Latest Features in MATLAB Coder

October 2014

R2014b

Additional Code Generation Support

Use 41 functions and System objects in MATLAB, Communications System Toolbox, Computer Vision System Toolbox, DSP System Toolbox, and Image Processing Toolbox

bboxOverlapRatio
bwdist
bwtraceboundary
comm.IQImbalanceCompensator
dsp.CICCompensationDecimator
dsp.CICCompensationInterpolator
dsp.FarrowRateConverter
dsp.FilterCascade
dsp.FIRDecimator
dsp.FIRHalfbandDecimator
dsp.FIRHalfbandInterpolator
dsp.PeakToPeak
dsp.PeakToRMS
dsp.PhaseExtractor
dsp.SampleRateConverter
dsp.StateLevels
feof
fitgeotrans
frewind
histeq
imadjust
imclearborder
imlincomb
Imquantize
intlut
iptcheckmap
Iqcoef2imbal
vision.DeployableVideoPlayer
iqimbal2coef
ishermitian
issymmetric
medfilt2
multithresh
ode23
ode45
ordfilt2
rgb2ycbcr
selectStrongestBbox
str2double
stretchlim
ycbcr2rgb

Code Generation for Enumerated Types Based on Built-In MATLAB Integer Types

MathWorks R2014b

Control base type of enumerations for code generation

- Use int8, uint8, int16, uint16 and int32 as enumeration types
- Reduce memory usage of enumerated types
- Interface to legacy code or match company standards

MATLAB

```
classdef(Enumeration) LEDcolor < int32
    enumeration
        GREEN(1),
        RED(2)
    end
end
```

C

```
enum LEDcolor
{
    GREEN = 1,
    RED
};

typedef enum LEDcolor LEDcolor;
```

MATLAB

```
classdef(Enumeration) LEDcolor < int16
    enumeration
        GREEN(1),
        RED(2)
    end

end
```

C

```
typedef short LEDcolor;
#define GREEN ((LEDcolor)1)
#define RED ((LEDcolor)2)
```

MathWorks R2014b

Code Generation for Function Handles in Structures

Invoke functions indirectly and parameterize operations that you repeat frequently

- Define handles that reference user-defined functions and built-in functions supported for code generation
- Define function handles as scalar values
- Define structures that contain function handles
- Pass function handles as arguments to other functions (excluding extrinsic functions)

For Use with Embedded Coder

ARM Cortex-A Optimized Code for System Objects

Replace System objects with NEON-optimized functions for ARM Cortex-A cores

```
persistent h;
if isempty(h)
    h = dsp.FIRFilter('Numerator', fir1(63, 0.33));
end
y1 = step(h, u1);
```

- Use 13 System objects including:
  - `dsp.FIRFilter, dsp.FFT, dsp.IFFT, dsp.CICCompensationInterpolator, dsp.DigitalUpConverter, dsp.DigitalDownConverter`

```
/* System object Outputs function: dsp.FIRFilter */
ne10_fir_float_neon(&obj->cSFunObject.S, &U0[0], &b_y1[0], 76U);
```

- ARM Cortex-A Code Replacement Library supports Ne10 functions such as:
  - `ne10_fir_init_float(), ne10_fft_c2c_1d_float32_neon(), ne10_fir_interpolate_float_neon(), ne10_fir_decimate_float_neon()`

Detailed listing here

» `verifyFIRfilteronARMCortexAprocessorMLworkflow`

Multiple Entry Point Support for Software-in-the-Loop (SIL) Execution

SIL/PIL verification for code libraries generated from multiple entry-point functions

```
sil_config = coder.config('lib');
sil_config.VerificationMode = 'SIL';

codegen('-config', sil_config, foo, '-args', 3, bar, '-args', 4, '-report');

foo_sil('foo',3);
foo_sil('bar',6);
```

Execution Time Profiling Using SIL/PIL Execution

Use execution time profile to check whether your code runs within the required time on your target hardware

- View and compare plots of function execution times
- Access and analyze execution time profiling data
- Execution times calculated from data obtained through instrumentation probes added to SIL or PIL test harness

Code Execution Profiling Report for kalman01

The code execution profiling report provides metrics based on data collected from a SIL or PIL execution. Execution times are calculated from data recorded by instrumentation probes added to the SIL or PIL test harness or inside the code generated for each component. See Code Execution Profiling for more information.

1. Summary

Total time (seconds × 1e-09)	2206501
Measured time display options	('Units', 'Seconds', 'ScaleFactor', '1e-09', 'NumericFormat', '%0.0f')
Timer frequency (ticks per second)	3.06e+09
Profiling data created	01-Apr-2014 10:21:25

2. Profiled Sections of Code

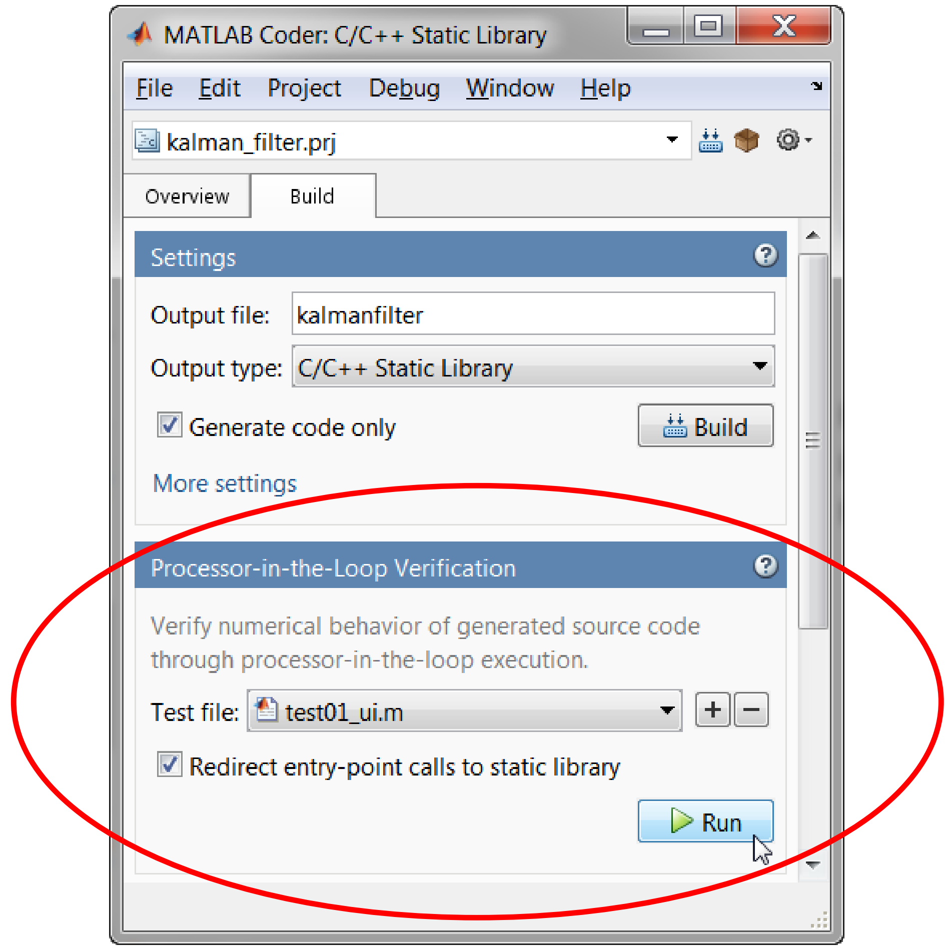
Section	Maximum Execution Time	Average Execution Time	Maximum Self Time	Average Self Time	Calls	
kalman01_initialize	1076	1076	1076	1076	1	
kalman01	16009	7351	16009	7351	300	
kalman01_terminate	138	138	138	138	1	

Processor-in-the-Loop (PIL) Verification

Verify numerical behavior of generated code on target processor

- Cross-compile generated code and execute object code on target processor or instruction set simulator

- Reuse MATLAB-based test cases to exercise generated code on connected hardware



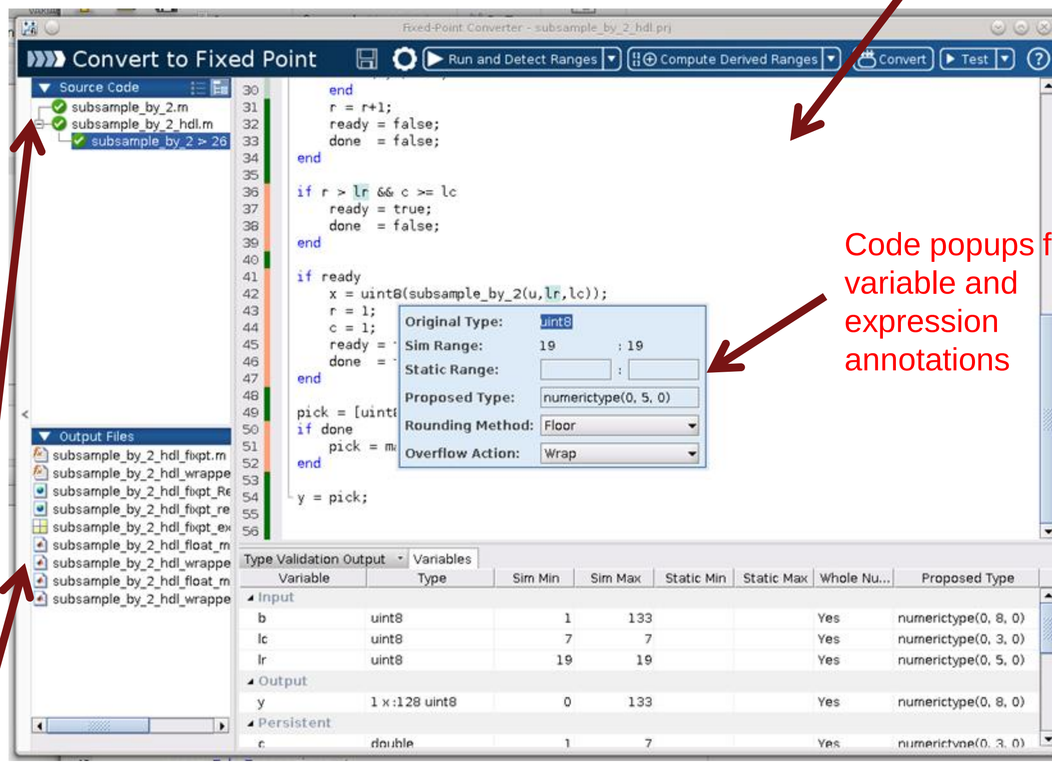
For Use with Fixed-Point Designer

Fixed-Point Converter App for Automated Conversion of Floating-Point MATLAB Code

Standalone UI enables automatic conversion of MATLAB code to fixed point

- Run test benches and/or code snippets to autodefine input types or manually specify input types.
- Iteratively refine numeric types with simulations and derived ranges before building and testing the converted code.
- Works outside of MATLAB and HDL Coder workflows

Live editor for easy design modification



Code popups for variable and expression annotations

Integrated editor for simultaneously viewing source files and generated artifacts

Automated Fixed-Point Conversion for Commonly Used DSP System objects

Propose and apply fixed-point data types for some System objects based on simulation range data

Enable conversion of the following DSP System Toolbox System objects to fixed point using the Fixed-Point Converter app:

- `dsp.BiquadFilter`
- `dsp.FIRFilter`, direct form only
- `dsp.FIRRateConverter`
- `dsp.LowerTriangularSolver`
- `dsp.UpperTriangularSolver`
- `dsp.ArrayVectorAdder`