Simulink Fixed Point Release Notes

Note The Fixed-Point Blockset has been replaced by two new products, Fixed-Point Toolbox and Simulink Fixed Point. This product restructuring reflects the broad expansion of fixed-point capabilities in MATLAB and Simulink. The Fixed-Point Toolbox introduces fixed-point operations to the MATLAB language, and Simulink Fixed Point enables fixed-point capabilities across much of the Simulink product family.

The "Simulink Fixed Point 5.0 Release Notes" on page 1-1 describe the changes introduced in Simulink Fixed-Point 5.0. The following topics are discussed in these Release Notes:

- "New Features" on page 1-2
- "Major Bug Fixes" on page 1-14

The Simulink Fixed Point Release Notes also provide information about recent versions of the Fixed-Point Blockset, in case you are upgrading from a version that was released prior to Release 13 with Service Pack 1.

- "Fixed-Point Blockset 4.0.1 Release Notes" on page 2-1
- "Fixed-Point Blockset 4.0 Release Notes" on page 3-1

Printing the Release Notes

If you would like to print the Release Notes, you can link to a PDF version.

Contents

1 Simulink Fixed Point 5.0 Release Notes

2 Fixed-Point Blockset 4.0.1 Release Notes

3 Fixed-Point Blockset 4.0 Release Notes

1

Simulink Fixed Point 5.0 Release Notes

New Features

This section introduces the new features and enhancements introduced in Simulink Fixed Point 5.0 since Version 4.1 of the Fixed-Point Blockset.

Product Restructuring

The Fixed-Point Blockset has been replaced by two new products, Fixed-Point Toolbox and Simulink Fixed Point. This product restructuring reflects the broad expansion of fixed-point capabilities in MATLAB and Simulink. The Fixed-Point Toolbox introduces fixed-point operations to the MATLAB language, and Simulink Fixed Point enables fixed-point capabilities across much of the Simulink product family.

Simulink Fixed Point requires Fixed-Point Toolbox. If you are on maintenance, you will automatically receive both of these new products in place of the Fixed-Point Blockset.

Fixed-Point Blocks Fully Integrated into Simulink

All former Fixed-Point Blockset blocks have been moved into the Simulink block libraries with this release. Each of these blocks, as well as all other Simulink blocks, can be used with or without Simulink Fixed Point installed. You can share models with any fixed-point and floating-point Simulink blocks among the users in your organization, whether or not they have Simulink Fixed Point installed. However, Simulink Fixed Point is required to take full advantage of the fixed-point features of Simulink blocks. For more information, refer to "Sharing Fixed-Point Models" in the Simulink Fixed Point documentation.

The following table lists all of the blocks in the Fixed-Point Blockset as of Release 13. It tells you the current name of the block in Simulink and the

Simulink library in which you can find the block. Most blocks have the same name as in the last release, however some block names have changed.

Former Fixed-Point Blockset block	Former Fixed-Point Blockset Library	Simulink Block	Simulink Library
Abs	Math	Abs	Math Operations
Accumulator	Calculus	Discrete-Time Integrator	Discrete
Accumulator Resettable	Calculus	Discrete-Time Integrator	Discrete
Accumulator Resettable Limited	Calculus	Discrete-Time Integrator	Discrete
Add	Math	Add	Math Operations
Bit Clear	Bits	Bit Clear	Logic and Bit Operations
Bit Set	Bits	Bit Set	Logic and Bit Operations
Bitwise Operator	Bits	Bitwise Operator	Logic and Bit Operations
Compare To Constant	Logic & Comparison	Compare To Constant	Logic and Bit Operations
Compare To Zero	Logic & Comparison	Compare To Zero	Logic and Bit Operations
Constant	Sources	Constant	Sources
Conversion	Data Type	Data Type Conversion	Signal Attributes
Conversion Inherited	Data Type	Data Type Conversion Inherited	Signal Attributes
Cosine	Lookup	Cosine	Lookup Tables

Former Fixed-Point Blockset block	Former Fixed-Point Blockset Library	Simulink Block	Simulink Library
Counter Free	Sources	Counter Free-Running	Sources
Counter Limited	Sources	Counter Limited	Sources
Data Type Duplicate	Data Type	Data Type Duplicate	Signal Attributes
Data Type Propagation	Data Type	Data Type Propagation	Signal Attributes
Dead Zone	Nonlinear	Dead Zone	Discontinuities
Dead Zone Dynamic	Nonlinear	Dead Zone Dynamic	Discontinuities
Decrement Real World	Math	Decrement Real World	Additional Math & Discrete / Additional Math: Increment - Decrement
Decrement Stored Integer	Math	Decrement Stored Integer	Additional Math & Discrete / Additional Math: Increment - Decrement
Decrement Time To Zero	Math	Decrement Time To Zero	Additional Math & Discrete / Additional Math: Increment - Decrement
Decrement To Zero	Math	Decrement To Zero	Additional Math & Discrete / Additional Math: Increment - Decrement
Derivative	Calculus	Discrete Derivative	Discrete
Detect Change	Edge Detect	Detect Change	Logic and Bit Operations

Former Fixed-Point Blockset block	Former Fixed-Point Blockset Library	Simulink Block	Simulink Library
Detect Decrease	Edge Detect	Detect Decrease	Logic and Bit Operations
Detect Fall Negative	Edge Detect	Detect Fall Negative	Logic and Bit Operations
Detect Fall Nonpositive	Edge Detect	Detect Fall Nonpositive	Logic and Bit Operations
Detect Increase	Edge Detect	Detect Increase	Logic and Bit Operations
Detect Rise Nonnegative	Edge Detect	Detect Rise Nonnegative	Logic and Bit Operations
Detect Rise Positive	Edge Detect	Detect Rise Positive	Logic and Bit Operations
Difference	Calculus	Difference	Discrete
Divide	Math	Divide	Math Operations
Dot Product	Math	Dot Product	Math Operations
Filter Direct Form I	Filters	This block is obsolete.	
Filter Direct Form I Time Varying	Filters	This block is obsolete.	
Filter Direct Form II	Filters	Transfer Fcn Direct Form II	Additional Math & Discrete / Additional Discrete
Filter Direct Form II Time Varying	Filters	Transfer Fcn Direct Form II Time Varying	Additional Math & Discrete / Additional Discrete
Filter First Order	Filters	Transfer Fcn First Order	Discrete

Former Fixed-Point Blockset block	Former Fixed-Point Blockset Library	Simulink Block	Simulink Library
Filter Lead or Lag	Filters	Transfer Fcn Lead or Lag	Discrete
Filter Real Zero	Filters	Transfer Fcn Real Zero	Discrete
FIR	Filters	Weighted Moving Average	Discrete
Gain	Math	Gain	Math Operations
Gateway In	Data Type	Data Type Conversion	Signal Attributes
Gateway In Inherited	Data Type	Data Type Conversion Inherited	Signal Attributes
Gateway Out	Data Type	Data Type Conversion	Signal Attributes
Increment Real World	Math	Increment Real World	Additional Math & Discrete / Additional Math: Increment - Decrement
Increment Stored Integer	Math	Increment Stored Integer	Additional Math & Discrete / Additional Math: Increment - Decrement
Index Vector	Select	Index Vector	Signal Routing
Integer Delay	Delays & Holds	Integer Delay	Discrete
Integrator Backward	Calculus	Discrete-Time Integrator	Discrete
Integrator Backward Resettable	Calculus	Discrete-Time Integrator	Discrete

Former Fixed-Point Blockset block	Former Fixed-Point Blockset Library	Simulink Block	Simulink Library
Integrator Backward Resettable Limited	Calculus	Discrete-Time Integrator	Discrete
Integrator Forward	Calculus	Discrete-Time Integrator	Discrete
Integrator Forward Resettable	Calculus	Discrete-Time Integrator	Discrete
Integrator Forward Resettable Limited	Calculus	Discrete-Time Integrator	Discrete
Integrator Trapezoidal	Calculus	Discrete-Time Integrator	Discrete
Integrator Trapezoidal Resettable	Calculus	Discrete-Time Integrator	Discrete
Integrator Trapezoidal Resettable Limited	Calculus	Discrete-Time Integrator	Discrete
Interval Test	Logic & Comparison	Interval Test	Logic and Bit Operations
Interval Test Dynamic	Logic & Comparison	Interval Test Dynamic	Logic and Bit Operations
Logical Operator	Logic & Comparison	Logical Operator	Logic and Bit Operations
Lookup Table	Lookup	Lookup Table	Lookup Tables
Lookup Table Dynamic	Lookup	Lookup Table Dynamic	Lookup Tables
Lookup Table (2-D)	Lookup	Lookup Table (2-D)	Lookup Tables
Matrix Gain	Math	Gain	Math Operations

Former Fixed-Point Blockset block	Former Fixed-Point Blockset Library	Simulink Block	Simulink Library
MinMax	Math	MinMax	Math Operations
MinMax Running Resettable	Math	MinMax Running Resettable	Math Operations
Multiply	Math	Product	Math Operations
Multiply Matrix	Math	Product	Math Operations
Multi-Port Switch	Select	Multiport Switch	Signal Routing
Product	Math	Product	Math Operations
Product of Elements	Math	Product of Elements	Math Operations
Product of Elements Inverted	Math	Product of Elements	Math Operations
Rate Limiter	Nonlinear	Rate Limiter	Discontinuities
Rate Limiter Dynamic	Nonlinear	Rate Limiter Dynamic	Discontinuities
Relational Operator	Logic & Comparison	Relational Operator	Logic and Bit Operations
Relay	Nonlinear	Relay	Discontinuities
Repeating Sequence Interpolated	Sources	Repeating Sequence Interpolated	Sources
Repeating Sequence Stair	Sources	Repeating Sequence Stair	Sources
Sample Rate Probe	Calculus	Weighted Sample Time	Signal Attributes
Sample Time Add	Calculus	Weighted Sample Time	Signal Attributes

Former Fixed-Point Blockset block	Former Fixed-Point Blockset Library	Simulink Block	Simulink Library
Sample Time Divide	Calculus	Weighted Sample Time	Signal Attributes
Sample Time Multiply	Calculus	Weighted Sample Time	Signal Attributes
Sample Time Probe	Calculus	Weighted Sample Time	Signal Attributes
Sample Time Subtract	Calculus	Weighted Sample Time	Signal Attributes
Saturation	Nonlinear	Saturation	Discontinuities
Saturation Dynamic	Nonlinear	Saturation Dynamic	Discontinuities
Scaling Strip	Data Type	Data Type Scaling Strip	Signal Attributes
Shift Arithmetic	Bits	Shift Arithmetic	Logic and Bit Operations
Sign	Nonlinear	Sign	Math Operations
Sine	Lookup	Sine	Lookup Tables
State-Space	Filters	Fixed-Point State-Space	Additional Math & Discrete / Additional Discrete
Subtract	Math	Subtract	Math Operations
Sum	Math	Sum	Math Operations
Sum of Elements	Math	Sum of Elements	Math Operations
Sum of Elements Negated	Math	Sum of Elements	Math Operations
Switch	Select	Switch	Signal Routing

Former Fixed-Point Blockset block	Former Fixed-Point Blockset Library	Simulink Block	Simulink Library
Tapped Delay	Delays & Holds	Tapped Delay	Discrete
Unary Minus	Math	Unary Minus	Math Operations
Unit Delay	Delays & Holds	Unit Delay	Discrete
Unit Delay Enabled	Delays & Holds	Unit Delay Enabled	Additional Math & Discrete / Additional Discrete
Unit Delay Enabled External IC	Delays & Holds	Unit Delay Enabled External IC	Additional Math & Discrete / Additional Discrete
Unit Delay Enabled Resettable	Delays & Holds	Unit Delay Enabled Resettable	Additional Math & Discrete / Additional Discrete
Unit Delay Enabled Resettable External IC	Delays & Holds	Unit Delay Enabled Resettable External IC	Additional Math & Discrete / Additional Discrete
Unit Delay External IC	Delays & Holds	Unit Delay External IC	Additional Math & Discrete / Additional Discrete
Unit Delay Resettable	Delays & Holds	Unit Delay Resettable	Additional Math & Discrete / Additional Discrete
Unit Delay Resettable External IC	Delays & Holds	Unit Delay Resettable External IC	Additional Math & Discrete / Additional Discrete
Unit Delay With Preview Enabled	Delays & Holds	Unit Delay With Preview Enabled	Additional Math & Discrete / Additional Discrete

Former Fixed-Point Blockset block	Former Fixed-Point Blockset Library	Simulink Block	Simulink Library
Unit Delay With Preview Enabled Resettable	Delays & Holds	Unit Delay With Preview Enabled Resettable	Additional Math & Discrete / Additional Discrete
Unit Delay With Preview Enabled Resettable External RV	Delays & Holds	Unit Delay With Preview Enabled Resettable External RV	Additional Math & Discrete / Additional Discrete
Unit Delay With Preview Resettable	Delays & Holds	Unit Delay With Preview Resettable	Additional Math & Discrete / Additional Discrete
Unit Delay Preview Resettable External RV	Delays & Holds	Unit Delay With Preview Resettable External RV	Additional Math & Discrete / Additional Discrete
Wrap To Zero	Nonlinear	Wrap To Zero	Discontinuities
Zero-Order Hold	Delays & Holds	Zero-Order Hold	Discrete

API for User-Written Fixed-Point S-Functions

You can now write your own Simulink C S-functions that directly handle fixed-point data types with a newly published API. For more information, refer to “Writing Fixed-Point S-Functions” in the Simulink Fixed Point documentation.

Fixed-Point Advisor Wizard

Simulink Fixed Point now includes a Model Advisor to help you to configure your fixed-point models to achieve a more efficient design and optimize your generated code. To use the Model Advisor to check your fixed-point models:

1 Select **Model Advisor** from the **Tools** menu of the model you wish to analyze. The Model Advisor appears in the Documents window on the MATLAB desktop.

2 Click **Select All** to enable all Model Advisor checks. For fixed-point code generation, the most important check boxes to select are **Identify questionable fixed-point operations**, **Identify blocks that generate expensive saturation and rounding code**, and **Check the Hardware Implementation**.

3 Click **Check Model**. Any tips for improving the efficiency of your fixed-point model appear in the browser.

Arithmetic with Non-Zero Bias Fully Supported

Code generation has been enhanced to generate bit-true fixed-point code that supports multiplication, division, and reciprocal for signal and parameters with non-zero bias. Previously, these cases lead to code generation errors. Code will now be generated for these cases, and that code will make efficient use of just C integer operations.

Generated Code for Lookup Tables Uses Less ROM

In prior releases, the size of the generated code for models that contained lookup tables with similar attributes was larger than necessary. Such lookup tables produced similar algorithms that appeared throughout the code multiple times. In this release, some common algorithms have been placed into functions which are called by the lookup tables. This enables the same code to be reused multiple times. The overall size of the generated code has been reduced through this enhancement.

Functions Moved to Simulink

The following former Fixed-Point Blockset functions are now installed with Simulink:

- `fixptbestexp`
- `fixptbestprec`
- `fixpt_interp1`
- `fixpt_look1_func_approx`
- `fixpt_look1_func_plot`
- `fixpt_set_all`
- `float`
- `fxptdlg`

- num2fixpt
- sfix
- sfrac
- sint
- ufix
- ufrac
- uint

Obsolete Functions

The functions fixpt_restore_links and fpupdate are obsolete. These functions are no longer needed to update models.

Major Bug Fixes

The Fixed-Point Blockset 5.0 includes several bug fixes made since Version 4.1. This section describes the particularly important Version 5.0 bug fixes.

Simulation Error for 65-Bit+ Multiplication Corrected

In prior releases, fixed-point multiplication could produce the wrong answer under certain simulation conditions. For this error to occur, one input had to have at least 33 bits and the other input at least 32 bits. The correct answer had to be negative, and some additional numerical criteria had to be met. This error could only occur in simulation; it never occurred in generated code. This error has been fully corrected for this release.

Fixed-Point Settings Interface Usable for Large Fonts

In the previous release, the Fixed-Point Settings interface was unusable if your system setup defined large default system fonts. When trying to open the dialog, an error would be reported and the dialog would not appear. The creation of the dialog has now been made robust to large fonts, and this problem is solved.

Lookup Table (2-D) Code Generation Bug Fixed

In a previous release, code generation would error out for the Lookup Table (2-D) block if the input data type had non-zero bias or non-one fractional slope, and the corresponding breakpoints were evenly spaced. This problem has been fixed.

2

Fixed-Point Blockset 4.0.1 Release Notes

Major Bug Fixes

The Fixed-Point Blockset 4.0.1 includes several important bug fixes made since Version 4.0.

If you are viewing these Release Notes in PDF form, please refer to the HTML form of the Release Notes, using either the Help browser or the MathWorks Web site and use the link provided.

Upgrading from an Earlier Release

Below is an upgrade issue involved in upgrading from the Fixed-Point Blockset 4.0 to Version 4.0.1.

If you are upgrading from a version earlier than 4.0, then you should see "Upgrading from an Earlier Release" on page 3-6 in the Fixed-Point Blockset 4.0 Release Notes.

Backwards Compatibility of Tunable Parameters for Unified Fixed-Point Blocks

Unified fixed-point blocks with tunable parameters have compatibility problems under certain conditions in Release 13. The problem arises only if a tunable parameter is mapped to a built-in integer or `single` data type. When tunable parameters are mapped to built-in integers or `single`, the code generated by Real Time Workshop will be different for unified blocks than it was for Fixed-Point Blockset blocks in prior releases. There are no compatibility problems if a tunable parameter maps to a nonbuilt-in data type, such as a scaled fixed-point integer.

Tunable parameters are entered in a Simulink model by specifying the name of a MATLAB variable in a block's dialog. This variable can be either a plain MATLAB variable or a Simulink parameter object. In either case, a numerical value will be defined for this tunable parameter by doing an assignment in MATLAB. MATLAB supports several numerical data types including the eight Simulink built-in numerical data types: `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, and `uint32`. One of these eight data types can be used when a value is defined for a MATLAB variable. The effect of the data type of the MATLAB variable is significantly different depending on how the tunable parameter is used in Simulink.

For Simulink built-in blocks, the legacy rule is to fully respect the data type used for the value of a MATLAB variable. Whatever data type is used in MATLAB when assigning a value to a variable is also be used when declaring that parameter in code generated by Real Time Workshop. The use of that parameter by a block may require the value to be represented using a different data type. If so, additional code is generated to convert the parameter every time it is used by the block. To get the most efficient code for a given block, the value of the MATLAB variable should use the same data type as is needed by the block.

For Fixed-Point Blockset blocks, the legacy rule is to expect no data type information from the MATLAB variable used for the tunable parameter. A fundamental reason for this is that MATLAB does not have native support for fixed-point data types and scaling, so the Simulink built-in legacy rule could not be directly extended to the general fixed-point case. Many fixed-point blocks automatically determine the data type and scaling for parameters based on what leads to the most efficient implementation of a given block. However, certain blocks such as Constant, as well as blocks that use tunable parameters in multiplication, do not imply a unique best choice for the data type and scaling of the parameter. These blocks have provided separate parameters on their dialogs for entering this information.

In Release 13, many Simulink built-in blocks and Fixed-Point Blockset blocks were unified. The Saturation block is an example of a unified block. The Saturation block appears in both the Simulink Library and in the Fixed-Point Blockset Library, but regardless of where it appears it has identical behavior. This identical unified behavior includes the treatment of tunable parameters. The dissimilarity of the legacy rules for tunable parameters has lead to a shortcoming in the unified blocks. Unified blocks obey the Simulink legacy rule sometimes and the Fixed-Point Blockset legacy rule at other times. If the block is using the parameter with built-in Simulink data types, then the Simulink legacy rule applies. If the block is using the parameter with nonbuilt-in data types, such as scaled fixed-point data types, then the Fixed-Point Blockset legacy rule applies. This gives full backwards compatibility with one important exception.

The backwards compatibility issue arises when a model created prior to R13 uses a Fixed-Point Blockset block with a tunable parameter, and the data type used by the block happens to be a built-in data type. If the block is unified, it will now handle the parameter using the Simulink legacy rule rather than the Fixed-Point Blockset legacy rule. This can have a significant impact. For example, suppose the tunable parameter is used in a Saturation block and the data type of the input signal is a built-in `int16`. In prior releases, the Fixed-Point Blockset block would have declared the parameter as an `int16`. For legacy fixed-point models, the MATLAB variables used for tunable parameters invariably gave their value using floating-point `double`. The unified Saturation block would now declare the tunable parameter in the generated code as `double`. This has several negatives. The variable takes up six more bytes of memory as a `double` than as an `int16`. The code for the Saturation block now includes conversions from `double` to `int16` that execute every time the block executes. This increases code size and slows down

execution. If the design was intended for use on a fixed-point processor, the use of floating-point variables and floating-point conversion code is likely to be unacceptable. It should be noted that the numerical behavior of the blocks is not changed even though the generated code is different.

For an individual block, the backwards compatibility issue is easily solved. The solution involves understanding that the Simulink legacy rule is being applied. The Simulink legacy rule preserves the data type used when assigning the value to the MATLAB variable. The problem is that an undesired data type will be used in the generated code. To solve this, you should change the way you assign the value of the tunable parameter. Determine what data type is desired in the generated code, then use an explicit type cast when assigning the value in MATLAB. For example, if `int16` is desired in the generated code and the initial value is `3`, then assign the value in MATLAB as `int16(3)`. The generated code will now be as desired.

A preliminary step to solving this issue with tunable parameters is identifying which blocks are affected. In most cases, the treatment of the parameter will involve a downcast from `double` to a smaller data type. On the **Diagnostics** tab of the **Simulation Parameters** dialog is a line item called `Parameter downcast`. Setting this item to `Warning` or `None` will help identify the blocks whose tunable parameters require reassignment of their variables.

In R13, the solution described above did not work for three unified blocks: Switch, Look-Up Table, and Lookup Table (2-D). These blocks caused errors when the value of a tunable parameter was specified using integer data types. This was a false error and has been removed. Using an explicit type cast when assigning a value to the MATLAB variable now solves the issue of generating code with the desired data types.

3

Fixed-Point Blockset 4.0 Release Notes

New Features

This section summarizes the new features and enhancements introduced in the Fixed-Point Blockset 4.0.

This section is organized into the following subsections:

Installation and Licensing

To support the sharing of models in a large organization, Version 4.0 of the Fixed-Point Blockset is automatically installed whenever Simulink is installed. You can configure models to either take full advantage of all fixed-point features, or to run without a Fixed-Point Blockset license. Therefore all Simulink users in your organization can run and work on the same model, regardless of whether they have a Fixed-Point Blockset license.

You must have a Fixed-Point Blockset license to run a model if it is configured to log minimums, maximums, or overflows. You control logging with the system-level setting **Logging mode**. If you turn logging off at the top-level system in a model, then no data is logged for any block in any subsystem of the model, and a Fixed-Point Blockset license is not required. You also need a Fixed-Point Blockset license to run a model that uses any nonbuilt-in, fixed-point data types. However, you can use the system-level setting **Data type override** to force blocks to use `doubles` or `singles` instead of fixed-point data types. Therefore, by turning the **Data type override** parameter on and the **Logging mode** parameter off at the top level of a model, a Simulink user without a Fixed-Point Blockset license can run a model with fixed-point enabled blocks. See "Global Data Type Override and Logging Modes" on page 3-5 for more information on these settings.

If you have a Fixed-Point Blockset license, you can run bit-true simulations with your models that contain fixed-point enabled blocks. If a Fixed-Point Blockset license is not available or desired, you can turn logging off and data type override on at the top level of your model and perform idealized floating point-based simulations.

If you have both a Fixed-Point Blockset license and a Real-Time Workshop license, you can generate bit-true integer code from your models with fixed-point enabled blocks. If you do not have a Fixed-Point Blockset license but you do have a Real-Time Workshop license, you can generate idealized floating-point code from your models with fixed-point enabled blocks.

Unified Simulink and Fixed-Point Blockset Blocks

Many core Simulink and Fixed-Point Blockset blocks with similar functions have been unified in this release. For example, the Sum block in the Simulink Math Operations library and the Sum block in the Fixed-Point Blockset Math library are now the same block. All the functionality from each original block has been maintained in unifying these blocks. Compatibility with fixed-point data types and/or specific fixed-point features are now available with all of these blocks, whether the blocks used are from Simulink or from the Fixed-Point Blockset. You do not need to make any changes to your earlier models as a result of this improvement. You can now use any of the unified blocks with either built-in data types or fixed-point data types, which eliminates the need to replace blocks in your models when you want to use different data types. This change does not require Simulink users to have a Fixed-Point Blockset license. Refer to “Installation and Licensing” on page 3-2 above for more information.

Fixed-Point Blockset blocks that have been unified no longer have an “F” on their block icon. However, not all Fixed-Point Blockset blocks that have counterparts in Simulink libraries have been unified. You can still use the `fixpt_convert` function to replace nonunified Simulink blocks with their Fixed-Point Blockset counterparts in your models.

Nonunified Fixed-Point Blockset blocks have an advantage over their Simulink counterparts in that they can handle more data types. As discussed above, you can easily switch them between fixed-point data types and singles or doubles using the global data type override setting. However, you may still want to use the Simulink counterparts of nonunified Fixed-Point Blockset blocks in some cases, because they support faster simulation times for the data types they handle.

The following table lists the unified blocks in this release, and the Simulink and Fixed-Point Blockset libraries in which they are found.

Block	Simulink Library	Fixed-Point Blockset Library
Abs	Math Operations	Math
Constant	Sources	Sources
Data Store Memory	Signal Routing	N/A
Data Store Read	Signal Routing	N/A
Data Store Write	Signal Routing	N/A
Gain	Math Operations	Math
Inport	Ports & Subsystems, Sources	N/A
Logical Operator	Math Operations	Logic & Comparison
Look-Up Table	Look-Up Tables	LookUp
Look-Up Table (2-D)	Look-Up Tables	LookUp
Manual Switch	Signal Routing	N/A
Memory	Discrete	N/A
Merge	Signal Routing	N/A
Multi-Port Switch	Signal Routing	Select
Outport	Ports & Subsystems, Sinks	N/A
Product	Math Operations	Math
Rate Transition	Signal Attributes	N/A
Relational Operator	Math Operations	Logic & Comparison
Relay	Discontinuities	Nonlinear
Saturation	Discontinuities	Nonlinear
Sign	Math Operations	Nonlinear

Block	Simulink Library	Fixed-Point Blockset Library
Signal Specification	Signal Attributes	N/A
Slider Gain	Math Operations	N/A
Sum	Math Operations	Math
Switch	Signal Routing	Select
Unit Delay	Discrete	Delays & Holds
Zero-Order Hold	Discrete	Delays & Holds

Global Data Type Override and Logging Modes

You can now set data type override and logging modes for systems or subsystems in the Fixed-Point Blockset Interface. The **Override data type(s) with doubles** and **Log minimums and maximums** check boxes have been removed from the mask of every Fixed-Point Blockset block. See “Data Type Override and Logging Parameters” on page 3-6.

Shift Arithmetic Block

The Fixed-Point Blockset now includes the Shift Arithmetic block in the Bits library. The Shift Arithmetic block shifts the bits or binary point of a signal, or both.

Upgrading from an Earlier Release

This section describes the upgrade issues involved in moving from the Fixed-Point Blockset 3.1 to Version 4.0.

Restoring Broken Links

Breaking library links to Fixed-Point Blockset blocks will almost certainly produce an error when you attempt to run the model. If broken links exist, you will likely uncover them when upgrading to the latest release of the Fixed-Point Blockset. The `fixpt_restore_links` command can be used to restore links for Fixed-Point Blockset blocks.

Data Type Override and Logging Parameters

The **Override data type(s) with doubles** and **Log minimums and maximums** check boxes have been removed from the mask of every Fixed-Point Blockset block. You can now set these parameters on the system or subsystem level.

When you upgrade to Version 4.0, all doubles override and logging information is cleared from your models. You can reset these controls in the Fixed-Point Blockset Interface for any system or subsystem. Access the Fixed-Point Blockset Interface from the Simulink **Tools** menu, or by typing `fxptdlg('modelname')` at the MATLAB command line.

If you have been getting or setting the block parameters `DblOver` or `dolog` in your M-code, you must now use the system parameters `DataTypeOverride` and `MinMaxOverflowLogging`.