configuration GUI allows you to specify multiple clocks on a black box.

In order to handle more than one clock in your black box design, the System Generator needs to know how fast each clock should run. To specify a clock's speed, you must associate the clock to a port on the black box: The frequency of the clock is then the frequency of the signal passing through the port. The System Generator allows more than one port to be associated to a clock, but only if all the associated ports have the same frequency (note: constant inputs will match any paired frequency).

For example, assume you have a black box with two ports: a fast input, and a slow output. Say the black box should have clocks called `fast_clk` and `slow_clk` whose frequencies should match those of the input and output ports respectively. In

order to configure the black box, enter the following in the black box configuration GUI:

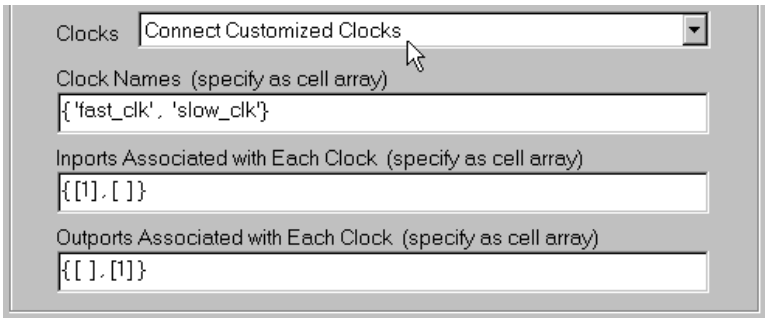


Figure: Customizing clocks in the Black Box GUI

This configuration indicates that the black box should have clocks named `fast_clk` and `slow_clk`. The `fast_clk` should have the same frequency as the samples presented to input port #1, and the `slow_clk` should have the same frequency as output port #1.

Concat

The Xilinx Concat block performs a concatenation of two bit-vectors represented by unsigned integer numbers, i.e. two unsigned numbers with binary points at position zero.

The block has two input ports and one output port. The two input ports are labeled **`hi`** and **`low`**. The number input to the `hi` port will occupy the most significant bits of the output and the number that is input to the `low` port will occupy the least significant bits of the output.

The Concat block may be configured via its parameterization GUI.

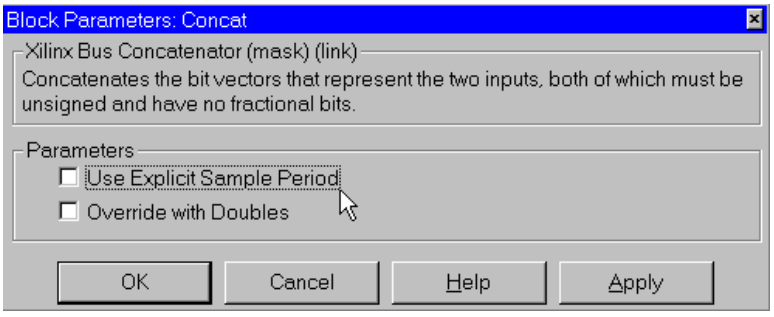


Figure: Concat block parameterization GUI

Parameters used by this block are explained in the Common Parameters section of the previous chapter of the Reference Guide.

The Concat block does not use a Xilinx LogiCORE.

Constant

The Xilinx Constant block generates a constant. This element is similar to the Simulink constant block, but can be used to drive the inputs on Xilinx blocks.

The Constant block may be configured via its parameterization GUI.

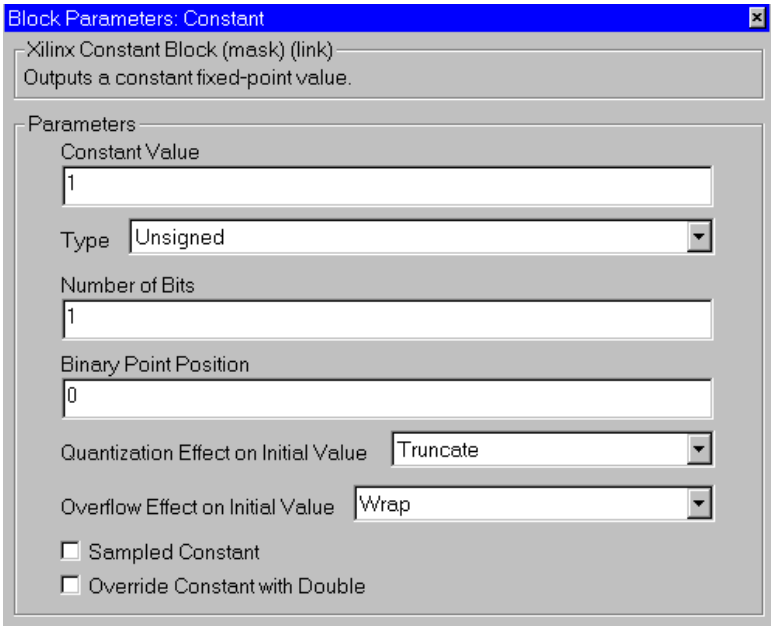


Figure: Constant block parameterization GUI

You may specify the value of the constant, as well as its arithmetic type, number of bits (in a user-defined precision), plus effect of overflow or quantization on the value. The input displayed on the block will represent the specified quantization and overflow effects.

The sampled constant option allows a sample period to be associated with the constant output and inherited by blocks that the constant block drives. (This is a useful option solely because the blocks eventually target hardware and the sample periods of Simulink are used to establish hardware clock periods.)

Other parameters used by this block are explained in the Common Parameters section of the previous chapter of the Reference Guide.

The Constant block does not use a Xilinx LogiCORE.

Convert

The Xilinx Convert block converts each input sample to a number of a desired arithmetic type. For example, a number can be converted to a signed (two's complement) or unsigned value.

The Convert block may be parameterized via its Block Parameters GUI.

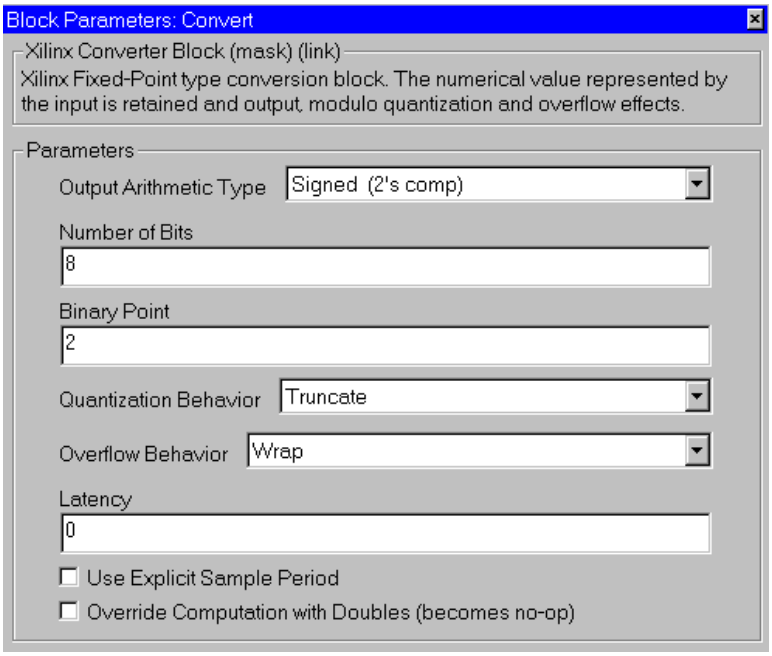


Figure: Convert block parameterization GUI

All the parameters of the Convert block are parameters common to more than one block. Please refer to the Common Parameters section in the previous chapter for more details about the parameters that you may set for this element.

Parameters defining the desired output type are:

- Output Arithmetic Type
- Number of Bits
- Binary Point

Parameters defining the quantization effect and the overflow effect are:

- Quantization Behavior
- Overflow Behavior

The Convert block does not use a Xilinx LogiCORE.

Counter

The Xilinx Counter block implements a counter. It may be an up counter or a down counter. It may be configured to start and end at any value, and to increment by any value. The increment must be evenly divisible by the difference between the counter's starting and ending values.

The Counter parameterization GUI may be invoked by double-clicking on the block icon.

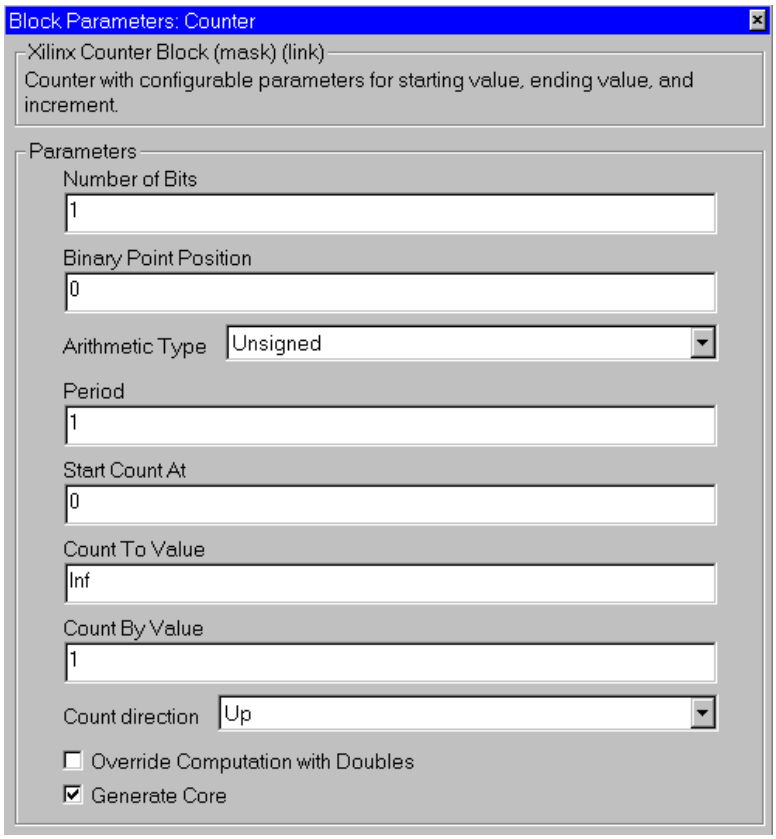


Figure: Counter block parameterization GUI

Parameters specific to the block are:

- Start Count at: this is the initial value of the counter. The default value is zero.
- Count to Value: this is the “Stop counter” value, the number at which the counter will stop. The default value is “Inf” which is a MATLAB internal value interpreted hear to mean the largest representable output, ala the “max” option for the core. Note that the “Inf” value may not be used in VHDL.
- Count By Value: this is the counter’s increment. The default value is 1. This number cannot be the same as the start count.
- Count direction: choice of up- or down-counter.

Other parameters used by this block are explained in the Common Parameters section of the previous chapter of the Reference Guide.

Xilinx LogiCORE

The Counter block always uses the Binary Counter Xilinx LogiCORE. The Core datasheet may be found on your local disk at:

```
%XILINX%\coregen\ip\xilinx\baseblox_v2_0\com\xilinx\ip\baseblox_v2_0\doc\C_COUNTER_BINARY_V2_0.pdf
```

Delay

The Xilinx Delay block is a delay line (also called shift register) of configurable length, allowing you to add latency to your design. You may wish to add latency to balance pipelining elsewhere in your design. The System Generator implements latency by adding the appropriate number of registers to your design at the position where this block is placed.

Data presented at input will appear at the output after a user specified number of sample periods. Initial output samples are designated as “invalid data.”

The Delay block differs from the Register block because the Register only allows latency of 1, and contains an Initial Value parameter. The Delay block supports a user specified latency, but no initial value.

The Delay block is configurable via its parameterization GUI.

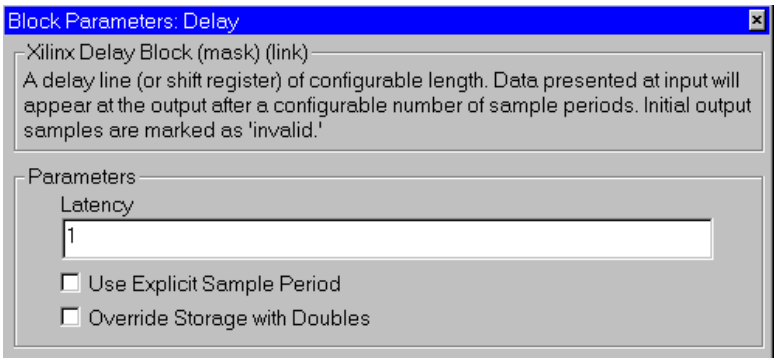


Figure: Delay block parameterization GUI

You may enter the amount of latency in the Latency field.

Other parameters used by this block are explained in the Common Parameters section of the previous chapter of the Reference Guide.

The Delay block does not use a Xilinx LogiCORE, but the synthesizable output will be efficiently mapped to utilize the SRL16 feature of the Xilinx parts.

Down Sample

The Xilinx Down Sample block reduces the sample rate at the point where the block is placed in your design. The input signal is under-sampled so that every nth input sample is presented at the output and held.

Output sample period is **k***(input sample period) where the ratio **k** is a configurable block parameter.

The Down Sample block is configured via its parameterization GUI:

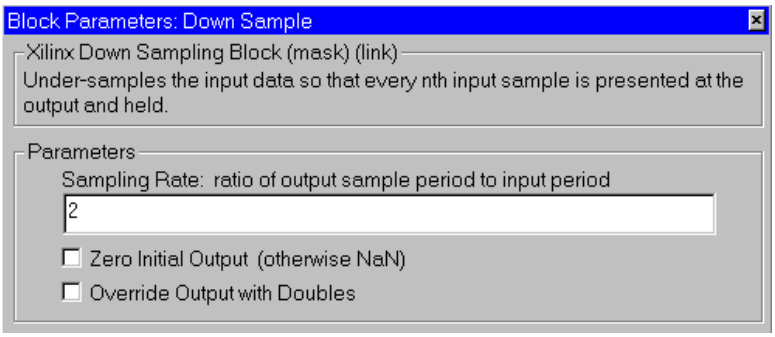


Figure: Down Sample block parameterization GUI

The *Sampling Rate* field must contain an integer, greater or equal to 2. This is the ratio of the output sample period to the input, and is essentially a sample rate divider. For example, a ratio of 2 indicates a 2:1 divider of the input sample rate. If a non-integer ratio is desired, the Up Sample block can be used in combination with the Down Sample block to create smaller differences in sample rate.

The *Zero initial output* checkbox lets you choose what the first value of the new sample (before it has valid data) will be. By selecting Zero Initial Output, you can validate the first sample with valid data of zero. Otherwise, an invalid data ("NaN = not a number") will be the block's first output.

Other parameters used by this block are explained in the Common Parameters section of the previous chapter of the Reference Guide.

The Down Sample block does not use a Xilinx LogiCORE.

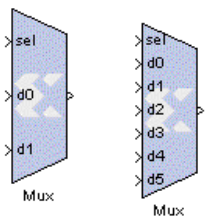
Get Valid Bit

The Xilinx Get Valid Bit element sets its output to 1 when its input is a valid data value. The output is set to 0 otherwise.

In the Xilinx Blockset, every data sample that flows through the model is accompanied by a handshake validation signal. In the corresponding hardware, every data-carrying bus has a companion net that carries a valid or invalid status indicator. There are different circumstances under which the status indicator may be set to invalid. For example, invalid data might mean that a pipelined dataflow hasn't yet filled up, or it may denote bursty outputs, as with an FFT. This block simply reports the valid status of the samples presented to it.

There are no configurable parameters for this block.

Mux



The Xilinx Mux block implements a multiplexer. Inputs include the select line and a configurable number of data lines. From 2 to 8 lines may be set, with the input of the select line determining which input line of the multiplexer will be passed through to the block's output.

The Mux block can be configured via its parameterization GUI.

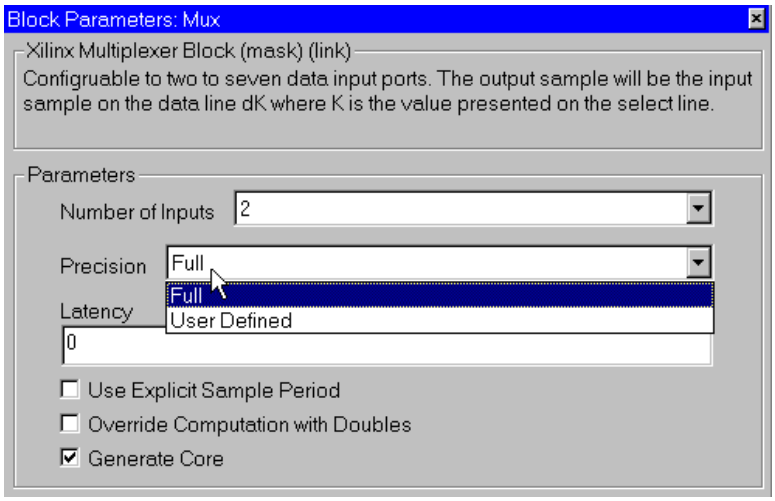


Figure: Mux block parameterization GUI

You may select the number of inputs, and after applying this selection to the block, the Mux token on your sheet will change to graphically show the proper number of inputs.

The multiplexer precision defaults to Full. If User-Defined precision is selected, the parameterization GUI will then also include configurable parameters: Arithmetic Type, Number of Bits, Quantization and Overflow. These parameters are explained in the Common Parameters section of the previous chapter of the Reference Guide.

Xilinx LogiCORE

The Multiplexer block always uses the Xilinx Bus Multiplexer V2.0 LogiCORE.

The Core datasheet may be found on your local disk at:

```
%XILINX%\coregen\ip\xilinx\baseblox_v2_0\com\xilinx\ip\baseblox_v2_0\doc\C_MUX_BUS_V2_0.pdf
```

Register

The Xilinx Register block inserts one register into your design. Each register adds a latency of 1 to your design. There are two inputs to the block: a data line and a reset line. Data presented at input will appear at the output after one sample period. The initial output value is configurable.

The Register block differs from the Delay block because the Register always gives latency of 1, and contains an Initial Value parameter. The Delay block allows a configurable latency amount, but has no value.

The Register block may be configured via its parameterization GUI:

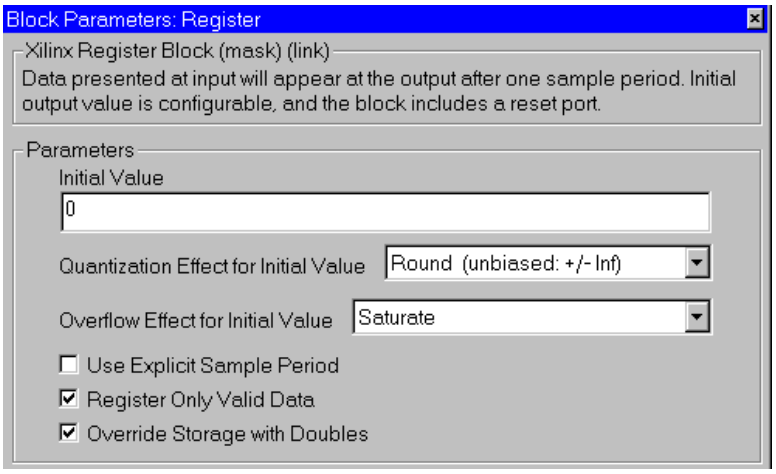


Figure: Register block parameterization GUI

The Register Valid Data checkbox, if selected, will not register invalid data as output.

Other parameters used by this block are explained in the Common Parameters section of the previous chapter of the Reference Guide.

The Register block does not use a Xilinx LogiCORE.

Set Valid Bit

The Xilinx Set Valid Bit block flags input data as invalid when the signal on the valid bit input port is zero. This block only sets data invalid; invalid input data cannot be changed to valid. Invalid data is converted to NaN (Not a Number) by the Xilinx Gateway-out block.

In the Xilinx Blockset, every data sample that flows through the model is accompanied by a handshake validation signal. In the corresponding hardware, every data-carrying bus has a companion net that carries a valid or invalid status indicator. This block provides some explicit control over this hanshake mechanism.

The Set Valid Bit block may be configured via the parameterization GUI:

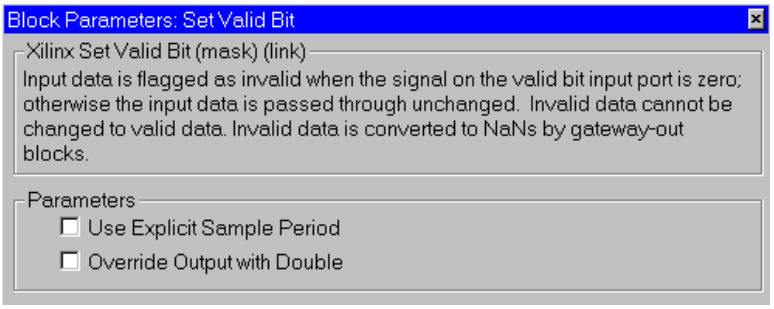


Figure: Set Valid Bit block parameterization GUI

Slice

The Xilinx Slice block allows you to “slice off” a sequence of bits from your input data and create a new data value. This new value is presented as the output from the block. The output is of type UFix_N_0.

Specifications as to how to extract your data slice and how to construct the new data value are included in the parameterization GUI. Two views of the GUI are shown:

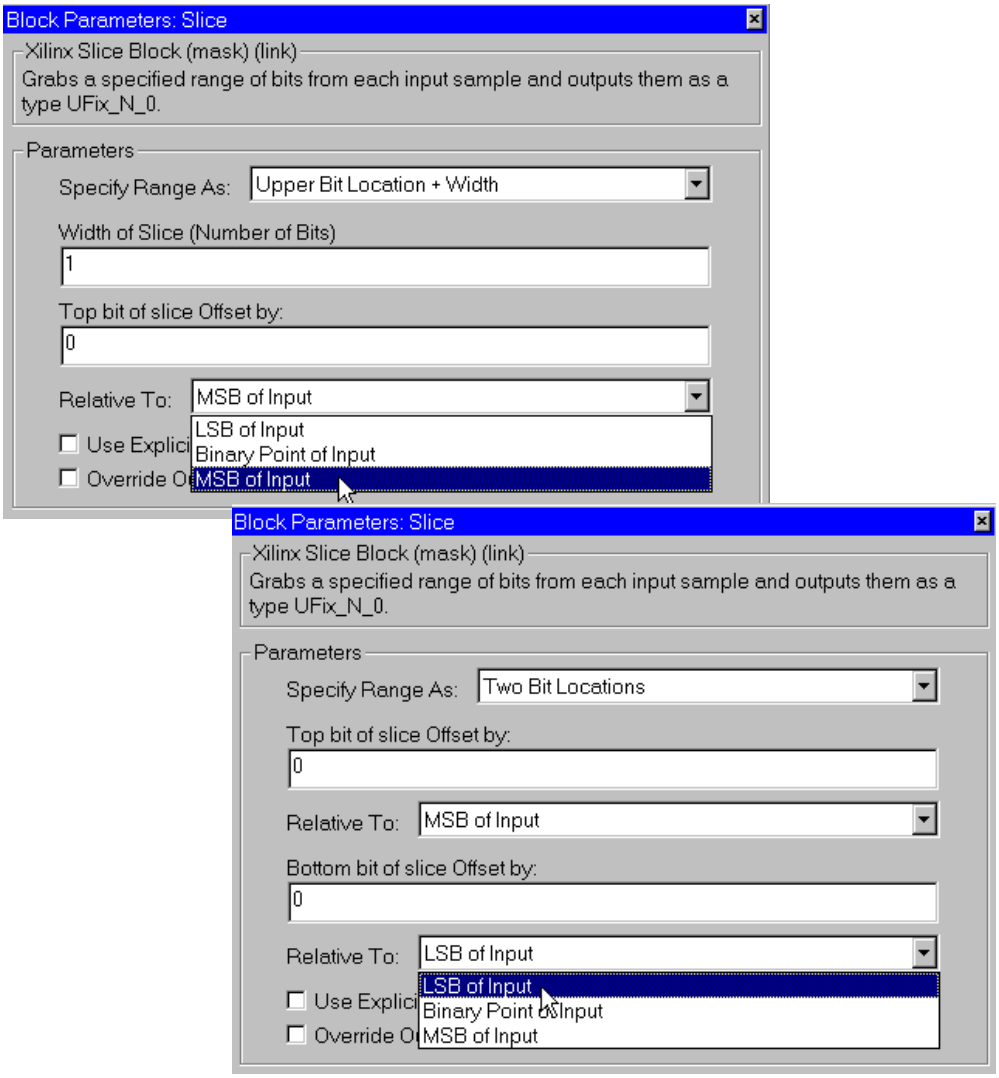


Figure: Slice block parameterization GUI showing different options

You may choose one of three ways to specify the range of bits to slice:

- Two Bit Locations: specify the starting bit position to extract, and the ending bit position.
- Upper Bit Location + width: specify the bit that you want as the upper bit of your output, and number of bits to extract.
- Lower Bit Location + width: specify the bit that you want to be the lower bit of your output, and number of bits to extract.

All bit slice positions are specified relative to the Most Significant Bit (MSB), Least Significant Bit (LSB), or Binary point of the top or bottom of the slice. The "Offset"values are always towards the MSB.

Other parameters used by this block are explained in the Common Parameters section of the previous chapter of the Reference Guide.

The Slice block does not use a Xilinx LogiCORE.

Up Sample

The Xilinx Up Sample block increases the sample rate at the point where the block is placed in your design. The input signal is over-sampled so that every nth input sample is presented at the output, or presented once with (n-1) zeroes interspersed.

The output sample period is (input sample period) / **k** where the ratio **k** is a configurable block parameter.

The Up Sample block may be configured via its parameterization GUI.

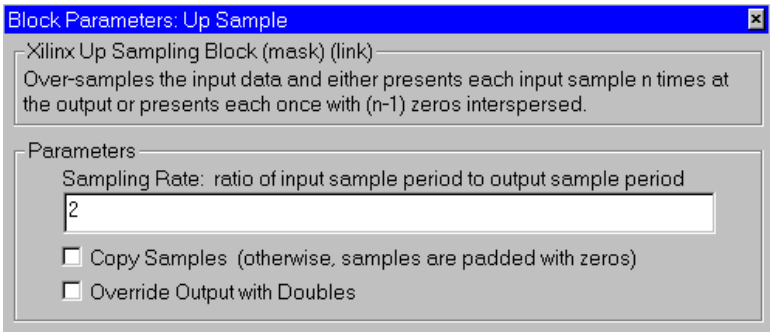


Figure: Up Sample block parameterization GUI

The *Sampling Rate* field must contain an integer, greater or equal to 2. This is the ratio of the output sample period to the input, and is essentially a sample rate multiplier. For example, a ratio of 2 indicates a 2*1 multiplier of the input sample rate. If a non-integer ratio is desired, the Up Sample block can be used in combination with the Down Sample block to create smaller differences in sample rate.

The *Copy Samples* checkbox lets you choose what to do with the additional samples produced by the increased clock rate. By selecting Copy Samples, the same sample will be duplicated (copied) during the extra sample times. If this checkbox is not selected, then this additional samples output will contain zeroes.

Other parameters used by this block are explained in the Common Parameters section of the previous chapter of the Reference Guide.

The Up Sample block does not use a Xilinx LogiCORE.

DSP

Blockset elements of Digital Signal Processing (DSP) specific functions.

FFT

The Xilinx FFT Block computes the Discrete Fourier Transform (DFT) using the radix-4 Cooley-Tukey algorithm, explained below:

The N-point DFT of a complex vector
x(n) = [x(0), x(1), ..., x(N-1)]
is the vector **X(k)** = [X(0), X(1), ..., X(N-1)],
where the k-th element:

$$X(k) = \sum_{m=0}^{N-1} x(m) W_N^{mk}$$

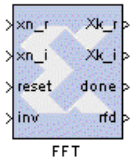
for k=0, 1, ... , N-1, where

$$W_N = e^{(-i)\frac{2\pi}{N}}$$

is a principal N-th root of unity.

The FFT block accepts as input a stream of complex data represented as a pair of Xilinx fixed-point data{x(0), x(1), ...}and computes successive DFTs of nonoverlapping frames of N data samples.

The Block Interface (inputs and outputs as seen on the FFT icon) are as follows:



Input signals:

xn_r	real component of input data stream
xi_r	imaginary component of input data stream
reset	reset signal
inv	0 for forward transform, 1 for inverse

Output signals:

Xk_r	real component of output data stream
Xk_i	imaginary component of output data stream
done	active high on first output sample in a frame
rfd	active high when block can accept input data

The FFT parameterization GUI is accessible by double-clicking the FFT icon on your Simulink model sheet.

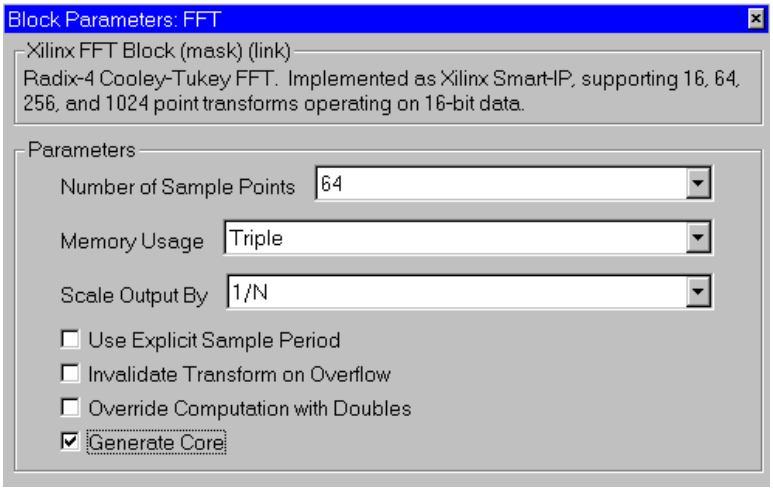


Figure: FFT block parameterization GUI

Parameters specific to the FFT block are:

- Number of points: you can choose 16, 64, 256, or 1024.
- Memory usage: choose Single, Double, Triple (16 pt: Single only).
- Output scaling: choose 1 / N or 1 / (2N).

- Overflow characteristic: you may choose to Invalidate transform (if checkbox selected) or to stop the simulation in the event of an overflow (if checkbox not selected).

Other parameters used by this block are explained in the Common Parameters section of the previous chapter of the Reference Guide.

Block Timing

Below is a timing diagram that illustrates the behavior of the FFT block indicating the number of sample periods between the taking of input samples and the production of the output samples for a particular frame. (Note that the timing characteristics depend on the number of points in the FFT and the memory usage mode selected. For Triple Memory configurations the timing numbers are specified in terms of the output data sample period.)

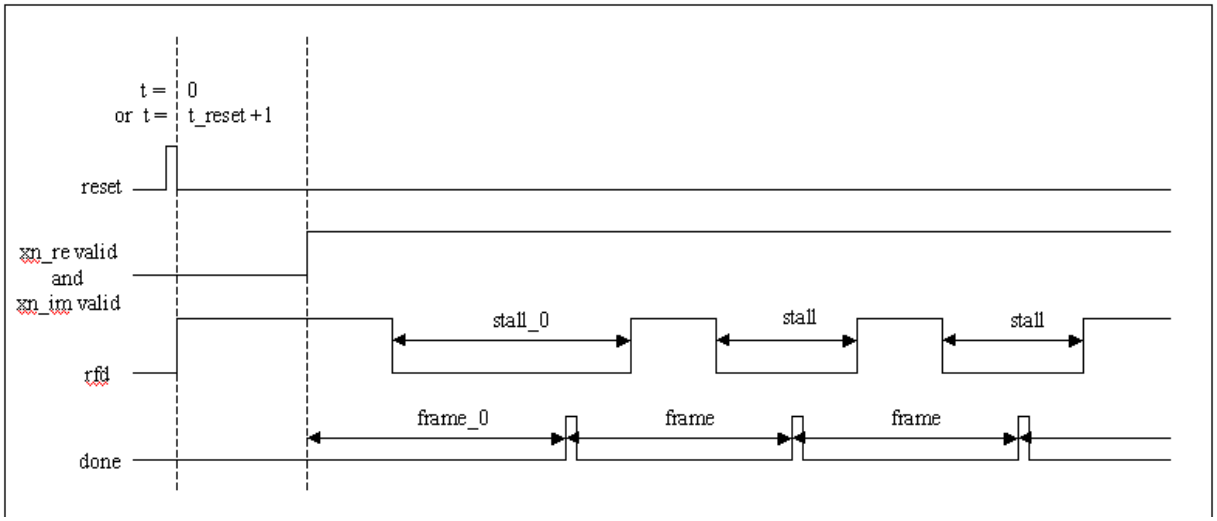


Figure: FFT Timing Diagram

	Single Memory	Double Memory	Triple Memory
64-point	stall_0 = 275 stall = 275 frame_0 = 277 frame = 339	stall_0 = 146 stall = 128 frame_0 = 276 frame = 192	stall_0 = 0 stall = 0 frame_0 = 406 frame = 192
256-point	stall_0 = 1074 stall = 1074 frame_0 = 1076 frame = 1330	stall_0 = 789 stall = 768 frame_0 = 1075 frame = 1024	stall_0 = 0 stall = 0 frame_0 = 1589 frame = 768
1024-point	stall_0 = 5170 stall = 5170 frame_0 = 5172 frame = 6194	stall_0 = 4117 stall = 4096 frame_0 = 5171 frame = 5120	stall_0 = 0 stall = 0 frame_0 = 8246 frame = 4096

Table: FFT Timing Characteristics

For the 16-point FFT, there is no stall time and the frame_0 time is 84 ticks and subsequent frames are delivered continuously, so frame = 16.

Xilinx LogiCORE

The Xilinx FFT block is implemented with the Xilinx FFT LogiCORE v1.0, consequently the number of points supported are N=16, 64, 256, or 1024. The 64, 256, and 1024 point FFTs also contain external memories implemented with the Dual Port Block Memory LogiCORE v1.0. The number of logical memory blocks (either 1, 2, or 3) determines the timing characteristics and size of the implementation. There is no synthesizable implementation. A realizable implementation requires 16-bit data inputs, although the block will simulate with other input sizes.

To help avoid overflow in the internal arithmetic calculations, the output is always scaled; you can choose whether to scale the output by 1/N or by 1/(2N). Scaling by 1/(2N) will always avoid overflow, but will limit your range.

Core operation

The timing characteristics of the FFT block depends on the number of points and block parameters. Synchronization is accomplished using the two output control signals, rfd (ready for data), and done (beginning of new output frame). The block accepts data only when the rfd output signal is active high, otherwise any input samples are ignored. The done output signal is asserted active high for the first sample of each output frame and is low otherwise. For example, the rfd signal can be used to drive the control port of an enabled subsystem that is feeding data to the FFT block.

The block initially wakes up in “reset” mode, in which it ignores invalid data samples. The first coincident valid xn_r and xn_i data is considered the beginning of a new input frame, and upon receipt, the block begins processing data. Asserting reset will immediately return the block to reset mode, interrupting the current output frame. The output data will remain invalid until the subsequent frame is processed as indicated by the done signal.

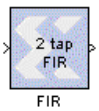
The four Xilinx LogiCORE datasheets (one for each of the 16, 64, 256, and 1024-point FFTs) may be found on your local disk at:

```
%XILINX%\coregen\ip\xilinx\vfft_v1_0\com\xilinx\ip\vfft\doc\c_
fft1024_v1_0.pdf
%XILINX%\coregen\ip\xilinx\vfft_v1_0\com\xilinx\ip\vfft\doc\c_
fft16_v1_0.pdf
%XILINX%\coregen\ip\xilinx\vfft_v1_0\com\xilinx\ip\vfft\doc\c_
fft256_v1_0.pdf
%XILINX%\coregen\ip\xilinx\vfft_v1_0\com\xilinx\ip\vfft\doc\c_
fft64_v1_0.pdf
```

The Dual Port Block Memory LogiCORE datasheet may be found on your local disk at:

%XILINX%\coregen\ip\xilinx\blkmemv2dp_v2_0\com\xilinx\ip\blkmemv2dp_v2_0\doc\BLKMEMV2DP_V2_0.pdf

FIR



The Xilinx FIR Filter Block implements a finite-impulse response (FIR) digital filter. An N-tap filter is defined by N filter coefficients, or taps, each represented as a Xilinx fixed-point number. h(i) are the set of user-defined coefficients.

The filter block accepts a stream of Xilinx fixed-point data samples x(0), x(1), ..., and at time n computes the output:

$$y(n) = \sum_{i=0}^{N-1} h(i)x(n-i)$$

The FIR block takes one input **xn**: a Xilinx Blockset signal fixed-point data sample.

The block produces one output signal **yn**: a Xilinx Blockset fixed-point output sample.

The FIR parameterization GUI may be accessed by double-clicking on the token on your Simulink model sheet:

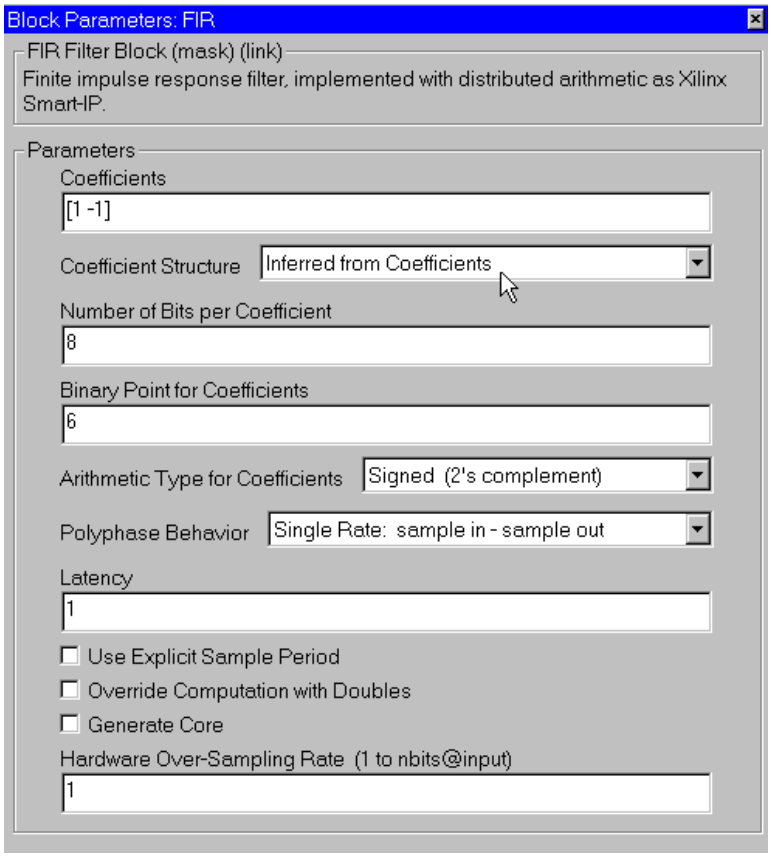


Figure: FIR block parameterization GUI

Parameters specific to this block are:

- Coefficents - vector of filter coefficients; note that these can be evaluated from a MATLAB workspace variable and may in turn be computed by a MATLAB tool (also see the Quickstart Guide and Tutorials examples).
- Coefficient structure - Xilinx Smart-IP core preferred implementation depends on the structure of the sequence of filter taps. You may choose one of: inferred from coefficients, none, symmetric, negative symmetric, half band, and interpolated fir.
- Number of bits per coefficient - Xilinx fixed-point parameter.
- Binary point for coefficients - Xilinx fixed-point parameter.
- Coefficient arithmetic type - Xilinx fixed-point parameter.

- Polyphase behavior - Decimation, Interpolation, Single rate.
- Latency - specify input sample period latency.
- Hardware over clocking - Hardware clocks per sample.

Other parameters used by this block are explained in the Common Parameters section of the previous chapter of the Reference Guide.

Xilinx LogiCORE

The block is implemented with the Xilinx Distributed Arithmetic FIR Filter v4.0 LogiCORE. (Version 4.0 of the LogiCORE is an update from version 3.0, which was supported in System Generator v1.0. If you have upgraded to the System Generator v1.0.1, you will need to install the newer version of this core, available online in the IP Update #2 from the Xilinx IP Center.)

The Simulink model operates on a sample in / sample out basis, but the core has the capability of using serial arithmetic by overclocking. Although this adds latency, it has the benefit of reducing the number of hardware resources required for the filter. Refer to the Core Data Sheet for more details of the filter modes and parameters. The datasheets for versions 3.0 and 4.0 may be found on your local disk at:

```
%XILINX%\coregen\ip\xilinx\da_fir_v3_0\com\xilinx\ip\da_fir_v3_0\doc\C_DA_FIR_V3_0.pdf
```

```
%XILINX%\coregen\ip\xilinx\da_fir_v3_0\com\xilinx\ip\da_fir_v3_0\doc\C_DA_FIR_V4_0.pdf
```

Math

The Math section of the Xilinx Blockset contains mathematical functions.

Accumulator

The Xilinx Accumulator block is an adder-based or a subtractor-based accumulator.

The block has a data input port **b**, a reset port **rst**, and an output port **q**. The output must have the same width as the input data. The output is produced as:

if rst=1, then q(n)=0;
if rst=0, then q(n) = q(n-1) * *FeedbackScaling* + / - b(n-1).

The Accumulator Block parameterization GUI can be invoked by double-clicking on the icon in your Simulink model:

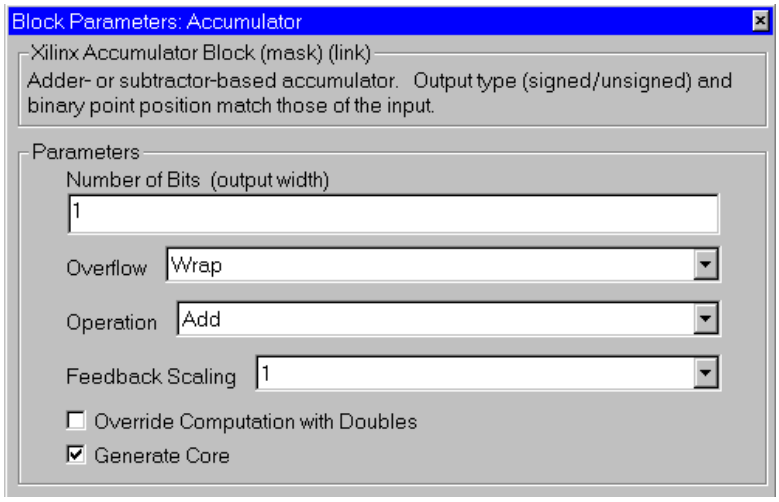


Figure: Accumulator block parameterization GUI

Parameters specific to the Accumulator block are:

- Number of Bits (output width): specifies the output width and must be the same as the input width
- Operation: a list of two choices, Add and Subtract. This determines if the block is an adder-based or subtractor-based accumulator.
- Feedback Scaling: there are nine choices of feedback multiplier: 1, 1/2, 1/4, 1/8, 1/16, 1/32, 1/64, 1/128, and 1/256.

The output is produced as:
if rst=1, then q(n)=0;
if rst=0 and the operation is Add, then q(n) = q(n-1) * *FeedbackScaling* + b(n-1);
if rst=0 and the operation is Subtract, then q(n) = q(n-1) * *FeedbackScaling* - b(n-1).

The arithmetic type of the output is the same as that of the input. The block always has a latency of 1.

Other parameters used by this block are explained in the Common Parameters section of the previous chapter of the Reference Guide.

Xilinx LogiCORE

The Accumulator block always uses the Xilinx LogiCore: Accumulator V2.0. The data width must be between 1 and 66, inclusive.

The Core datasheet may be found on your local disk at:

```
%XILINX%\coregen\ip\xilinx\baseblox_v2_0\com\xilinx\ip\baseblox_v2_0\doc\C_ACCUM_V2_0.pdf
```

AddSub

The Xilinx AddSub block performs an addition or a subtraction. The operation can be fixed, or can be specified dynamically through an optional mode port.

The AddSub block can be configured via the Block Parameters GUI:

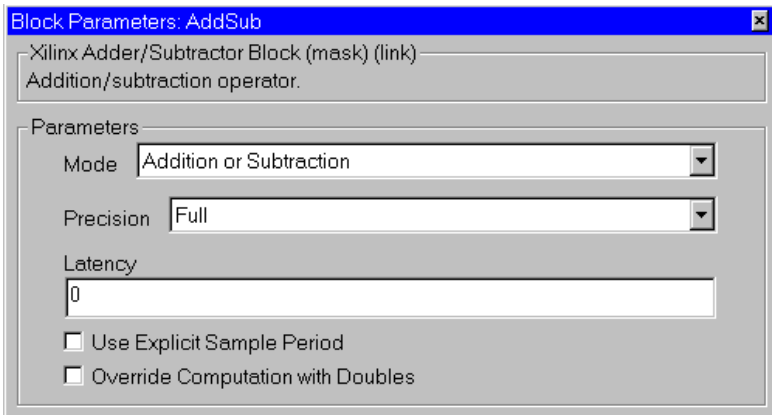


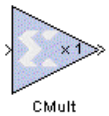
Figure: AddSub block parameterization GUI

The GUI contains one parameter which is specific to the AddSub block. The *Mode* parameter is a list of three choices: Addition, Subtraction, and Addition / Subtraction. When Addition is selected, the block is a two-input adder. When Subtraction is selected, the block is a two-input subtractor. When Addition / Subtraction is selected as the Mode parameter, the block is an Adder / Subtractor and the operation it will perform is determined by the third input port (titled *sub). The *sub input port is a 1-bit unsigned number. When it is 0, the block will perform an addition. Otherwise, it will perform a subtraction.

Other parameters used by this block are explained in the Common Parameters section of the previous chapter of the Reference Guide.

The AddSub block does not use a Xilinx LogiCORE.

CMult



The Xilinx CMult block multiplies an input by a constant. The constant is specified as one of the block parameters. The block has one input and one output.

The CMult block can be configured via the Block Parameters GUI:

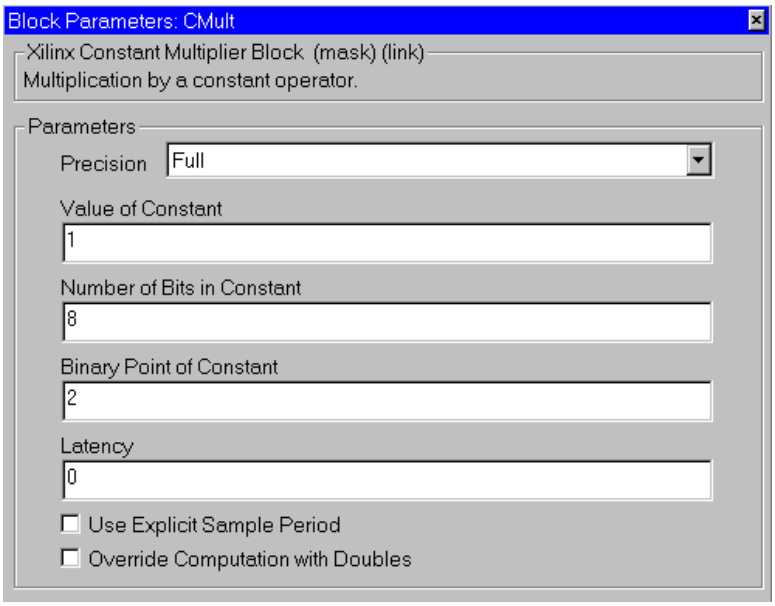


Figure: CMult block parameterization GUI

Parameters specific to the CMult block are:

- Value of Constant: The value entered as the Value of Constant parameter may not fit the container. In this case, System Generator rounds and saturates to force the constants into ints container. Whether the constant is unsigned or signed is determined automatically by the System Generator.
- Number of Bits in Constant - defines the width of the constant.
- Binary Point of Constant - defines the binary point of the constant.

Other parameters used by this block are explained in the Common Parameters section of the previous chapter of the Reference Guide.

The CMult block does not use a Xilinx LogiCORE.

Inverter

The Xilinx Inverter block calculates a bit-wise complement on its input value, which is presented as the output. The block has two implementations: a Xilinx LogiCORE version and a VHDL synthesized version.

The Inverter block can be configured via the Block Parameters GUI:

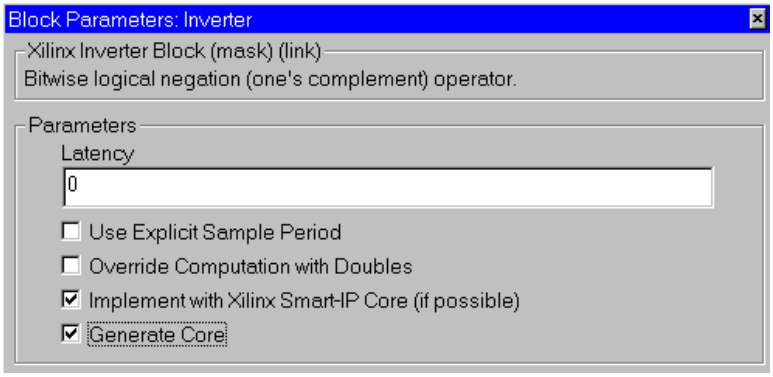


Figure: Inverter block parameterization GUI

Parameters used by this block are explained in the Common Parameters section of the previous chapter of the Reference Guide.

Xilinx LogiCORE

The Inverter uses the Xilinx LogiCORE Bus Gate V2.0 when possible. When the Use Explicit Sample Period parameter is checked, system generator will use a core if the input width is from 1 to 64, inclusive. Otherwise it will use the VHDL synthesized version.

The Core datasheet may be found on your local disk at:

```
%XILINX%\coregen\ip\xilinx\baseblox_v2_0\com\xilinx\ip\baseblox_v2_0\doc\C_GATE_BUS_V2_0.pdf
```

Logical

The Xilinx Logical block performs a bit-wise logical operation on 2, 3, or 4 numbers. It can have 2, 3, or 4 inputs and one output. Operands are aligned at the binary point, zero or sign extend appropriately, then the bit-wise logical operation is performed.

The block has two implementations: a Xilinx LogiCORE version and a VHDL synthesized version.

Note that in trees of logical gates, it is typically better to **not** map the blocks to cores as the synthesized versions can be combined, optimized, and trimmed by the design synthesis tools that run downstream.

The Logical block can be configured via the Block Parameters GUI:

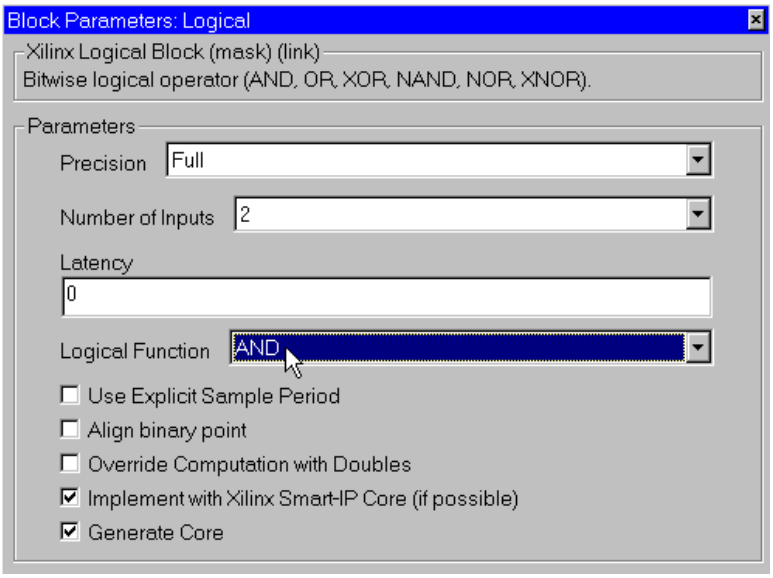


Figure: Logical block parameterization GUI

Parameters specific to the block are:

- Number of Inputs - a list of three choices: 2, 3, or 4.
- Logical Function - a list of six choices: AND, NAND, OR, NOR, XOR, and XNOR. This parameter specifies which logical operation the block will perform.
- Align Binary Point - a checkbox specifying whether the System Generator should align automatically. If this box is checked, numbers are automatically aligned at the binary point. Otherwise, all inputs of the Logical block must have the same binary point position.

Other parameters used by this block are explained in the Common Parameters section of the previous chapter of the Reference Guide.

Xilinx LogiCORE

A Logical block will use the Xilinx LogiCORE Bus Gate V2.0 if possible. When the Use Xilinx Smart-IP Core checkbox is selected, the System Generator will use a core if the *logical operation width* is between 1 and 64, inclusive. The logical operation width is the width after binary point alignment, and after appropriate sign or zero extension. If the operation width is greater than 64, a VHDL synthesized implementation will be used.

The Core datasheet may be found on your local disk at:

```
%XILINX%\coregen\ip\xilinx\baseblox_v2_0\com\xilinx\ip\baseblo
x_v2_0\doc\C_GATE_BUS_V2_0.pdf
```

Mult

A Xilinx Multiplier block performs a multiplication on two numbers. The block can have one of three implementations:

- a pipelined multiplier using a Xilinx LogiCORE

- a parallel multiplier using a Xilinx LogiCORE
- a parallel multiplier implemented in synthesizable VHDL

The Multiplier block can be configured via its Block Parameters GUI:

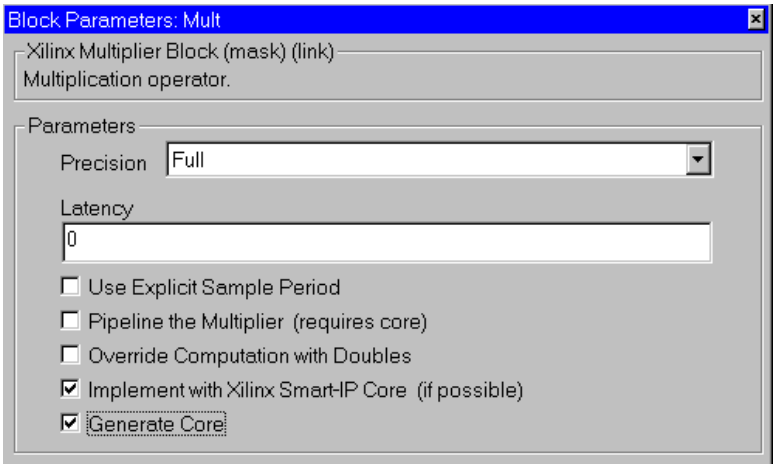


Figure: Multiplier block parameterization GUI

The pipeline option parameter is specific to the Multiplier block. This parameter indicates whether to use the pipelined version of a multiplier. When this is checked, the block latency and the input widths must fall within the capabilities of the Xilinx LogiCORE.

Other parameters used by this block are explained in the Common Parameters section of the previous chapter.

Xilinx LogiCORE

The Multiplier block uses the Xilinx LogiCORE Variable Parallel Multiplier V2.0. When the generate-core parameter is checked, the System Generator will generate a core if possible. Both input widths must be between 2 to 32 inclusive, if a Xilinx LogiCORE is to be used. If an input exceeds this range, then synthesizable VHDL will be used. When the Pipeline-the-Multiplier checkbox is selected, the Multiplier block latency must not be less than the latency of the core. The latency of the core is a function of the B input width as described in the following table:

B input width (number of bits)	Xilinx LogiCORE Latency (number of clocks)
3 and 4	1
5 to 8	2
9 to 16	3
17 to 32	4

Table: Multiplier latency

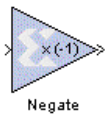
If the latency is greater than the latency of the Xilinx LogiCORE, registers are added after the core.

For pipelined implementation, the B input width must be greater than 2.

The Core datasheet for the variable parallel virtex multiplier may be found on your local disk at:

```
%XILINX%\coregen\ip\xilinx\mult_vgen_v2_0\com\xilinx\ip\mult_v
gen_v2_0\doc\mult_vgen_v2_0.pdf
```

Negate



The Xilinx Negate block negates its input. It has one input and one output. The block has two implementations: a Xilinx LogiCORE, and synthesizable VHDL.

The Negate block is configurable via its Block Parameters GUI:

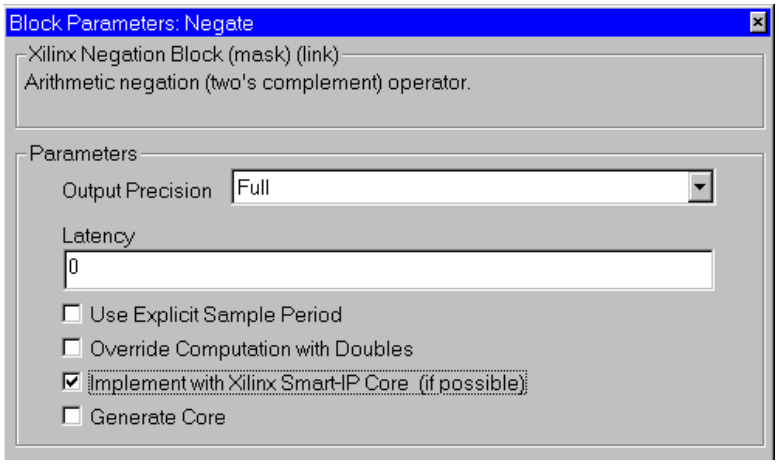


Figure: Negate block parameterization GUI

Parameters used by this block are explained in the Common Parameters section of the previous chapter of the Reference Guide.

Xilinx LogiCORE

A Negate block will use the Xilinx LogiCORE Twos Complementer V2.0 if possible. When the Use Xilinx Smart-IP Core checkbox is selected, the System Generator will use a core provided the input width is between 1 and 64, inclusive. Otherwise sysnthesizable VHDL will be used.

The Core datasheet may be found on your local disk at:

```
%XILINX%\coregen\ip\xilinx\baseblox_v2_0\com\xilinx\ip\baseblo
x_v2_0\doc\C_TWOS_COMP_V2_0.pdf
```

Relational

The Xilinx Relational block compares two numbers and produces the result of the comparison as its output. The six comparisons the block can perform are:

- equal-to (=),
- not-equal-to (!=),
- less-than (<),
- greater-than (>),

- less-than-or-equal-to (<=),
- greater-than-or-equal-to (>=)

The output of a Relational block is a 1-bit unsigned number. It is set to 1 if the comparison result is true and 0 if the result is false.

Parameters of a Relational block can be set through the parameterization GUI.

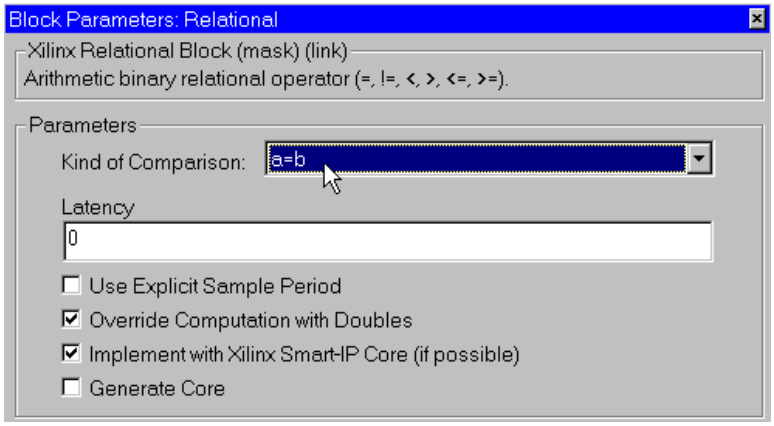


Figure: Relational block parameterization GUI

The comparison operation parameter is particular to this block. This parameter is a pulldown menu and can be set to one of the six comparisons a Relational block can perform. Other parameters used by this block are explained in the Common Parameters section of the previous chapter of the Reference Guide.

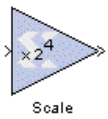
Xilinx LogiCORE

The inputs to the block must be between 1 and 64 bits wide, inclusive. If requested, the block will be implemented with the Xilinx LogiCORE: Comparator V2.0.

The Core datasheet may be found on your local disk at:

```
%XILINX%\coregen\ip\xilinx\baseblox_v2_0\com\xilinx\ip\baseblox_v2_0\doc\C_COMPARE_V2_0.pdf
```

Scale



The Xilinx Scale block scales its input by a power of two. The power can be either positive or negative. The block has one input and one output. The scale operation has the effect of moving the binary point without changing the bits in the container.

The Scale block may be configured via its parameterization GUI:

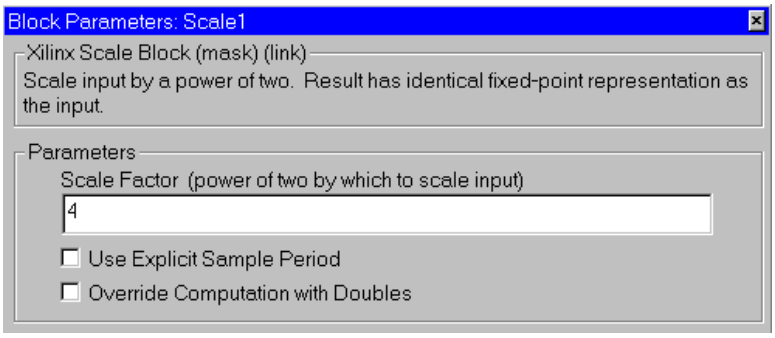


Figure: Scale block parameterization GUI

The only parameter that is specific to the Scale block is **Scale Factor**. It can be a positive or negative integer. The output of the block is $Output = Input \times 2^k$ where i is the input value and k is the scale factor. The effect of scaling is to move the binary point, which in hardware has **no** cost (a "shift," on the other hand, may add logic).

Other parameters used by this block are explained in the Common Parameters section of the previous chapter of the Reference Guide.

The Scale block does not use a Xilinx LogiCORE.

Shift

The Xilinx Shift block performs a left or right shift on the input number. The result of the shift will have the same fixed number container as that of the input number. The block has one input and one output.

The Shift block may be configured via its parameterization GUI:

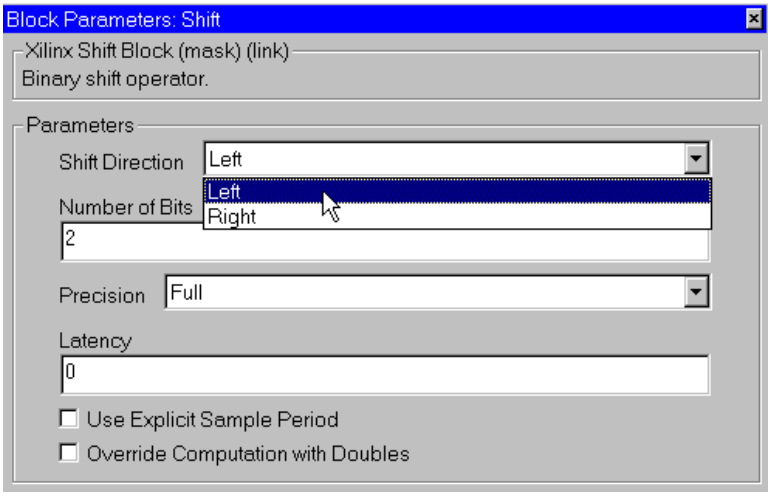


Figure: Shift block parameterization GUI

Parameters specific to the Shift block are those that specify how to do the shift operation.

- Shift Direction is a list with two choices: Left and Right. The **Left** shift operation shifts the input number to the left within its fixed number container . Appropriate sign extension will be performed. Bits that are shifted out of the container are discarded. The **Right** shift operation shifts the input number to the right within its

container. Zeroes will be appended to the least significant bits. Bits that are shifted out of the container are discarded.

- Number of Bits specifies how many bits are shifted. If the Number of Bits is a negative number, the block will shift in the opposite direction from the selected shift direction.

Other parameters used by this block are explained in the Common Parameters section of the previous chapter of the Reference Guide.

The Shift block does not use a Xilinx LogiCORE.

Threshold

The Xilinx Threshold block tests the sign of the input number. If the input number is negative, the output of the block is -1, otherwise the output is 1. The output is a signed fixed-point integer which is 2 bits long. The block has one input and one output.

The Threshold block may be configured via its parameterization GUI:

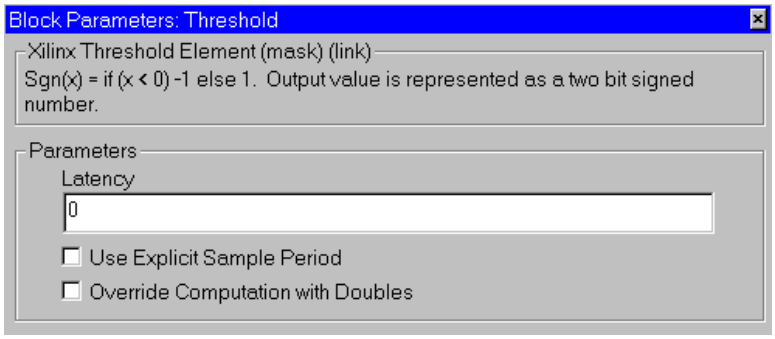


Figure: Threshold block parameterization GUI

The block parameters do not control the output data type because the output is always a signed fixed-point integer which is 2 bits long.

All the parameters used by this block are explained in the Common Parameters section of the previous chapter of the Reference Guide.

The Threshold block does not use a Xilinx LogiCORE.

MATLAB I/O

The MATLAB I/O section includes Xilinx Gateway blocks, blocks to report quantization error, the enabled subsystem gateway, and display elements.

Gateway Blocks

The Xilinx Gateway blocks have several functions:

- Convert data from double-precision floating-point to the System Generator fixed-point type and vice-versa during Simulink simulation.
- Define I/O ports for the top level of the HDL design generated by System Generator. A Gateway In block defines a top level input port, and a Gateway Out block defines a top level output port.
- Define testbench stimulus and predicted output files and HDL components in the generated HDL design when the System Generator “Create Testbench” option is

selected. In this case, during HDL code generation, Simulink simulation data is logged as logic vectors into a data file for each top level port defined by a Gateway block. An HDL component is inserted in the top level testbench for each top level port, which during HDL simulation, reads the data from the file and compares it to the internally calculated value.

Note - Gateway blocks ("gateway in," "gateway out," and "enable adapter" blocks) cannot be used within an enabled subsystem.

Gateway In

The Xilinx Gateway In block is the input into the Xilinx FPGA part of your Simulink design. It converts Simulink double precision input to the System Generator fixed point type, and defines an input port for the top level of the HDL design generated by System Generator.

The Gateway In block is configurable via its parameterization GUI:

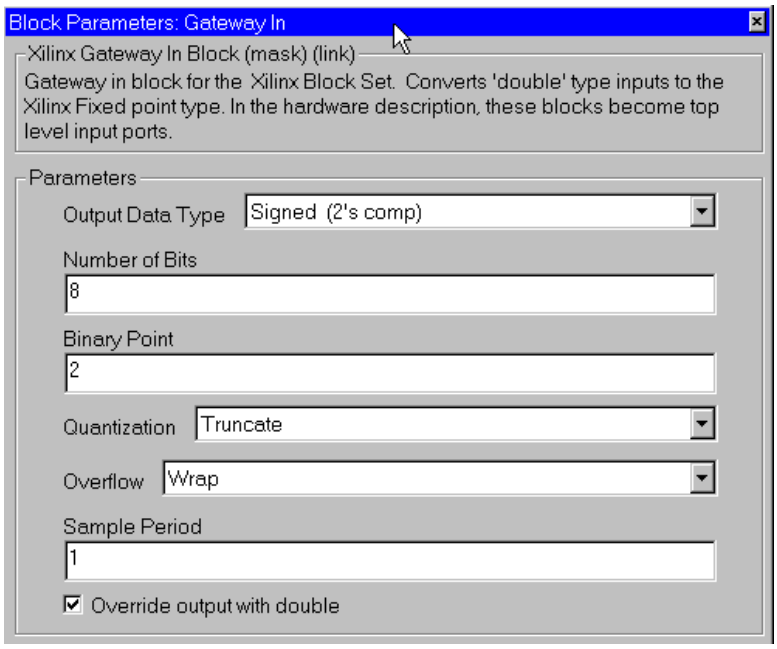


Figure: Gateway In block parameterization GUI

Parameters used by this block are explained in the Common Parameters section of the previous chapter of the Reference Guide.

Gateway Out

The Xilinx Gateway Out block is output from the Xilinx FPGA part of your Simulink design. It converts System Generator fixed point data to Simulink double precision, and defines an output port for the top-level of the HDL design generated by System Generator.

Enable Adapter

When using an enabled subsystem that contains Xilinx blocks, the enable port must be driven by a Xilinx Enable Adapter block. This block is a required interface to any enabled subsystem that contains a System

Generator block. The Enable Adapter block's output port must drive the subsystem's enable port.

Quantization Error blocks

Quantization error occurs in a design when your fixed-point data differs from the double-precision value in the least significant bits (LSB), i.e. in the fraction of your design. The System Generator tracks the degree of quantization error throughout the design by comparing the fixed-point value with the double-precision value. You can monitor the degree of quantization error by using the following blocks in your design.

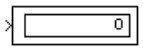
Quantization Error

The Xilinx Quantization Error block extracts the quantization error from a fixed-point signal. This error is tracked as the difference between the expected value (exact to machine precision) and the actual value of the fixed-point signal. You may view the quantization error by sending the output of the block into a display or scope.

Clear Quantization Error

The Clear Quantization Error block clears the quantization error tracking mechanism on a trace. Inserting this block has no effect on the computation other than the error analysis sections.

Display



This is the Simulink Display block, linked into the Xilinx Blockset's MATLAB I/O section as a convenience. It is presented as output to the Sample Time display (described below).

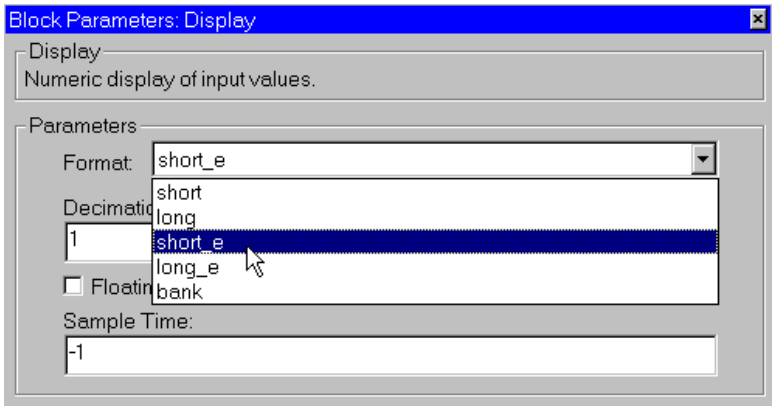


Figure: Display block parameterization GUI

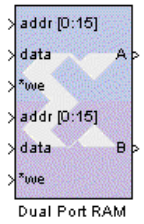
Sample Time

The Sample Time block reports the sample period of its input. It is meant to be displayed using the Display block, above.

Memory

This section contains Xilinx Blockset elements that use Xilinx memory LogiCOREs.

Dual Port RAM



The Xilinx Dual Port random access memory (RAM) has two independent sets of ports that allow read and write access. Data of a dual port RAM is stored by word, with all words having the same arithmetic type, width, and binary point position.

Each port set has one output port and three input ports for address, input data, and write enable (WE). Both data input ports must be of the same arithmetic type, width, and binary point position. The data out ports, labeled A and B, have the same types as the input data ports.

Each data word is associated with exactly one address. A valid address can be any unsigned fixed point integer from 0 to d-1, where d denotes the RAM depth (number of words in the RAM). An attempt to read past the end of the memory will be caught as an error in the simulation. The initial RAM contents can be specified through the block mask parameters.

The write enable port must be a 1-bit unsigned fixed point value with binary point zero. If the WE port is 1, the value on the data input is written to the memory contents indicated by the address input port. Write contention is an error caught during simulation. A checkbox in the GUI allows the user to specify that only valid data should be stored in the RAM. The RAM block has latency 1.

The Dual Port RAM block may be configured via its parameterization GUI:

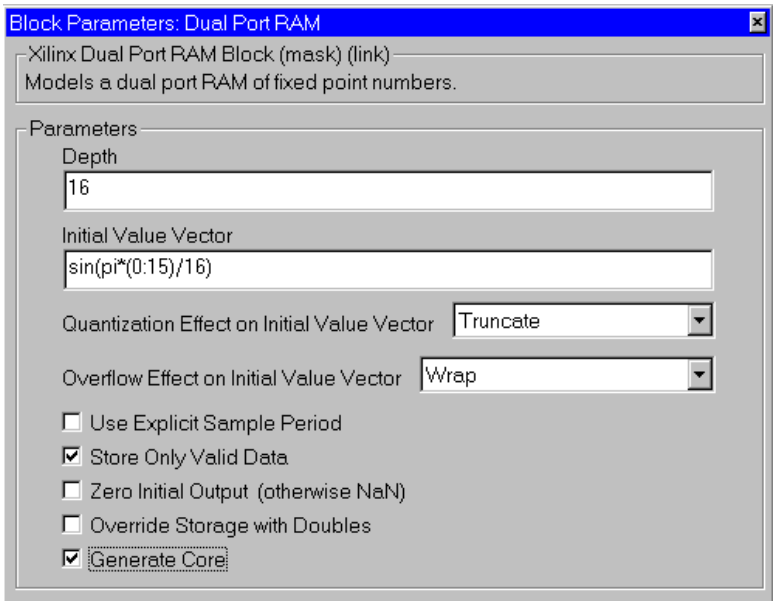


Figure: Dual Port RAM block parameterization GUI

Parameters specific to the block are:

- Depth: specifies the number of words in the block. The depth must be a positive integer.

- Initial Value Vector: specifies the initial value. When the vector is longer than the RAM, the vector's trailing elements are discarded. When the RAM is longer than the vector, the RAM's trailing words are set to zero.
- Store Only Valid Data: configures the block to accept only valid data.
- Zero Initial Output (otherwise NaN, "not a number"): if this box is checked, the data out ports have a value of zero at clock 0; otherwise the ports have a value of NaN.

Other parameters used by this block are explained in the Common Parameters section of the previous chapter of the Reference Guide.

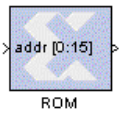
Xilinx LogiCORE

A Dual Port RAM block always uses the LogiCORE: Dual Port Block Memory V1.0. The memory depth must be from 16 to 2^{20}, the address width must equal $\lceil \log_2 \text{Depth} \rceil$, and the word width must be between 1 and 576.

The Core datasheet may be found on your local disk at:

```
%XILINX%\coregen\ip\xilinx\blockkmemex_v1_0\com\xilinx\ip\blockkmemex_v1_0\doc\c_mem_dp_block_v1_0.pdf
```

ROM



The Xilinx ROM block is a single port read-only memory. Data is stored by word and all words have the same arithmetic type, width, and binary point position.

Each word is associated with exactly one address. A valid address can be any unsigned fixed point integer from 0 to d-1, where d denotes the ROM depth (number of words in the ROM). Data in the ROM can only be read. The initial content of a ROM is specified through block mask parameters. The block has one input port for the memory address and one output port for the data out. The address port must be an unsigned fixed point integer. A ROM block has two possible implementations, either as Xilinx LogiCORE block memory or a Xilinx LogiCORE distributed memory.

The ROM block may be configured via its parameterization GUI:

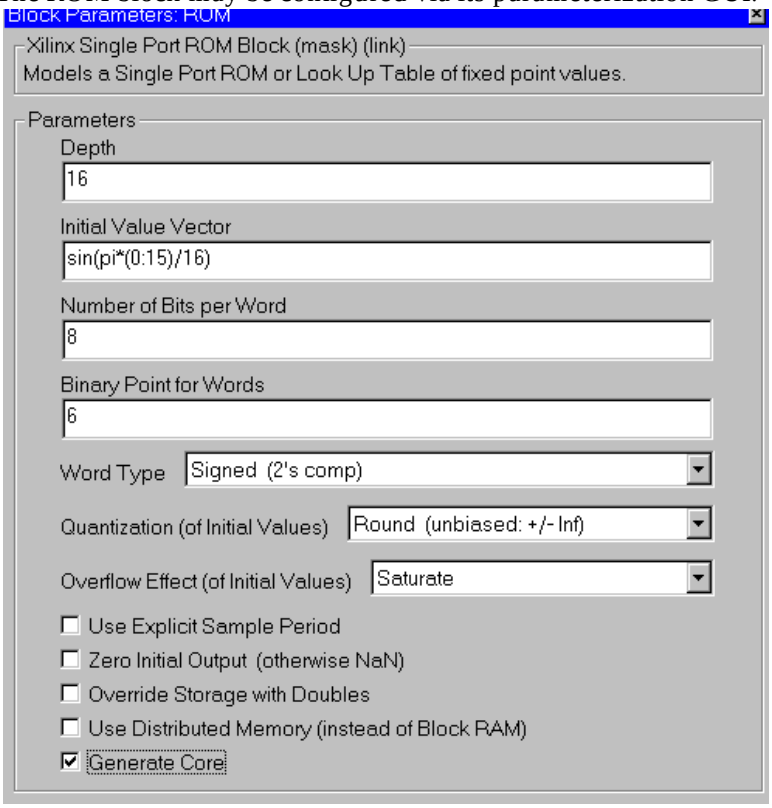


Figure: ROM block parameterization GUI

Parameters specific to this block are:

- Depth: specifies the number of words in the block. The depth must be positive.
- Initial Value Vector: specifies the initial value. When the vector is longer than the ROM, the vector's trailing elements are discarded. When the ROM is longer than the vector, the ROM's trailing words are set to zero.
- Zero Initial Output (otherwise NaN, "not a number"): if this box is checked, the data out ports have a value of zero at clock 0; otherwise the ports have a value of NaN.
- Use Distributed Memory: If this box is checked, the block is implemented with a Xilinx LogiCORE distributed memory. Otherwise, a block memory is used.

Other parameters used by this block are explained in the Common Parameters section of the previous chapter of the Reference Guide.

Xilinx LogiCORE

A ROM always uses a Xilinx LogiCORE. When the distributed memory parameter is not selected, Xilinx LogiCORE: Single Port Block Memory V1.0 is used. In this case, memory depth must be between 16 and 2^{20}. The word width must be between 1 and 576. When distributed memory parameter is selected, LogiCORE: Distributed Memory V2.0 is used. The memory depth must be between 16 and 256, and the word width must be between 1 and 256.

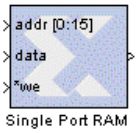
The Core datasheet for the Single Port Block Memory may be found on your local disk at:

```
%XILINX%\coregen\ip\xilinx\blockmemex_v1_0\com\xilinx\ip\block
memex_v1_0\doc\c_mem_sp_block_v1_0.pdf
```

The Core datasheet for the Distributed Memory may be found on your local disk at:

```
%XILINX%\coregen\ip\xilinx\baseblox_v2_0\com\xilinx\ip\baseblo
x_v2_0\doc\C_DIST_MEM_V2_0.pdf
```

Single Port RAM



The Xilinx Single Port RAM is a memory which can read / write one word at a time. All words have the same arithmetic type, width, and binary point position. The block has one output port and three input ports for address, input data, and write enable (WE).

Each data word is associated with exactly one address. A valid address can be any unsigned fixed point integer from 0 to d-1, where d denotes the RAM depth (number of words in the RAM). An attempt to read past the end of the memory will be caught as an error in the simulation. The initial RAM contents can be specified through the block mask parameters.

The write enable port must be a 1-bit unsigned fixed point integer. If the WE port is 1, the value on the data input is written to the memory contents indicated by the address input port. A checkbox in the GUI allows the user to specify that only valid data should be stored in the RAM. The RAM block has latency 1.

The output has the same type as the data in. Regardless of the WE value, the data out port always has the value at the location specified by the address line.

A Single port RAM block has two possible implementations, either as Xilinx LogiCORE block memory or a Xilinx LogiCORE distributed memory.

The RAM block may be configured via its parameterization GUI:

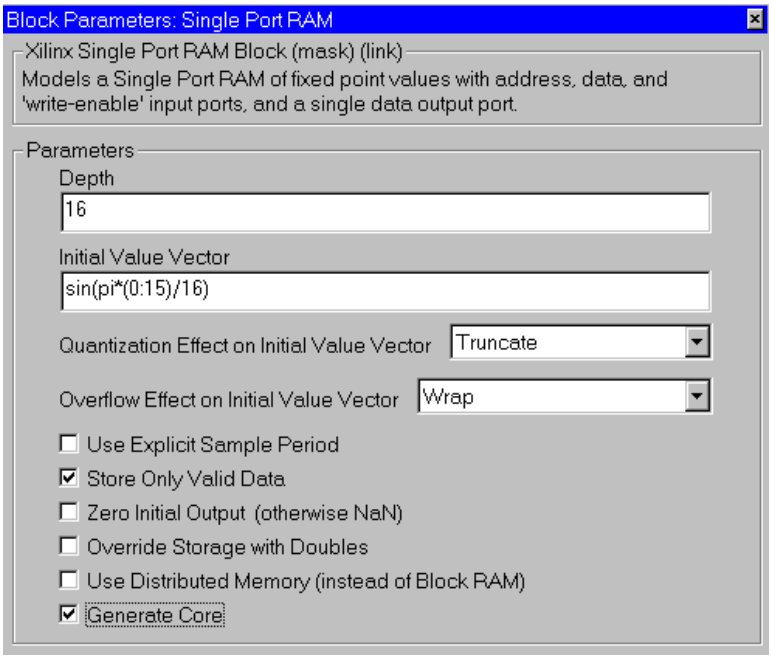


Figure: Single Port RAM block parameterization GUI

Parameters specific to this block are:

- Depth: specifies the number of words stored; must be a positive integer.
- Initial Value Vector: specifies the initial value. When the vector is longer than the RAM, the vector's trailing elements are discarded. When the RAM is longer than the vector, the RAM's trailing words are set to zero.
- Store Only Valid Data: configures the block to accept only valid data.
- Zero Initial Output (otherwise NaN, "not a number"): if this box is checked, the data out ports have a value of zero at clock 0; otherwise the ports have a value of NaN.
- Use Distributed Memory: If this box is checked, the block is implemented with a Xilinx LogiCORE distributed memory. Otherwise, a block memory is used.

Other parameters used by this block are explained in the Common Parameters section of the previous chapter of the Reference Guide.

Xilinx LogiCORE

A RAM block must use a Xilinx LogiCORE. When distributed memory is not selected, a Xilinx LogiCORE Single Port Block Memory V1.0 is used. The memory depth must be between 16 and 2^{20}. The word width must be between 1 and 576. When distributed memory is selected, a Xilinx LogiCORE Distributed Memory V2.0 is used. The memory depth must be between 16 and 256. The word width is between 1 and 256.

The Core datasheet for the Single Port Block Memory may be found locally at:

```
%XILINX%\coregen\ip\xilinx\blkmemex_v1_0\com\xilinx\ip\blockmemex_v1_0\doc\c_mem_sp_block_v1_0.pdf
```

The Core datasheet for the Distributed Memory may be found on your local disk at:

```
%XILINX%\coregen\ip\xilinx\baseblox_v2_0\com\xilinx\ip\baseblox_v2_0\doc\C_DIST_MEM_V2_0.pdf
```