MATLAB EXPO 2018
KOREA
MATLAB Coder User Story

- Using MATLAB®
  - Try a new idea quickly
  - Evaluation of the system by testing and analysis
  - High quality C code generation in short time
User Story: Development of IoT Device using MATLAB Coder

- **C code generation efficiency was not good** ➔ **MathWorks support**
  - Implementation of signal processing algorithms for battery-driven IoT equipment (Low Power ↔ High Speed Signal Processing)
  - Target Processor Core: Cortex-M4
  - How to set options
  - Embedded Coder® Introduced

- **In-depth support**
  - Adopted Code Replacement Library (CRL) for Cortex-M4
User Story: Image processing & recognition system

- Image processing algorithm implemented in the ARM Cortex-A8 core
- MATLAB code optimization reduces execution time of generated C code
- MATLAB exceeded Hand-Written Code just Two-days work
Why Engineers translate MATLAB to C today?

- **Implement** C code on processors or hand off to software engineers
- **Integrate** MATLAB algorithms with existing C environment using source code and static/dynamic libraries
- **Prototype** MATLAB algorithms on desktops as standalone executables
- **Accelerate** user-written MATLAB algorithms
Automatic Translation of MATLAB to C

- Maintain one design in MATLAB
- Design faster and get to C/C++ quickly
- Test more systematically and frequently
- Spend more time improving algorithms in MATLAB
Key Takeaways

MATLAB Coder
Basic

Coding techniques for efficient C code generation

Code Replacement Library and System Object

MATLAB EXPO 2018
Key Takeaways

MATLAB Coder
Basic
Code Replacement Library
and System Object

Coding techniques for
efficient C code generation
Optimized C

MATLAB EXPO 2018
Outline of C code generation workflow from MATLAB

1. C Code Generation
   - MATLAB algorithm development
   - Split into Algorithm and test bench
   - Replace with MATLAB functions supported by MATLAB Coder
   - Is C code generation possible?

2. Code Optimization
   - Configuration and setup
   - MATLAB Code optimization
   - Use Code Replacement Library
   - Rewrite/Modify the generated code directly for optimization
   - Is code efficiency OK?
How to use MATLAB Code and Test bench

Original

- Signal generation
- Algorithm
- Visualization

Test Bench

- Description
  - Signal Generation
  - Call Target Algorithm
  - Visualization / Analysis

- Restriction
  - None

Algorithm

- Description
  - Function MATLAB file with arguments

- Restriction
  - Supported function
  - Syntax

C source code
How to make MATLAB Coder Project

- MATLAB Coder App & project
  - APP: Beginner or for those unfamiliar with settings ➔ APP
  - Command: Experienced or when settings are fixed ➔ `codegen filename -options`
MATLAB Language Support for Code Generation

- MATLAB Coder Documentation
  - MATLAB Programming for Code Generation
  - Language, Functions, Objects Support

- Web Document

Language, Function, and Object Support
MATLAB® language features, functions, classes, and System objects supported for C and C++ code generation.

Topics
MATLAB Language Features Supported for C/C++ Code Generation
Use the MATLAB language features and functions that code generation supports.

Functions and Objects Supported for C/C++ Code Generation — Alphabetical List
Use the functions, classes, and System objects that code generation supports.

Functions and Objects Supported for C/C++ Code Generation — Category List
Use the functions, classes, and System objects that code generation supports.
1700 Functions & 20 Toolboxes Supported

- Aerospace Toolbox
- Audio System Toolbox
- Automated Driving System Toolbox
- Communications System Toolbox
- Computer Vision System Toolbox
- Control System Toolbox
- DSP System Toolbox
- Fixed-Point Designer
- Image Acquisition Toolbox
- Image Processing Toolbox
- Model Predictive Control Toolbox
- Neural Networks Toolbox
- Optimization Toolbox
- Phased Array System Toolbox
- Robotics System Toolbox
- Signal Processing Toolbox
- Stats & Machine Learning Toolbox
- System Identification Toolbox
- Wavelet Toolbox
- WLAN System Toolbox

MATLAB EXPO 2018
Determine if function is suitable for code generation

- Code Check with coder.screener
  - Using `coder.screener`, check before run MATLAB Coder APP
    ```matlab
    >> coder.screener('filename')
    ```
  - Check each user function to see the problem area

<Code Generation Readiness Result>
How to check in detail for code generation

- Check 「Check for Run-Time Issues」 in MATLAB Coder App
  - Identify unsupported functions and syntax
  - Display report of relevant line and explanation
Settings for Code Generation

- **Saturate on integer overflow**: Saturation processing is added and finally result in speed reduction.
- **Support only pure-integer numbers**: For fixed-point processors not to generate floating-point code, enable Support only pure-integer numbers.

- **Enable Variable-sizing**: Dynamic memory allocation increases the risk of stack overflow, performance degradation in large data sets.
- **Generate re-entrant code**: Data can be accepted by function arguments and called from multiple processes.

※ Embedded Coder option
Code Generation Report with Static code metrics

- File Information
  - Number of lines of code per file
  - Global variable, size (bytes)
  - Stack size
Validation and Equivalence Testing
SIL/PIL Verification (Embedded Coder feature)

- Equivalence verification function
  - SIL (Software-In-the-Loop): Run generated C code on PC
  - PIL (Processor-In-the-Loop): Run generated C code on Target Hardware or Emulator

- Code Execution Profiling
  - Measure task / function execution time
Tips for basic use

- Add `%#codegen` directive to your function

- No need to delete display functions like plot, disp, axis, figure

- Keep unsupported function from code generation by using `coder.extrinsic(‘func’)`

- Specify the data type and matrix size by putting the `assert` instruction in the generation target code.
  - `assert(isa(param, ’single’));`  % Parameter: Single Date Type
  - `assert(all(size(param) == [3, 4]));`  % Matrix size : 3x4
  - `assert(isscalar(param));`  % Scalar
Key Takeaways

MATLAB Coder Basic

Coding techniques for efficient C code generation

Code Replacement Library and System Object

Optimized C
Outline of C code generation workflow from MATLAB

1. C Code Generation
   - MATLAB algorithm development
   - Split into design object / test bench
   - Replace with MATLAB functions supported by MATLAB Coder
   - Is C code generation possible?

2. Code Optimization
   - Configuration and setup
   - MATLAB Code optimization
   - Use Code Replacement Library
   - Rewrite/Modify the generated code directly for optimization
   - Is code efficiency OK?
Why do we need MATLAB code optimization?
MATLAB is the Language that supports a variety of input and output

function a = foo(b, c)
a = b * c;

void foo(const double b[15], const double c[30], double a[18])
{
    int i0, i1, i2;
    for (i0 = 0; i0 < 3; i0++) {
        for (i1 = 0; i1 < 6; i1++) {
            a[i0 + 3 * i1] = 0.0;
            for (i2 = 0; i2 < 5; i2++) {
                a[i0 + 3 * i1] += b[i0 + 3 * i2] * c[i2 + 5 * i1];
            }
        }
    }
}

Expression is different in C code

double foo(double b, double c)
{
    return b*c;
}
MATLAB code optimization
Reduced branch path and variable (coder.Constant)

- Reduce unexecuted branch path ➞ Improved C code execution speed
- Variable reduction ➞ Memory reduction

```
>> codegen MF -args {coder.Constant(1), coder.Constant(4), zeros(10,1)}
```

```
function out = MF(flag, gain, in)
%# codegen 1 4 10x1
switch flag
  case 1
    out = gain * sin(in);
  case 2
    out = gain * cos(in);
  otherwise
    out = gain * sqrt(in);
end
```

```
#define gain (4.0)
void MF(const double in[10], double out[10])
{
  int k;
  for (k = 0; k < 10; k++) {
    out[k] = gain * sin(in[k]);
  }
}
```

- If the input arguments are variables, the generated code is treated as a variable
- If the input arguments are constant, branch deletion, define definition
MATLAB code optimization
Minimize Redundant Operations in Loops

- Increase speed of C code execution

Before improvement

```matlab
function C=SeriesFunc(A,B,n)
C=zeros(size(A));
for i=1:n
    C=C+inv(B)*A^i*B;
end
```

After improvement

```matlab
function C=SeriesFunc(A,B,n)
C=zeros(size(A));
Bi = inv(B);
for i=1:n
    C = C+Bi*A^i*B;
end
```

Move inv(B) out of loop
MATLAB code optimization
Multicore-capable code generation using OpenMP

- Multithreaded ➔ Performance improvement

Before improvement

```matlab
function a = test_for(r) %#codegen
    a=ones(10,256);
    for i=1:10
        a(i,:)=real(fft(r(i,:)));
    end
```

Run parallel with `parfor`

After improvement

```matlab
function a = test_for(r) %#codegen
    a=ones(10,256);
    parfor i=1:10
        a(i,:)=real(fft(r(i,:)));
    end
```

Add pragmas for OpenMP

```c
void test_for(const double r[2560], double a[2560]) {
    #pragma omp parallel for 
    num_threads(omp_get_max_threads()) 
    private(i0) 
    firstprivate(dcv0,b_r)
    for (i = 0; i < 10; i++) {
        ....
    }
}
```

- Parfor is a command for parallel operation of Parallel Computing Toolbox
- OpenMP is an API for C / C++ parallel computing

MATLAB EXPO 2018
MATLAB code optimization
Differentiate simulation and code generation behavior

- Switch the MATLAB code used for each purpose with the `coder.target` option

```matlab
function a = switch_fcn(r) %#codegen
    a = ones(10,256);
    if coder.target('Rtw')
        % for code generation
        out = myCodeGenFcn(input);
    else
        % for MATLAB and etc.
        out = mySimFcn(input);
    end
```

<table>
<thead>
<tr>
<th>Option</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>'MATLAB'</td>
<td>Running in MATLAB (not generating code)</td>
</tr>
<tr>
<td>'MEX'</td>
<td>Generating a MEX function</td>
</tr>
<tr>
<td>'Sfun'</td>
<td>Simulating a Simulink® model</td>
</tr>
<tr>
<td>'Rtw'</td>
<td>Generating a LIB, DLL, or EXE</td>
</tr>
<tr>
<td>'HDL'</td>
<td>Generating an HDL target</td>
</tr>
<tr>
<td>'Custom'</td>
<td>Generating a custom target</td>
</tr>
</tbody>
</table>
MATLAB code optimization
Eliminate Redundant Copies of Function Inputs

- Reusing Variables ➔ Reduce memory usage and execution time

Before improvement

```matlab
function y = foo( A, B )
  %#codegen
  y = A * B;
end

double foo(double A, double B){
  double y;
  y = A * B;
  return y;
}
```

After improvement

```matlab
function A = foo( A, B )
  %#codegen
  A = A * B;
end

void foo(double *A, double B) {
  *A *= B;
}
```

passed by reference ➔ Reduction of memory usage
Key Takeaways

MATLAB Coder

Basic

Coding techniques for efficient C code generation

Code Replacement Library and System Object
Concept of Code Replacement

Replace Green blocks with Blue blocks

```matlab
function r = my_add(a,b)
    %#codegen
    r=a + b;
end
```

Replace `+` with custom function `add_mul`

```matlab
double my_add(double a, double b)
{
    return a + b;
}

double my_add(double a, double b)
{
    return add_mul(a + b);
}
```
What is Code Replacement Library (CRL)?

- Replacement function for code optimized for target processor (Embedded Coder required)
  ※ Supported Processor: ARM Cortex-M/A, Intel x86, TI C55x/C64x/ C67x and so on
- Corresponds to arithmetic operations, mathematical operations, signal processing arithmetic, etc. (depending on the target processor)
- User registration is possible >> crtool

MATLAB program

```matlab
function [y1, y2, y3, y4] = arm_EC_sps(u)
    y1 = u1 + u2;
    y2 = u1 - u2;
    y3 = u1 .* u2;
    y4 = sin(u3);
    y5 = cos(u3);
    y6 = sqrt(u3);
end
```

C code generated for Cortex-M

```c
void arm_EC_sps(const float u1[2], const float u2[2], const float u3[2],
                 float y2[2], float y3[2], float y4[2],
                 float y5[2], float y6[2])
{
    mw_arm_add_f32(u1, u2, &y1[0], 2U);
    mw_arm_sub_f32(u1, u2, &y2[0], 2U);
    mw_arm_mul_f32(u1, u2, &y3[0], 2U);
    *y4 = arm_sin_f32(u3);
    *y5 = arm_cos_f32(u3);
    mw_arm_sqrt_f32(u3, &y6);
}
```

/* System object Outputs function: dsp.FIRFilter */
arm_fir_f32(&b_obj->S, &U0[0], &y1[0], 750);
ARM Cortex-M Optimized Code

Up to 10x speed boost for ARM Cortex-M cores

- Replace basic math operations with calls to CMSIS-optimized functions for ARM Cortex-M cores

- ARM Cortex-M Code Replacement Library supports CMSIS functions such as:
  - `arm_add_q15()`, `arm_sub_q31()`,
  - `arm_mult_f32()`, `arm_sin_f32()`,
  - `arm_cos_f32()`, `arm_sqrt_q31()`,
  - `arm_cmplx_mult_cmplx_f32()`,
  - `arm_q7_to_float()`, `arm_shift_q15()`
ARM Cortex-A Optimized Code

Faster execution speed on Cortex-A processor cores using NEON SIMD code replacements

- Replace basic vector operations with calls to NEON SIMD code, such as:
  - ne10_add_float_neon(), ne10_sub_float_neon(), ne10_mul_float_neon(), ne10_divc_float_neon(), ne10_abs_float_neon

- Map code replacements to libraries: Ne10 open software project
  - A set of common, heavily optimized functions for ARM Neon Architecture
How to Apply Code Replacement Library

1. Set the operation (Formula, System Object), property, data type, rounding

2. Set custom code in settings

3. Confirm the Result of replacement from the report
What is System Object?

- MATLAB class specialized for dynamic system streaming processing
- Have a common interface (setup / reset / step / release method)

```matlab
>> dft = dsp.FFT;
```

<table>
<thead>
<tr>
<th>Processing form</th>
<th>System Object</th>
<th>MATLAB Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory consumption</td>
<td>Streaming</td>
<td>Batch</td>
</tr>
<tr>
<td>Simulink compatible</td>
<td>Small</td>
<td>Big</td>
</tr>
<tr>
<td>State management</td>
<td>Almost all</td>
<td>Some</td>
</tr>
<tr>
<td>C Code Generation</td>
<td>Simple</td>
<td>User Setting</td>
</tr>
<tr>
<td>HDL Code Generation</td>
<td>Almost all</td>
<td>Some</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data input for each chunk</th>
<th>Data input for each chunk</th>
</tr>
</thead>
<tbody>
<tr>
<td>34 14 186 71</td>
<td>348 92 22 137 24 495 71</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Process per chunk</th>
<th>Process per chunk</th>
</tr>
</thead>
<tbody>
<tr>
<td>33 12 9 34 14 186 71</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Result output per chunk</th>
<th>Result output per chunk</th>
</tr>
</thead>
<tbody>
<tr>
<td>33 12 9 34 14 186 71</td>
<td>33 12 9 34 14 186 71</td>
</tr>
</tbody>
</table>
```matlab
System Object

H = dsp.FIRFilter;
for time = 1:10
    out = step(H, in)
end
```

MATLAB
Frame data
Batch processing code

```
out = filter(b,a,in)
```

MATLAB EXPO 2018
## Supported System Objects for Code Replacement Library

**CMSIS / Ne 10 library support**

<table>
<thead>
<tr>
<th>System Object</th>
<th>Cortex-M CMSIS</th>
<th>Cortex-A Ne10</th>
</tr>
</thead>
<tbody>
<tr>
<td>dsp.FIRFilter</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>dsp.FIRDecimator</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>dsp.FIRInterpolator</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>dsp.LMSFilter</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>dsp.BiquadFilter</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>dsp.FFT</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>dsp.IFFT</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>dsp.Convolver</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>dsp.Crosscorrelator</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>dsp.Mean</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>dsp.RMS</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>dsp.StandardDeviation</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>System Object</th>
<th>Cortex-M CMSIS</th>
<th>Cortex-A Ne10</th>
</tr>
</thead>
<tbody>
<tr>
<td>dsp.VariableBandwidthFIRFilter</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>dsp.FIRHalfbandInterpolator</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>dsp.FIRHalfbandDecimator</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>dsp.CICCompensationDecimator</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>dsp.CICCompensationInterpolator</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>dsp.DigitalDownConverter</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>dsp.DigitalUpConverter</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>dsp.SampleRateConverter</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>dsp.Variance</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

Display code replacement library list

`>> crviewer`

Supported Simulink Blocks and MATLAB System Objects for CMSIS Library
How to deal with functions not support code replacement?

Customizing code replacement library

```
crtool
```

- Create MATLAB function and corresponding C function as library
  - Input / output, data type, range of dimensions, etc.

Customize MATLAB functions

- Delete unnecessary lines (Input / output check, error handling, etc.)
- Data type change
- Rewrite to equivalent code substitution correspondence function
- Replace?
- Equivalence verification
How to find out unnecessary part of MATLAB functions

- Use Run and Time to generate profile report
- Use Debug functions to find out unnecessary part

The unexecuted part in the function is grayed out

Debug / step execution to determine unnecessary parts
Key Takeaways

MATLAB Coder Basic

Coding techniques for efficient C code generation

Code Replacement Library and System Object
R2018a New Features

- Row Major Code Generation
- Interactive Code Traceability
- Generated MEX Profiling
- New Code Generation Report
- Array Shape Preservation
- Sparse Matrix Support
- Destructor Support
- Polyspace Integration
- Code Optimization

New in MATLAB Coder 4.0
Useful Resources for MATLAB Coder

- Embedded Coder Capabilities for Code Generation from MATLAB Code
- Best Practices for Converting MATLAB Code to Fixed Point
- MATLAB and C/C++ Resources
- What is System Object?
- Device Driver Blocks
- MATLAB to C with MATLAB Coder