Fixed-Point Designer™
Release Notes

How to Contact MathWorks

www.mathworks.com	Web
comp.soft-sys.matlab	Newsgroup
www.mathworks.com/contact_TS.html	Technical Support

suggest@mathworks.com	Product enhancement suggestions
bugs@mathworks.com	Bug reports
doc@mathworks.com	Documentation error reports
service@mathworks.com	Order status, license renewals, passcodes
info@mathworks.com	Sales, pricing, and general information

508-647-7000 (Phone)

508-647-7001 (Fax)

The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA 01760-2098

For contact information about worldwide offices, see the MathWorks Web site.

Fixed-Point Designer™ Release Notes

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

MathWorks products are protected by one or more U.S. patents. Please see www.mathworks.com/patents for more information.

Contents

R2013b

R2013b

Version: 4.1

New Features: Yes

Bug Fixes: Yes

C99 long long integer data type for embedded code generation

If your target hardware and your compiler support the C99 long long integer data type, you can use this data type for code generation. Using long long results in more efficient generated code that contains fewer cumbersome operations. Multi-line fixed-point helper functions can be replaced by simple expressions. This data type also provides more accurate simulation results for fixed-point and integer simulations. If you are using Microsoft® Windows® (64-bit), using long long improves performance for many workflows including:

- Using Accelerator mode in Simulink®

- Working with Stateflow® software

- Generating C code with Simulink Coder™

- Accelerating fixed-point code using `fiaccel`

- Generating C code and MEX functions with MATLAB® Coder

For more information about enabling long long in Simulink, see the **Enable long long** and **Number of bits: long long** configuration parameters on the "Hardware Implementation Pane".

For more information about enabling long long for MATLAB Coder, see `coder.HardwareImplementation`.

Model Advisor fixed-point checks with additional coverage and optimization awareness

The Model Advisor fixed-point checks now cover additional blocks in base Simulink and System Toolboxes. The checks also now include the MATLAB Function block, System objects, Stateflow, and `fi` objects. These improved checks consider model settings such as hardware configuration and code generation settings. These updated checks also avoid false negative results.

These checks require an Embedded Coder® license.

For more information, see:

- "Identify blocks that generate expensive rounding code"
- "Identify questionable fixed-point operations"
- "Identify blocks that generate expensive fixed-point and saturation code"

fi object as an index in colon expressions and an argument to numel and bit index functions

fi object as an index in colon expressions

You can now use fi objects in colon expressions. When you use fi in a colon expression, all colon operands must have integer values. See the fi and colon reference pages for examples.

fi objects as bit index input argument

The bitget, bitset, bitsliceget, bitandreduce, bitorreduce, and bitxorreduce functions now accept fi objects as the bit index argument.

fi objects as shift-value input argument

The bitsra, bitsrl, bitsll, bitrol, and bitror functions now accept fi objects as the shift-value input argument. You can use fi and built-in data type shift values interchangeably in MATLAB functions. This new capability facilitates fixed-point conversion.

numel function support for fi inputs

Effective R2013b, the numel function returns the number of elements in a fi array. Using numel in your MATLAB code returns the same result for built-in types and fi objects. Use numel to write data-type independent MATLAB code for array handling; you no longer need to use the numberofelements function.

The numel function is supported for simulation and code generation and with the MATLAB Function block in Simulink.

For more information, see numel.

Improved efficiency of data type internal rules for Lookup Table blocks

Blocks in the Lookup Tables library have a new internal rule for fixed-point data types to enable faster hardware instructions for intermediate calculations (with the exception of the Direct Lookup Table (n-D), Prelookup and Lookup Table Dynamic blocks). To use this new rule, select Speed for the **Internal Rule Priority** parameter in the dialog box. To use the R2013a internal rule, select Precision.

Derived ranges for complex variables in MATLAB Coder projects

Using the Fixed-Point Conversion tool in MATLAB Coder projects, you can now derive ranges for complex variables. For more information, see "Propose Data Types Based on Derived Ranges". This capability requires a MATLAB Coder license.

Simplified modeling of single-precision designs
Compatibility Considerations: Yes

Fixed-Point Designer™ now uses strict single-precision algorithms for operations between singles and integer or fixed-point data types. Operations, such as cast, multiplication and division, use single-precision math instead of introducing higher-precision doubles for intermediate calculations in simulation and code generation. You no longer have to explicitly cast integer or fixed-point inputs of these operations to single precision. To detect the presence of double data types in your model, use the Model Advisor "Identify questionable operations for strict single-precision design" check.

Compatibility Considerations

In R2013b, for both simulation and code generation, Fixed-Point Designer avoids the use of double data types to achieve strict single design for operations between singles and integers or fixed-point types. In previous releases, Fixed-Point Designer used double data types in intermediate

calculations for higher precision. You might see a difference in numerical behavior of an operation between earlier releases and R2013b.

For example, when you cast from a fixed-point or integer data type to single or vice versa, the type used for intermediate calculations can significantly affect numerical results. Consider:

- Input type: `ufix128_En127`

- Input value: 1.999999999254942 — Stored integer value is ($2^{128}-2^{100}$).

- Output type: single

Release	Calculation performed by Fixed-Point Designer	Output Result	Design Goal
R2013b	`Y = single(2^-127) * single(2^128-2^100)` `= single(2^-127) * Inf`	Inf	Strict singles

Previous releases	`Y = single( double(2^-127) * double(2^128 -`
	`2^100))`
	`= single(2^-127 * 3.402823656532e+38)`
		2	Higher-precision intermediate calculation

There is also a difference in the generated code. Previously, Fixed-Point Designer allowed the use of doubles in the generated code for a mixed multiplication that used single and integer types.

```
m_Y.Out1 = (real32_T)(**(real_T)**m_U.In1***(real_T)**m_U.In2);
```

In R2013b, it uses strict singles.

```
m_Y.Out1=**(real32_T)**m_U.In1*m_U.In2;
```

You can revert to the numerical behavior of previous releases, if necessary. To do so, insert explicit casting from integer and fixed-point data types to doubles for the inputs of these operations.

Range analysis support on Mac platforms

You can now perform derived range analysis of your model on Mac platforms. For more information, see "Conversion Using Range Analysis".

Changes to showInstrumentationResults function options

New option to suppress display of MATLAB code

When generating a printable instrumentation report, you can now choose to display only the tables that show information about logged variables. Used with the -printable option, the -nocode option suppresses display of the MATLAB code. Displaying only the logged variable information is useful for large projects with many lines of code.

Removal of -browser option

The showInstrumentationResults function -browser option has been removed. Use the -printable option instead. The -printable option creates a printable report and opens it in the system browser.

For more information, see showInstrumentationResults.

Changes to Continuous state-space block family range analysis support
Compatibility Considerations: Yes

The Continuous Simulink blocks State-Space, Transfer Fcn, and Zero-Pole are not supported and not stubbable for range analysis. For more information on blocks that are supported for range analysis, see "Supported and Unsupported Simulink Blocks".

Compatibility Considerations

If you have a model that contains one or more continuous State-Space, Transfer Fcn, or Zero-Pole blocks, your model is incompatible with range

analysis. Consider analyzing smaller portions of your model to work around this incompatibility.

Enhanced `fiaccel` support for `int64` and `uint64` functions

The `fiaccel` function now supports `int64` and `uint64` with `fi` inputs.

Support for LCC compiler on Microsoft Windows (64-bit) machines

If you are using Microsoft Windows (64-bit), LCC-64 is now available as the default compiler. You no longer have to install a separate compiler to perform fixed-point acceleration using `fiaccel`.

Warning for use of inexact `fi` and `fimath` property names
Compatibility Considerations: Yes

All `fi` and `fimath` property names are case sensitive and require that you use the full property names. Effective R2013b, if you use inexact property names, Fixed-Point Designer generates a warning.

Compatibility Considerations

To avoid seeing warnings for `fi` and `fimath` properties, update your code so that it uses the full names and correct cases of all these properties. The full names and correct cases of the properties appear when you display a `fi` or `fimath` object on the MATLAB command line.

Conversion of numeric variables into `Simulink.Parameter` objects

You can now convert a numeric variable into a `Simulink.Parameter` object using a single step.

```
myVar = 5; % Define numerical variable in base workspace
myObject = Simulink.Parameter(myVar); % Create data object and assign variable value to data object value
```

Previously, you did this conversion using two steps.

```
myVar = 5; % Define numerical variable in base workspace
myObject = Simulink.Parameter; % Create data object
myObject.Value = myVar; % Assign variable value to data object value
```

Fixed-point conversion test file coverage results

The MATLAB Coder Fixed-Point Conversion tool now provides test file coverage results. After simulating your design using a test file, the tool provides an indication of how often the code is executed. If you run multiple test files at once, the tool provides the cumulative coverage. This information helps you determine the completeness of your test files and verify that they are exercising the full operating range of your algorithm. The completeness of the test file directly affects the quality of the proposed fixed-point types.

This capability requires a MATLAB Coder license.

For more information, see "Code Coverage".

Fixed-point conversion workflow supports designs that use enumerated types

Using the Fixed-Point Conversion tool in MATLAB Coder projects, you can now propose data types for enumerated data types using derived and simulation ranges.

For more information, see "Propose Fixed-Point Data Types Based on Derived Ranges" and "Propose Fixed-Point Data Types Based on Simulation Ranges". This capability requires a MATLAB Coder license.

Fixed-point conversion of variable-size data using simulation ranges

Using the Fixed-Point Conversion tool in MATLAB Coder projects, you can propose data types for variable-size data using simulation ranges.

For more information, see "Propose Fixed-Point Data Types Based on Simulation Ranges". This capability requires a MATLAB Coder license.

Error checking improvements for `bitconcat`, `bitandreduce`, `bitorreduce`, `bitxorreduce`, `bitsliceget` functions

The `bitconcat`, `bitandreduce`, `bitorreduce`, `bitxorreduce`, and `bitsliceget` functions now check that all input arguments are real. If any inputs are complex, these functions generate an error.

The `bitconcat` function now generates an error in the unary syntax case, `bitconcat(a)`, if the input argument `a` is a scalar or is empty. To use `bitconcat` with one input argument, the argument must have more than one array element available for bit concatenation (that is, `length(a)>1`).

Legacy data type specification functions return numeric objects
Compatibility Considerations: Yes

In previous releases, the following functions returned a MATLAB structure describing a fixed-point data type:

- `float`
- `sfix`
- `sfrac`
- `sint`
- `ufix`
- `ufrac`

- uint

Effective R2013b, they return a Simulink.NumericType object. If you have existing models that use these functions as parameters to dialog boxes, the models continue to run as before and there is no need to change any model settings.

These functions do not offer full Data Type Assistant support. To benefit from this support, use `fixdt` instead.

Function	Return Value in Previous Releases — MATLAB structure	Return Value Effective R2013b — `NumericType`
`float('double')`	`Class: 'DOUBLE'`	`DataTypeMode: 'Double'`
`float('single')`	`Class: 'SINGLE'`	`DataTypeMode: 'Single'`

`sfix(16)`	`Class: 'FIX'`
	`   IsSigned: 1`
	`  MantBits: 16`
		`DataTypeMode: 'Fixed-point: unspecified scaling'`
		`      Signedness: 'Signed'`
		`      WordLength: 16`

`ufix(7)`	`Class: 'FIX'`
	`   IsSigned: 0`
	`  MantBits: 7`
		`DataTypeMode: 'Fixed-point: unspecified scaling'`
		`      Signedness: 'Unsigned'`
		`      WordLength: 7`

`sfrac(33,5)`	`Class: 'FRAC'`
	`    IsSigned: 1`
	`   MantBits: 33`
	`  GuardBits: 5`
		`DataTypeMode: 'Fixed-point: binary point scaling'`
		`        Signedness: 'Signed'`
		`        WordLength: 33`
		`    FractionLength: 27`

`ufrac(44)`	`Class: 'FRAC'`
	`    IsSigned: 0`
	`   MantBits: 44`
	`  GuardBits: 0`
		`DataTypeMode: 'Fixed-point: binary point scaling'`
		`        Signedness: 'Unsigned'`
		`        WordLength: 44`
		`    FractionLength: 44`

Function	Return Value in Previous Releases — MATLAB structure	Return Value Effective R2013b — NumericType
sint(55)	Class: 'INT' IsSigned: 1 MantBits: 55	DataTypeMode: 'Fixed-point: binary point scaling' Signedness: 'Signed' WordLength: 55 FractionLength: 0
uint(77)	Class: 'INT' IsSigned: 0 MantBits: 77	DataTypeMode: 'Fixed-point: binary point scaling' Signedness: 'Unsigned' WordLength: 77 FractionLength: 0

Compatibility Considerations

MATLAB Code

MATLAB code that depends on the return arguments of these functions being a structure with fields named **Class**, MantBits or GuardBits no longer works correctly. Change the code to access the appropriate properties of a NumericType object, for example, DataTypeMode, Signedness, WordLength, FractionLength, Slope and Bias.

C Code

Update C code that expects the data type of parameters to be a legacy structure to handle NumericType objects instead. For example, if you have S-functions that take legacy structures as parameters, update these S-functions to accept NumericType objects.

MAT-files

Effective R2013b, if you open a Simulink model that uses a MAT-file that contains a data type specification created using the legacy functions, the model uses the same data types and behaves in the same way as in previous releases

but Simulink generates a warning. To eliminate the warning, recreate the data type specifications using `NumericType` objects and save the MAT-file.

You can use the `fixdtupdate` function to update a data type specified using the legacy structure to use a `NumericType`. For example, if you saved a data type specification in a MAT-file as follows in a previous release:

```
oldDataType = sfrac(16);
save myDataTypeSpecification oldDataType
```

use `fixdtUpdate` to recreate the data type specification to use `NumericType`:

```
load DataTypeSpecification
fixdtUpdate(oldDataType)

ans =

  NumericType with properties:

      DataTypeMode: 'Fixed-point: binary point scaling'
        Signedness: 'Signed'
        WordLength: 16
    FractionLength: 15
            IsAlias: 0
         DataScope: 'Auto'
        HeaderFile: ''
       Description: ''
```

For more information, at the MATLAB command line, enter:

```
fixdtUpdate
```

numberofelements function being removed in a future release

The `numberofelements` function will be removed in a future release of Fixed-Point Designer software. Use `numel` instead.

R2013a

Version: 4.0

New Features: Yes

Bug Fixes: Yes

Product restructuring

The Fixed-Point Designer product replaces two pre-existing products:
Fixed-Point Toolbox™ and Simulink Fixed Point™. You can access archived
documentation for both products on the MathWorks® Web site.

Histogram logging in instrumented MATLAB Code Generation report

The `buildInstrumentedMex` and `showInstrumentationResults`
instrumentation functions now can generate log2 histograms. A histogram is
generated for each named and intermediate variable and for each expression
in your code. The code generation report **Variables** tab includes a link to
the histogram for each variable. You can use this histogram to determine
the word and fraction lengths for your fixed-point values. Refer to the
`buildInstrumentedMex` and `showInstrumentationResults` reference pages
for information.

`fi` object in indexing and switch-case expressions

Effective this release, you can use `fi` objects as indices to arrays of built-in
types and `fi` types. You can also use `fi` objects in switch-case expressions.
These changes let you use `fi` objects without having to convert them. See
the `fi` reference page for examples.

`zeros`, `ones`, and `cast` code reuse for floating-point and fixed-point types

The `zeros`, `ones`, and `cast` functions now work with fixed-point data types
as well as built-in data types. The functions can now return an output
whose class matches that of a specified numeric variable or `fi` object. For
built-in data types, the output assumes the numeric data type, sparsity,
and complexity (real or complex) of the specified numeric variable. For `fi`
objects, the output assumes the `numerictype`, complexity (real or complex),
and `fimath` of the specified `fi` object.

For example:

```
>> a = fi([],1,24,12);
>> c = cast(pi,'like',a)

c =

    3.1416

          DataTypeMode: Fixed-point: binary point scaling
            Signedness: Signed
            WordLength: 24
         FractionLength: 12

>> z = zeros(2,3,'like',a)

z =

     0     0     0
     0     0     0

          DataTypeMode: Fixed-point: binary point scaling
            Signedness: Signed
            WordLength: 24
         FractionLength: 12

>> o = ones(2,3,'like',a)

o =

     1     1     1
     1     1     1

          DataTypeMode: Fixed-point: binary point scaling
            Signedness: Signed
            WordLength: 24
         FractionLength: 12
```

This capability allows you to cleanly separate algorithm code in MATLAB from data type specifications. Using separate data type specifications enables you to:

15

- Reuse your algorithm code with different data types.

- Switch easily between fixed-point and floating-point data types to compare fixed-point behavior to a floating-point baseline.

- Try different fixed-point data types to determine their effect on the behavior of your algorithm.

- Write clean, readable code.

For more information, see Implement FIR Filter Algorithm for Floating-Point and Fixed-Point Types using cast and zeros.

Code generation for `x.^n` when `n` is a variable and `x` is a `fi` object

If the output type can be derived from the input settings, the mpower and power functions no longer require a constant exponent input. For more information, see mpower and power.

Fixed-Point Advisor support for model reference

The Fixed-Point Advisor now performs checks on referenced models. It checks the entire model reference hierarchy against fixed-point guidelines. The Advisor also provides guidance about model configuration settings and unsupported blocks to help you prepare your model for conversion to fixed point.

Automated conversion of floating-point to fixed-point types in MATLAB Coder projects

You can now convert floating-point MATLAB code to fixed-point C code using the fixed-point conversion capability in MATLAB Coder projects. You can choose to propose data types based on simulation range data, static range data, or both.

Note You must have a MATLAB Coder license.

During fixed-point conversion, you can:

- Propose fraction lengths based on default word lengths.

- Propose word lengths based on default fraction lengths.

- Optimize whole numbers.

- Specify safety margins for simulation min/max data.

- Validate that you can build your project with the proposed data types.

- Test numerics by running the test file with the fixed-point types applied.

- View a histogram of bits used by each variable.

For more information, see Propose Fixed-Point Data Types Based on Simulation Ranges and Propose Fixed-Point Data Types Based on Derived Ranges.

Improved autoscaling for models with virtual bus signals

Autoscaling with the Fixed-Point Tool now handles data type constraints for virtual buses that do not have any associated bus objects. The data type proposals take into account the constraints introduced by these bus signals.

This improved autoscaling reduces data type mismatch errors. It also enables the Fixed-Point Tool to provide additional diagnostic information when you accept autoscaling proposals. For more information, see Shared Data Type Summary.

Data Type Override for MATLAB Function block using built-in doubles and singles
Compatibility Considerations: Yes

The data type override rules for MATLAB Function block input signals and parameters have changed. If the input signals and parameters are `double` or `single`, and you specify data type override to be `Double` or `Single`, the overridden data types are now built-in `double` or built-in `single`, not `fi double` and `fi single` as in previous releases. If the input signals and

parameters are `fi` objects or fixed-point signals, and you specify data type override to be `Double` or `Single`, the overridden data types are `fi double` and `fi single` as in previous releases. For more information, see MATLAB Function Block with Data Type Override.

Compatibility Considerations

If you have MATLAB Function block code from previous releases that contains special cases for `fi double` or `fi single`, and you specify data type override to be `Double` or `Single`, you might have to update this code to handle built-in `double` and `single`.

Instrumentation for arrays of structs

The `buildInstrumentedMex` and `showInstrumentationResults` instrumentation functions now show instrumentation results for arrays of structs. Each field of each struct is logged and appears in the code generation report on the **Variables** tab.

File I/O function support

The following file I/O functions are now supported for code acceleration and generation:

- `fclose`
- `fopen`
- `fprintf`

To view implementation details, see Functions Supported for Code Acceleration or Generation.

Support for nonpersistent handle objects

You can now accelerate code using `fiaccel` for local variables that contain references to handle objects or System objects. In previous releases,

accelerating code for these objects was limited to objects assigned to persistent variables.

Load from MAT-files for code acceleration

`fiaccel` now supports a subset of the `load` function for loading run-time values from a MAT-file. It also provides a new function, `coder.load`, for loading compile-time constants. This support facilitates code generation from MATLAB code that uses `load` to load constants into a function. You no longer have to manually type in constants that were stored in a MAT-file.

To view implementation details for the `load` function, see Functions Supported for Code Acceleration or Generation.

New toolbox functions supported for code acceleration and generation

To view implementation details, see Functions Supported for Code Acceleration or Generation.

Bitwise Operation Functions

- `flintmax`

Computer Vision System Toolbox Classes and Functions

- `binaryFeatures`
- `insertMarker`
- `insertShape`

Data File and Management Functions

- `computer`
- `fclose`
- `fopen`
- `fprintf`

19

- load

Image Processing Toolbox Functions

- conndef
- imcomplement
- imfill
- imhmax
- imhmin
- imreconstruct
- imregionalmax
- imregionalmin
- iptcheckconn
- padarray

Interpolation and Computational Geometry

- interp2

MATLAB Desktop Environment Functions

- ismac
- ispc
- isunix

String Functions

- strfind
- strrep

Function to be removed in a future release
Compatibility Considerations: Yes

The saveglobalfimathpref will be removed in a future release.

Compatibility Considerations

Do not save `globalfimath` as a MATLAB preference. If you have previously saved `globalfimath` as a MATLAB preference, use `removeglobalfimathpref` to remove it.

Function being removed
Compatibility Considerations: Yes

The `emlmex` function has been removed.

Compatibility Considerations

The `emlmex` function generates an error in R2013a. Use `fiaccel` instead.