MATLAB EXPO 2018

KOREA

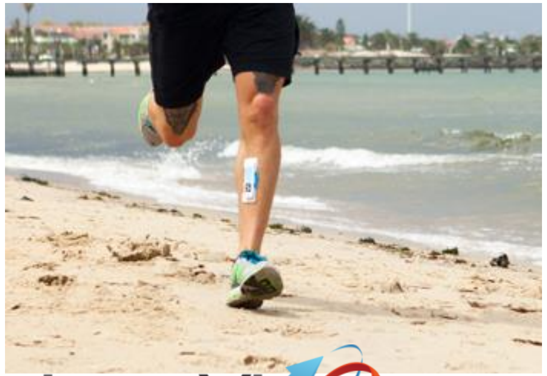
MATLAB EXPO 2018

MATLAB의 C코드 생성 워크플로우 및 최적화 요령

정승혁 과장

MATLAB Coder User Story

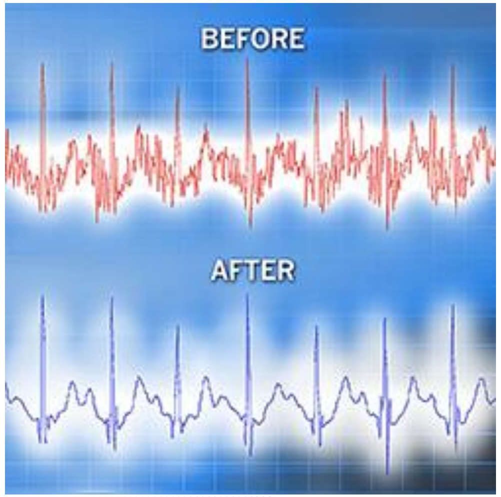
- Using MATLAB®
  - Try a new idea quickly
  - Evaluation of the system by testing and analysis
  - High quality C code generation in short time



dorsaVi
Inspiring the world to move

DELPHI

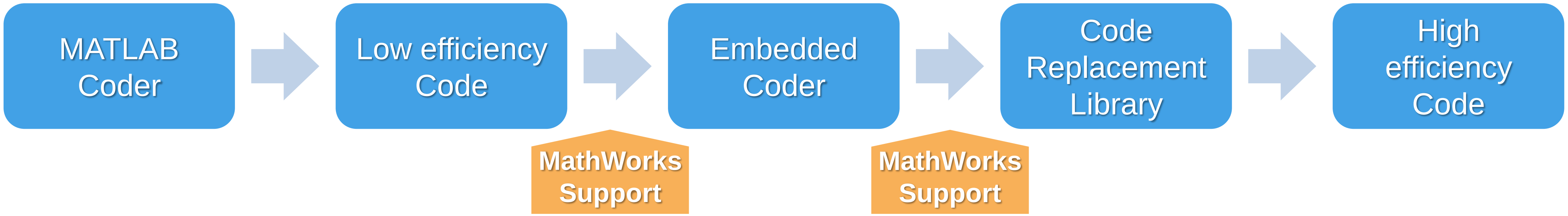
iSonea
Sound Asthma Management



VivaQuant™
Quantifying vital signs. Accurately.

BAKER HUGHES

User Story : Development of IoT Device using MATLAB Coder

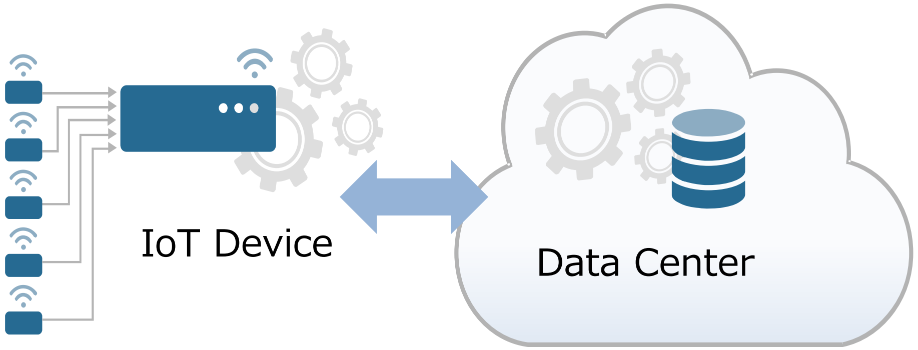


MATLAB Coder → Low efficiency Code → Embedded Coder → Code Replacement Library → High efficiency Code

MathWorks Support

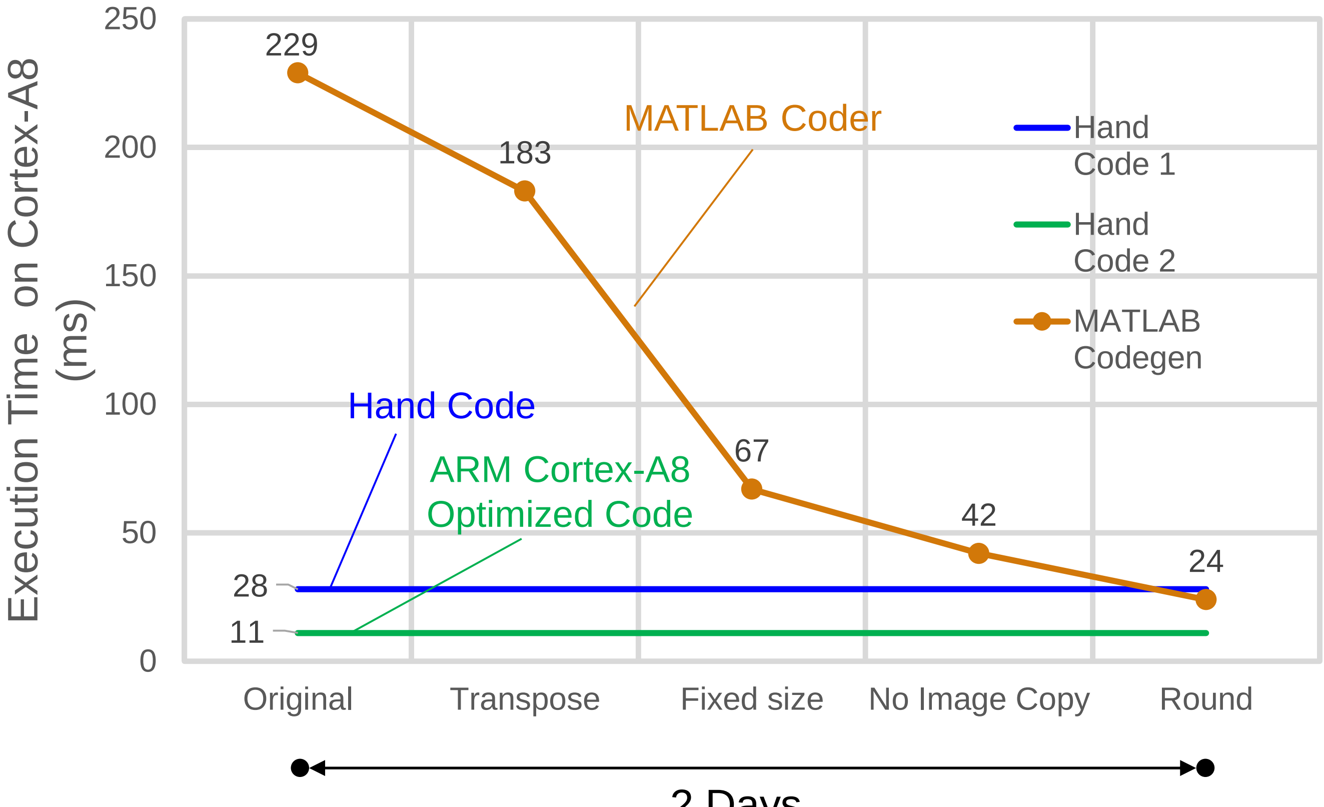
MathWorks Support

- 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

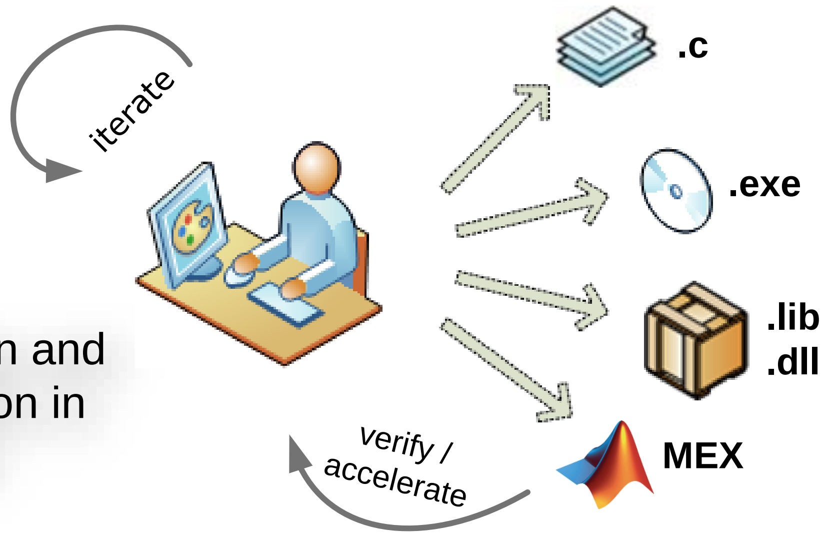


Why Engineers translate MATLAB to C today?

.c	**Implement** C code on processors or hand off to software engineers
.lib .dll	**Integrate** MATLAB algorithms with existing C environment using source code and static/dynamic libraries
.exe	**Prototype** MATLAB algorithms on desktops as standalone executables
MEX	**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



Algorithm Design and Code Generation in MATLAB

Key Takeaways

MATLAB Coder Basic

Coding techniques for efficient C code generation

Code Replacement Library and System Object

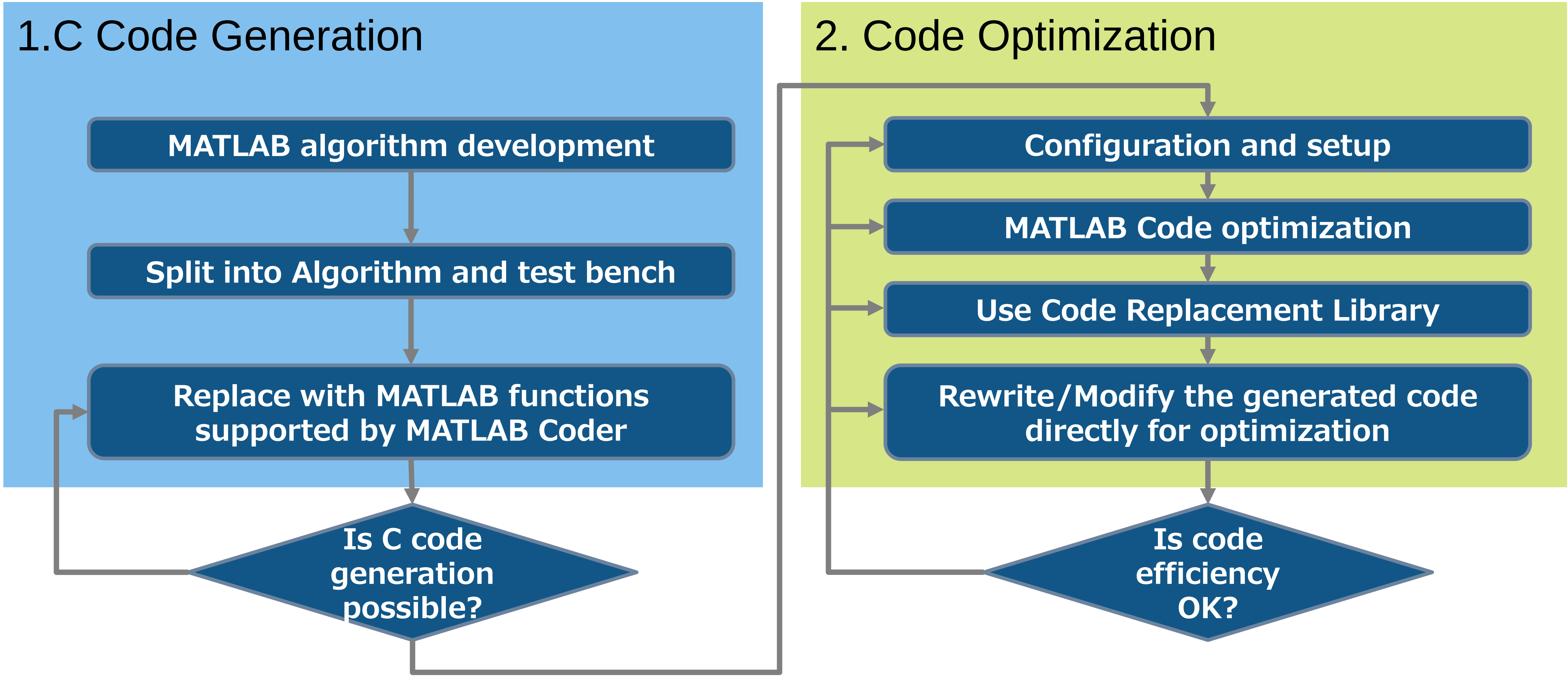
Key Takeaways

MATLAB Coder Basic

Coding techniques for efficient C code generation

Code Replacement Library and System Object

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

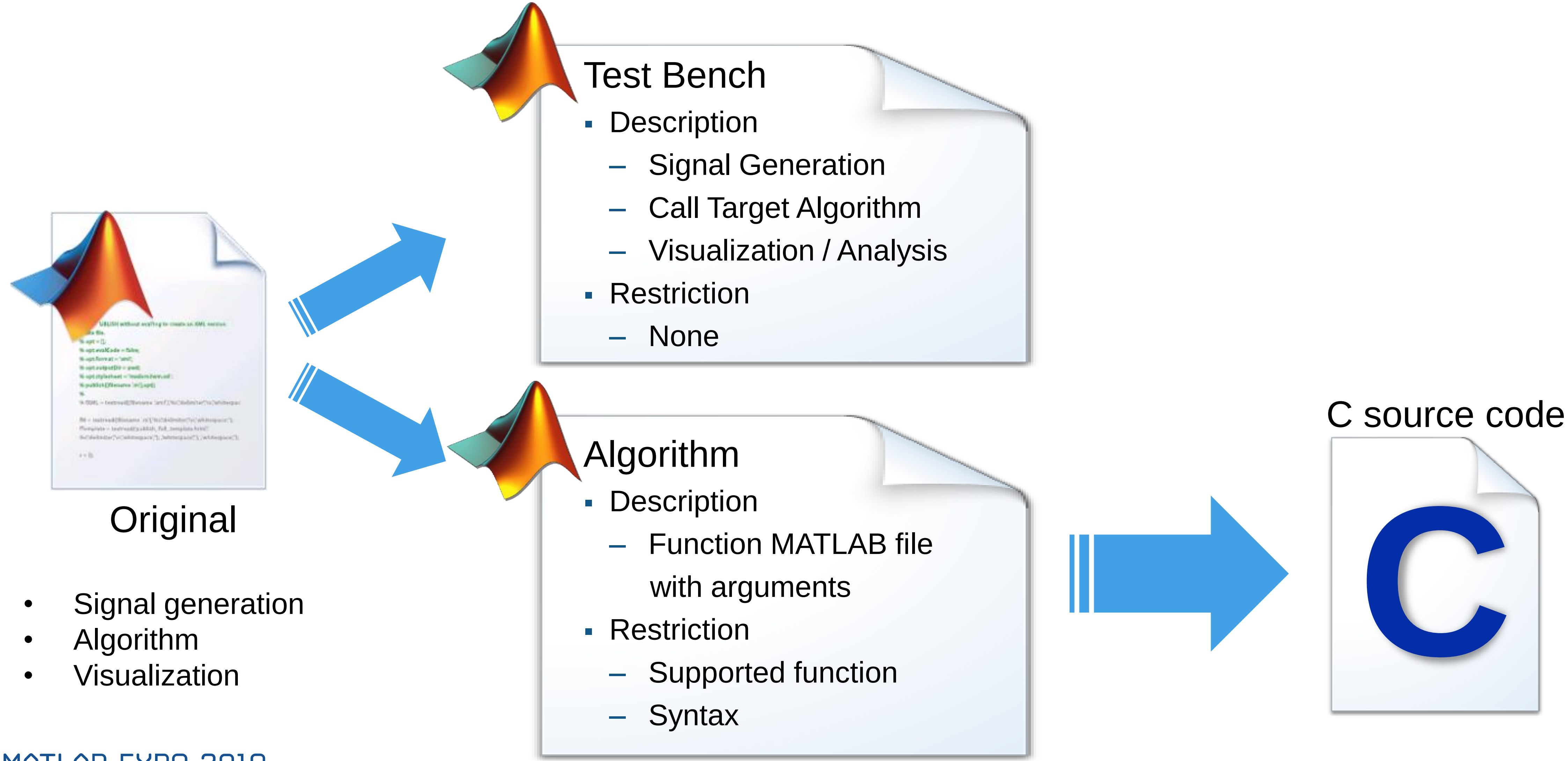
MATLAB Code optimization

Use Code Replacement Library

Rewrite/Modify the generated code directly for optimization

Is code efficiency OK?

How to use MATLAB Code and Test bench



Test Bench

- Description
  - Signal Generation
  - Call Target Algorithm
  - Visualization / Analysis
- Restriction
  - None

Algorithm

- Description
  - Function MATLAB file with arguments
- Restriction
  - Supported function
  - Syntax

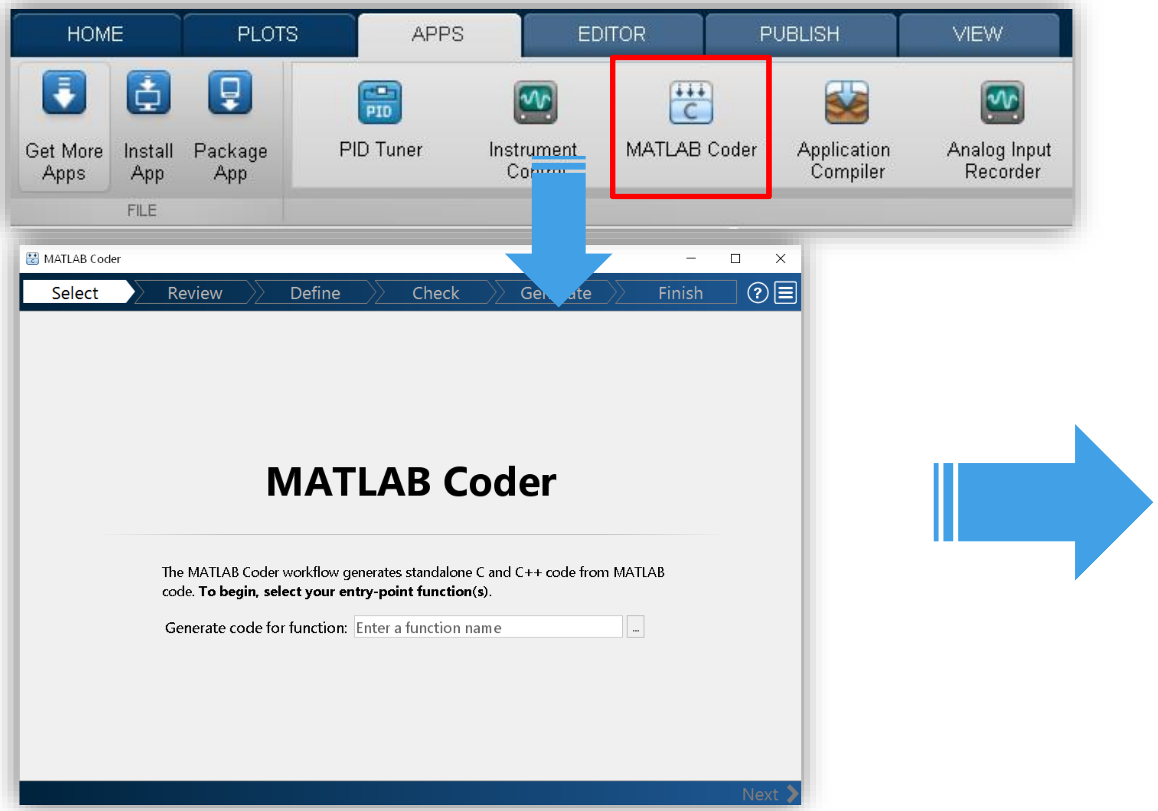
C source code

Original

- Signal generation
- Algorithm
- Visualization

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`

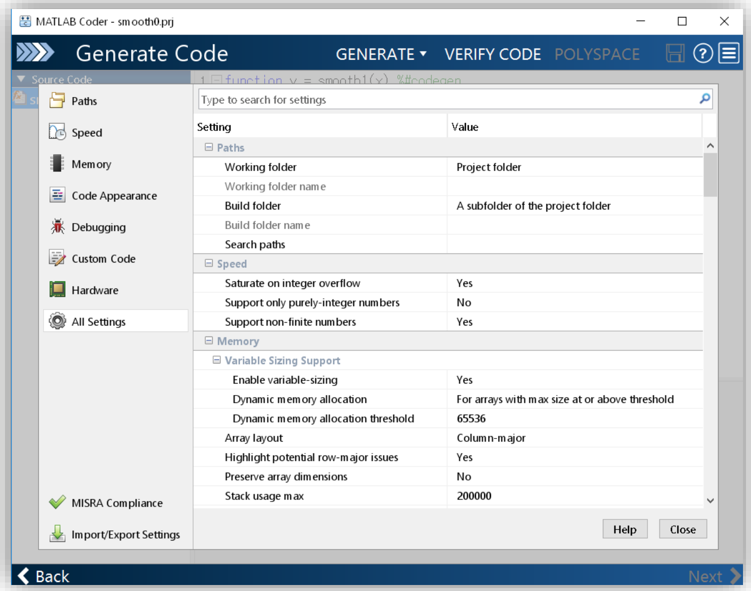


MATLAB Coder

The MATLAB Coder workflow generates standalone C and C++ code from MATLAB code. **To begin, select your entry-point function(s).**

Generate code for function: Enter a function name

Detail Settings

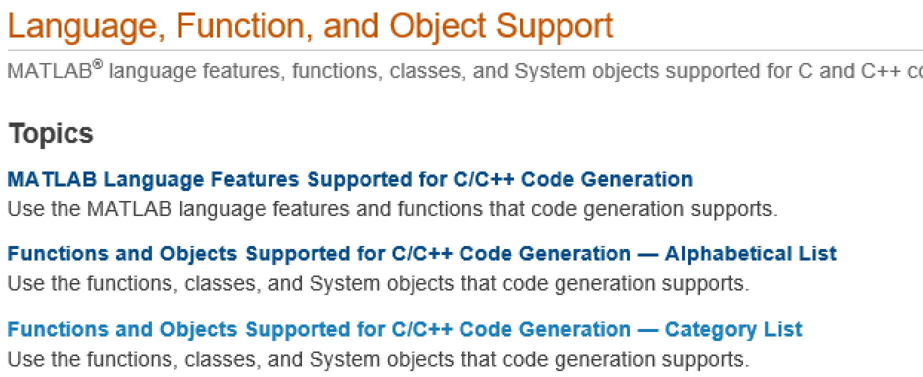


```
1 function y = smooth1(x) %#codegen
```

Setting	Value
⊟ Paths	
Working folder	Project folder
Working folder name	
Build folder	A subfolder of the project folder
Build folder name	
Search paths	
⊟ Speed	
Saturate on integer overflow	Yes
Support only purely-integer numbers	No
Support non-finite numbers	Yes
⊟ Memory	
⊟ Variable Sizing Support	
Enable variable-sizing	Yes
Dynamic memory allocation	For arrays with max size at or above threshold
Dynamic memory allocation threshold	65536
Array layout	Column-major
Highlight potential row-major issues	Yes
Preserve array dimensions	No
Stack usage max	200000

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

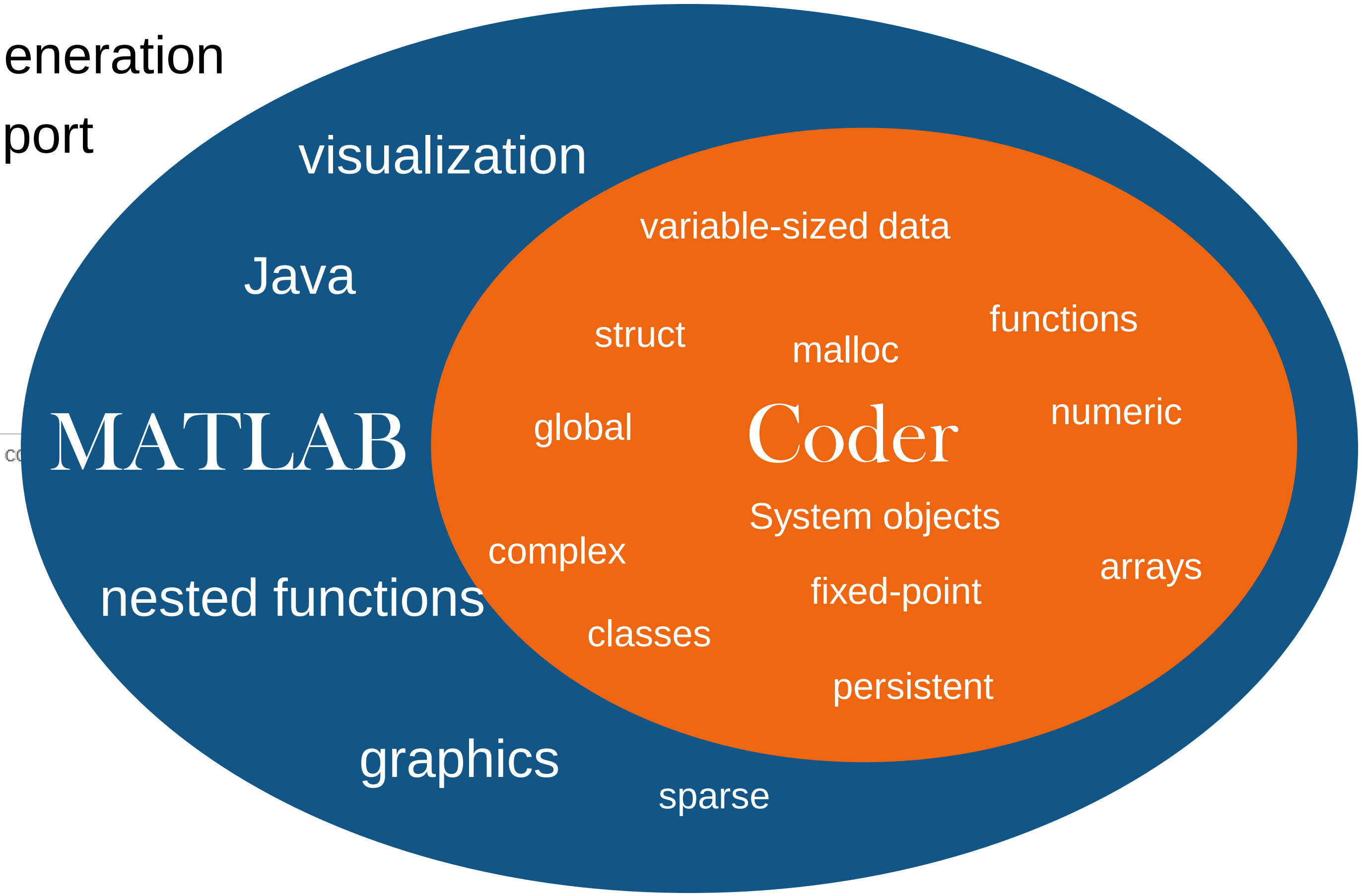
MATLAB® language features, functions, classes, and System objects supported for C and C++ co

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.



visualization

Java

MATLAB

nested functions

graphics

sparse

variable-sized data

struct

malloc

functions

global

Coder

numeric

System objects

complex

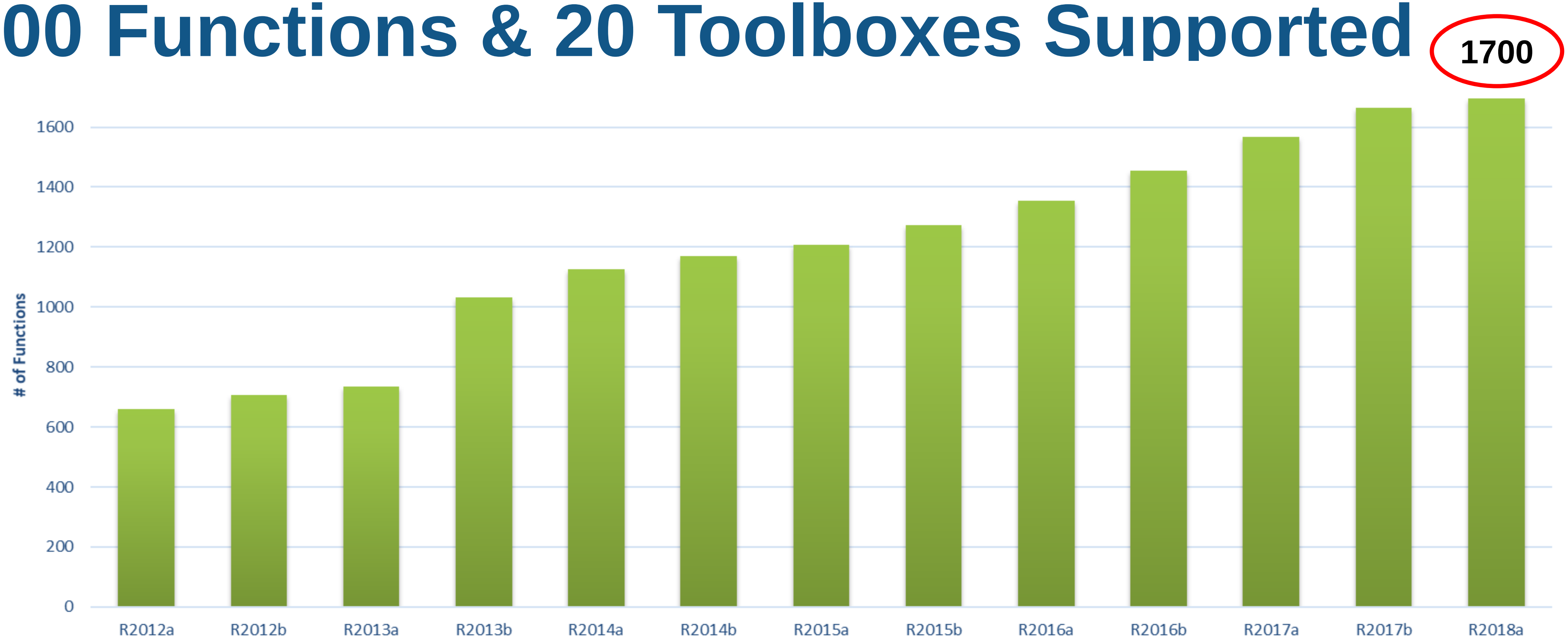
arrays

fixed-point

classes

persistent

1700 Functions & 20 Toolboxes Supported

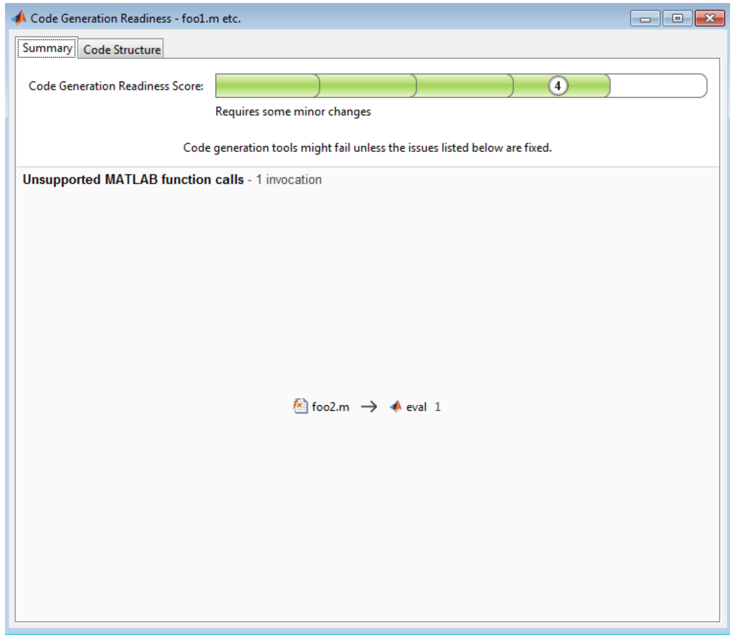


- Aerospace Toolbox
- Audio System Toolbox
- Automated Driving System Toolbox
- Communications System Toolbox
- Computer Vision System Toolbox
- Control System Toolbox
- DSP System Toolbox
- Fixed-Point Designer
- Image Acquisition Toolbox
- Image Processing Toolbox
- Model Predictive Control Toolbox
- 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

Determine if function is suitable for code generation

- Code Check with coder.screener
  - Using coder.screener, check before run MATLAB Coder APP

    >> coder.screener('filename')
  - Check each user function to see the problem area



Code Generation Readiness - foo1.m etc.

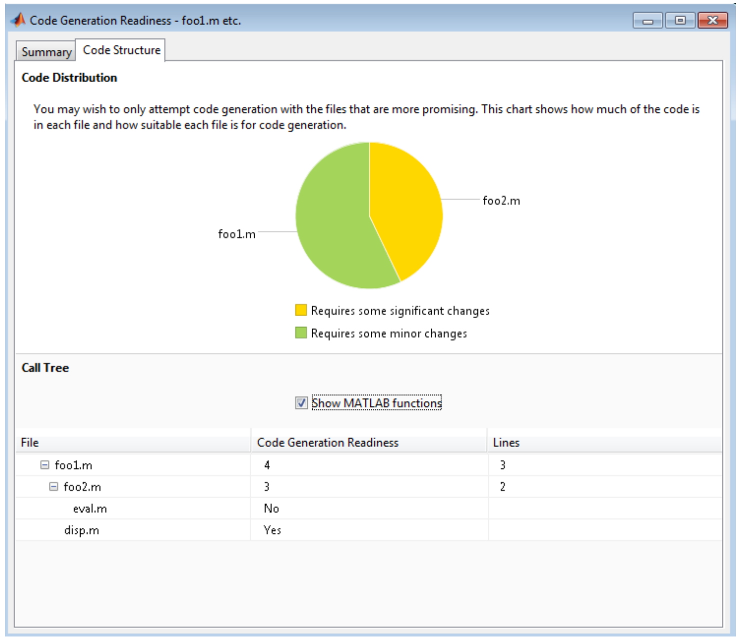
Code Generation Readiness Score: 4

Requires some minor changes

Code generation tools might fail unless the issues listed below are fixed.

Unsupported MATLAB function calls - 1 invocation

foo2.m → eval 1



Code Generation Readiness - foo1.m etc.

Code Distribution

You may wish to only attempt code generation with the files that are more promising. This chart shows how much of the code is in each file and how suitable each file is for code generation.

foo2.m
foo1.m
🟨 Requires some significant changes
🟩 Requires some minor changes

Call Tree

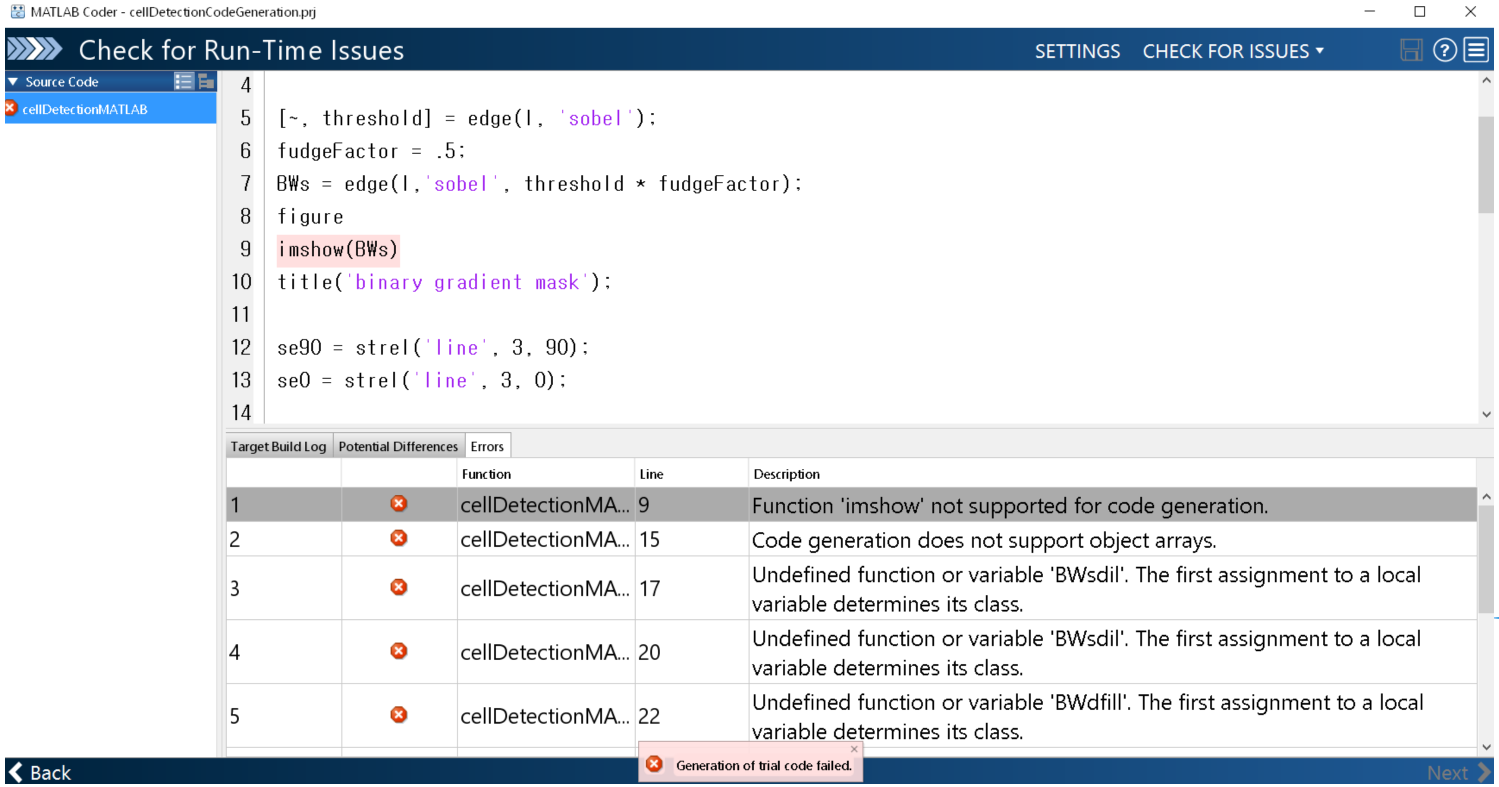
☑ Show MATLAB functions

File	Code Generation Readiness	Lines
⊟ foo1.m	4	3
⊟ foo2.m	3	2
eval.m	No	
disp.m	Yes	

<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



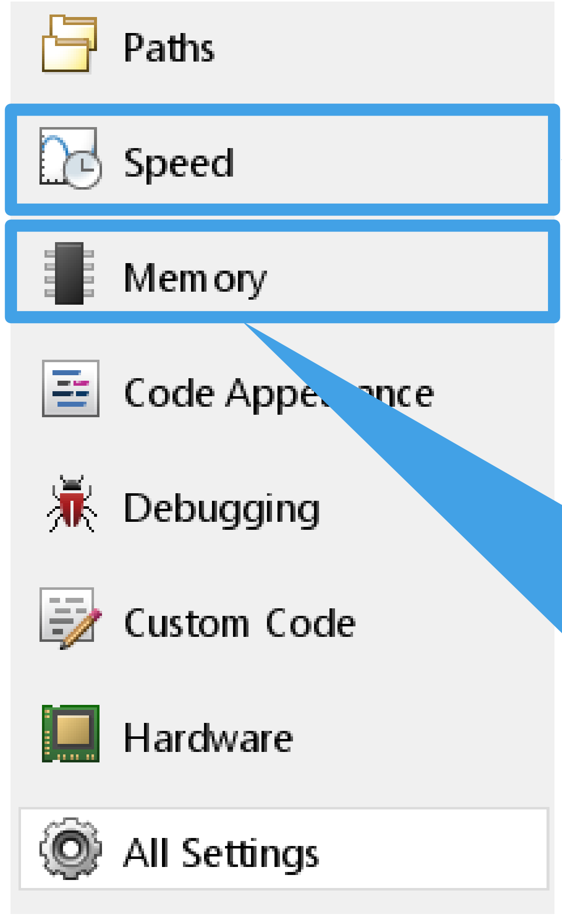
```
4
5   [~, threshold] = edge(I, 'sobel');
6   fudgeFactor = .5;
7   BWs = edge(I,'sobel', threshold * fudgeFactor);
8   figure
9   imshow(BWs)
10  title('binary gradient mask');
11
12  se90 = strel('line', 3, 90);
13  se0 = strel('line', 3, 0);
14
```

		Function	Line	Description
1	☒	cellDetectionMA...	9	Function 'imshow' not supported for code generation.
2	☒	cellDetectionMA...	15	Code generation does not support object arrays.
3	☒	cellDetectionMA...	17	Undefined function or variable 'BWsdil'. The first assignment to a local variable determines its class.
4	☒	cellDetectionMA...	20	Undefined function or variable 'BWsdil'. The first assignment to a local variable determines its class.
5	☒	cellDetectionMA...	22	Undefined function or variable 'BWdfill'. The first assignment to a local variable determines its class.

☒ Generation of trial code failed.

Display Errors

Settings for Code Generation



Paths

Speed

Memory

Code Appearance

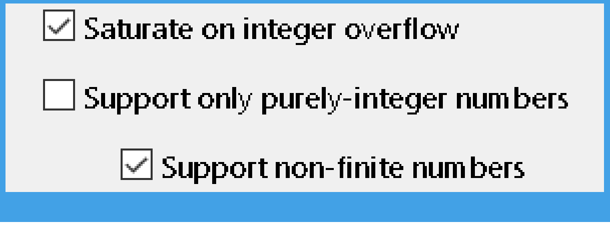
Debugging

Custom Code

Hardware

All Settings

- ✓ 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



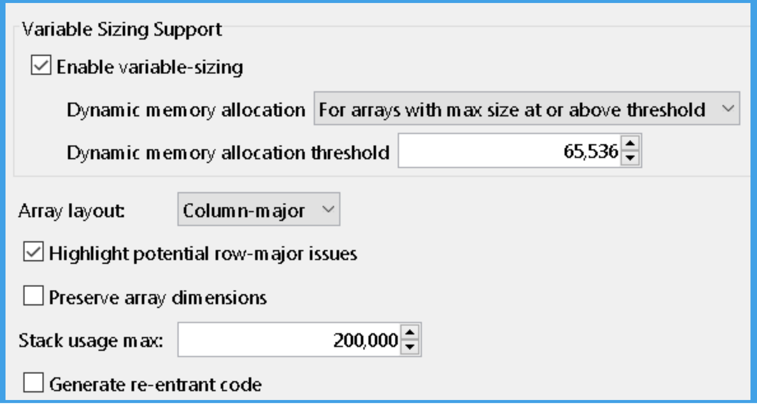
☑ Saturate on integer overflow

☐ Support only purely-integer numbers

☑ Support non-finite numbers

※Embedded Coder option

- ✓ 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



Variable Sizing Support

☑ Enable variable-sizing

Dynamic memory allocation: For arrays with max size at or above threshold

Dynamic memory allocation threshold: 65,536

Array layout: Column-major

☑ Highlight potential row-major issues

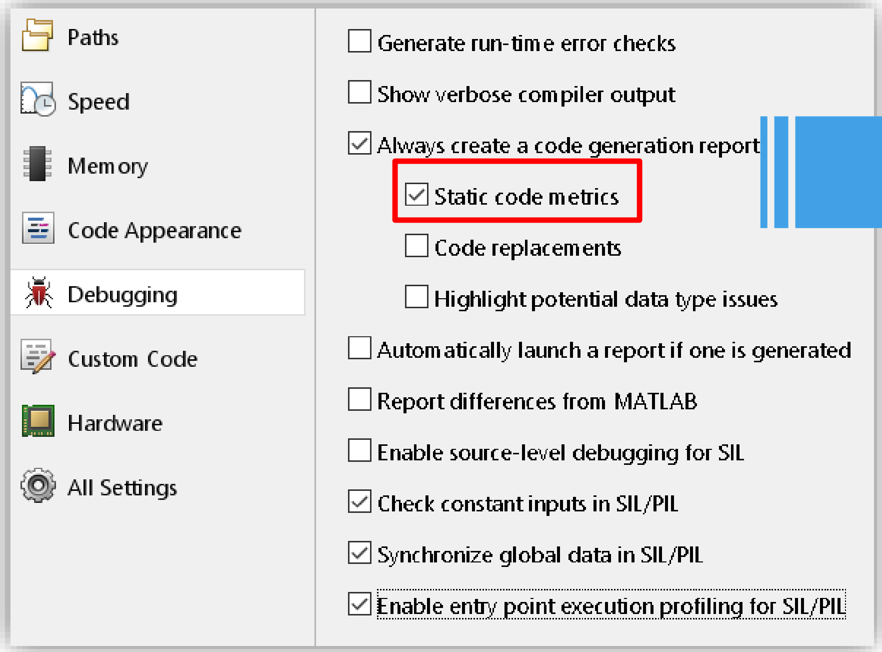
☐ Preserve array dimensions

Stack usage max: 200,000

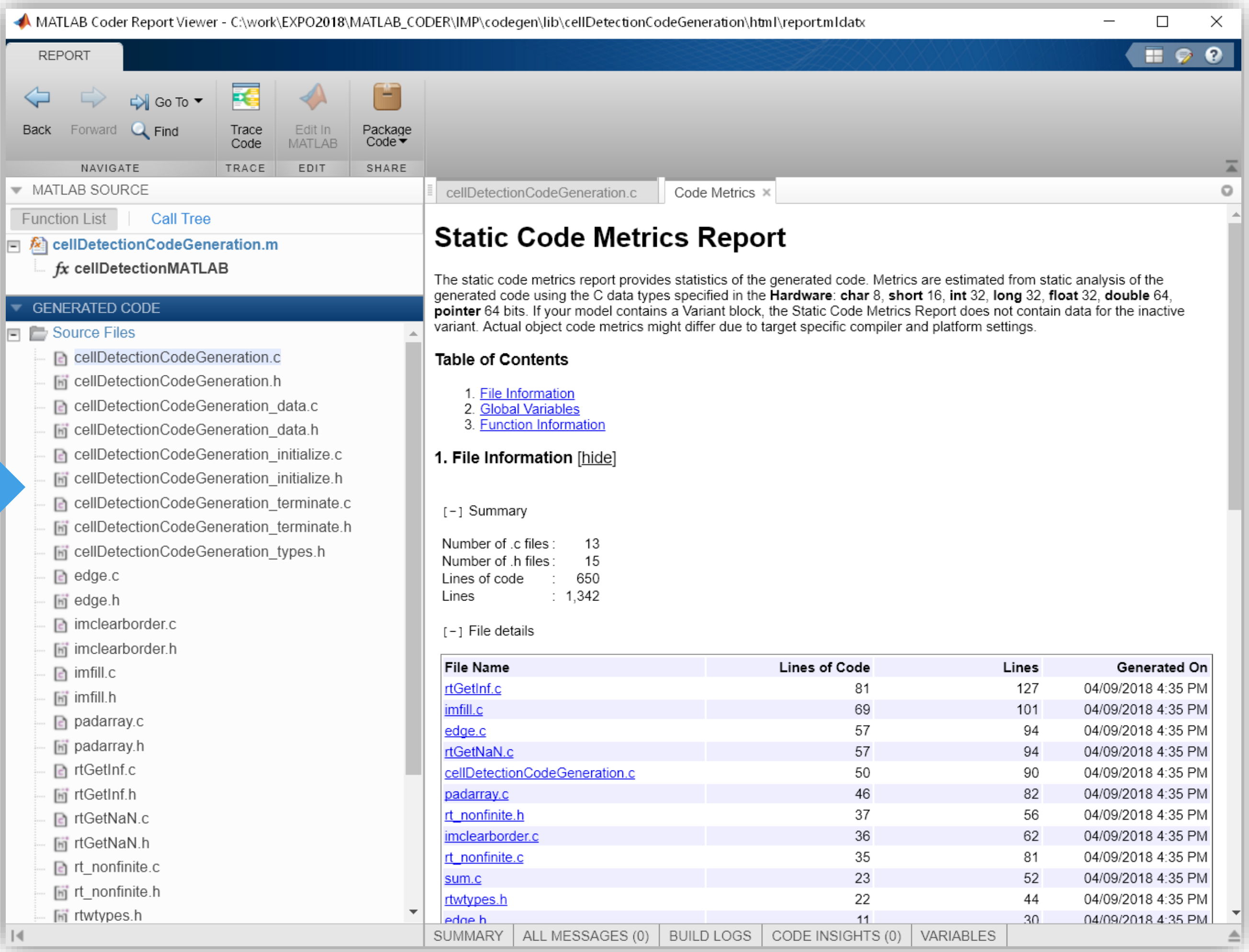
☐ Generate re-entrant code

Code Generation Report with Static code metrics

- File Information
  - Number of lines of code per file
  - Global variable, size (bytes)
  - Stack size



☐ Generate run-time error checks
☐ Show verbose compiler output
☑ Always create a code generation report
☑ Static code metrics
☐ Code replacements
☐ Highlight potential data type issues
☐ Automatically launch a report if one is generated
☐ Report differences from MATLAB
☐ Enable source-level debugging for SIL
☑ Check constant inputs in SIL/PIL
☑ Synchronize global data in SIL/PIL
☑ Enable entry point execution profiling for SIL/PIL



MATLAB Coder Report Viewer - C:\work\EXPO2018\MATLAB_CODER\IMP\codegen\lib\cellDetectionCodeGeneration\html\report.mldatx
REPORT
Back
Forward
Go To
Find
Trace Code
Edit In MATLAB
Package Code
NAVIGATE
TRACE
EDIT
SHARE
MATLAB SOURCE
Function List
Call Tree
cellDetectionCodeGeneration.m
cellDetectionMATLAB
GENERATED CODE
Source Files
cellDetectionCodeGeneration.c
cellDetectionCodeGeneration.h
cellDetectionCodeGeneration_data.c
cellDetectionCodeGeneration_data.h
cellDetectionCodeGeneration_initialize.c
cellDetectionCodeGeneration_initialize.h
cellDetectionCodeGeneration_terminate.c
cellDetectionCodeGeneration_terminate.h
cellDetectionCodeGeneration_types.h
edge.c
edge.h
imclearborder.c
imclearborder.h
imfill.c
imfill.h
padarray.c
padarray.h
rtGetInf.c
rtGetInf.h
rtGetNaN.c
rtGetNaN.h
rt_nonfinite.c
rt_nonfinite.h
rtwtypes.h
cellDetectionCodeGeneration.c
Code Metrics
SUMMARY
ALL MESSAGES (0)
BUILD LOGS
CODE INSIGHTS (0)
VARIABLES

Static Code Metrics Report

The static code metrics report provides statistics of the generated code. Metrics are estimated from static analysis of the generated code using the C data types specified in the **Hardware**: **char** 8, **short** 16, **int** 32, **long** 32, **float** 32, **double** 64, **pointer** 64 bits. If your model contains a Variant block, the Static Code Metrics Report does not contain data for the inactive variant. Actual object code metrics might differ due to target specific compiler and platform settings.

Table of Contents

1. File Information
2. Global Variables
3. Function Information

1. File Information [hide]

[−] Summary

```
Number of .c files :    13
Number of .h files :    15
Lines of code      :   650
Lines              : 1,342
```

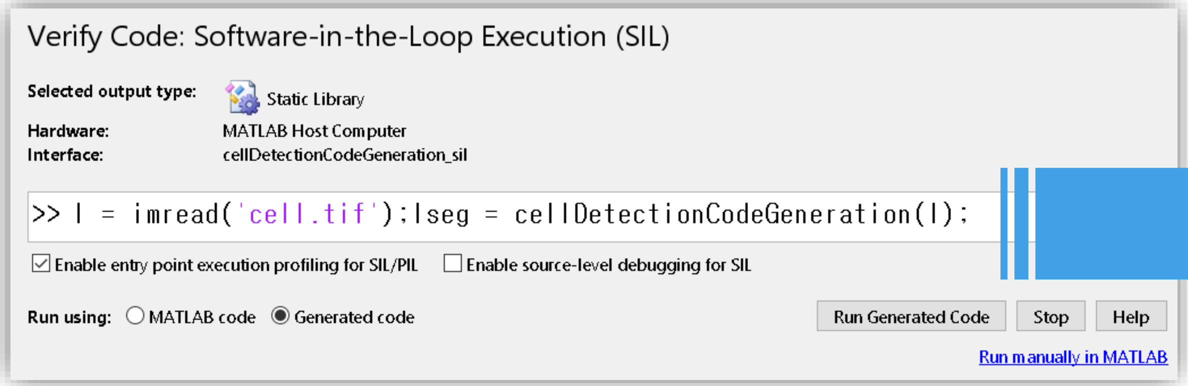
[−] File details

File Name	Lines of Code	Lines	Generated On
rtGetInf.c	81	127	04/09/2018 4:35 PM
imfill.c	69	101	04/09/2018 4:35 PM
edge.c	57	94	04/09/2018 4:35 PM
rtGetNaN.c	57	94	04/09/2018 4:35 PM
cellDetectionCodeGeneration.c	50	90	04/09/2018 4:35 PM
padarray.c	46	82	04/09/2018 4:35 PM
rt_nonfinite.h	37	56	04/09/2018 4:35 PM
imclearborder.c	36	62	04/09/2018 4:35 PM
rt_nonfinite.c	35	81	04/09/2018 4:35 PM
sum.c	23	52	04/09/2018 4:35 PM
rtwtypes.h	22	44	04/09/2018 4:35 PM
edge.h	11	30	04/09/2018 4:35 PM

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



Verify Code: Software-in-the-Loop Execution (SIL)

Selected output type: Static Library

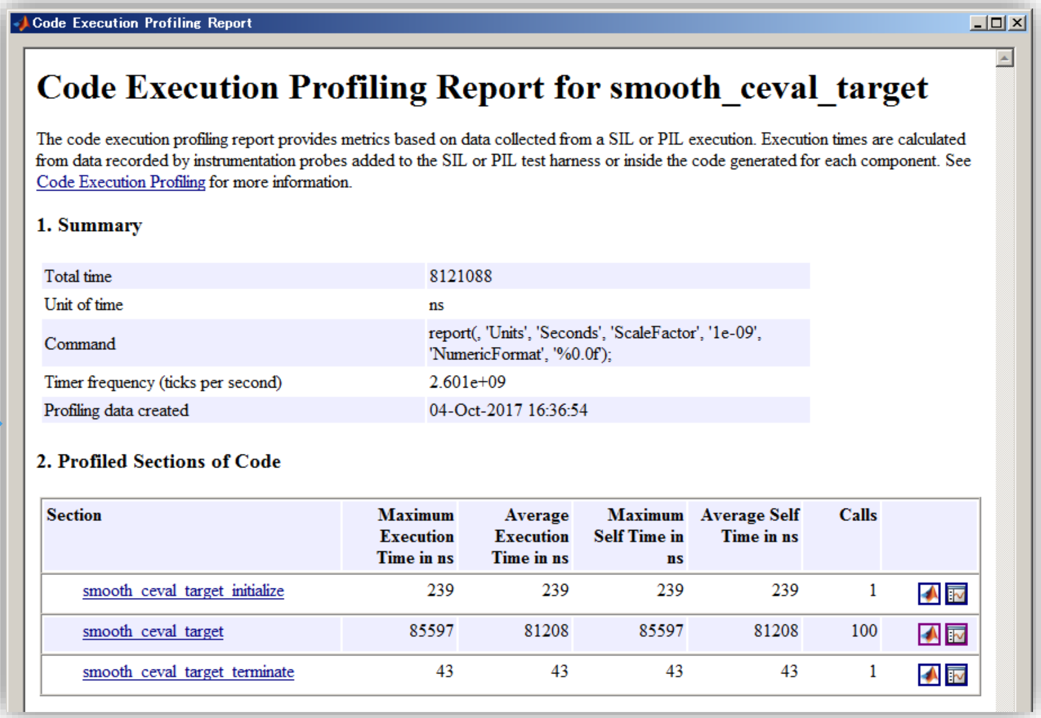
Hardware: MATLAB Host Computer

Interface: cellDetectionCodeGeneration_sil

```
>> I = imread('cell.tif');Iseg = cellDetectionCodeGeneration(I);
```

☑ Enable entry point execution profiling for SIL/PIL ☐ Enable source-level debugging for SIL

Run using: ○ MATLAB code ● Generated code



Code Execution Profiling Report

Code Execution Profiling Report for smooth_ceval_target

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	8121088
Unit of time	ns
Command	report(, 'Units', 'Seconds', 'ScaleFactor', '1e-09', 'NumericFormat', '%0.0f');
Timer frequency (ticks per second)	2.601e+09
Profiling data created	04-Oct-2017 16:36:54

2. Profiled Sections of Code

Section	Maximum Execution Time in ns	Average Execution Time in ns	Maximum Self Time in ns	Average Self Time in ns	Calls	
smooth_ceval_target_initialize	239	239	239	239	1	
smooth_ceval_target	85597	81208	85597	81208	100	
smooth_ceval_target_terminate	43	43	43	43	1	

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

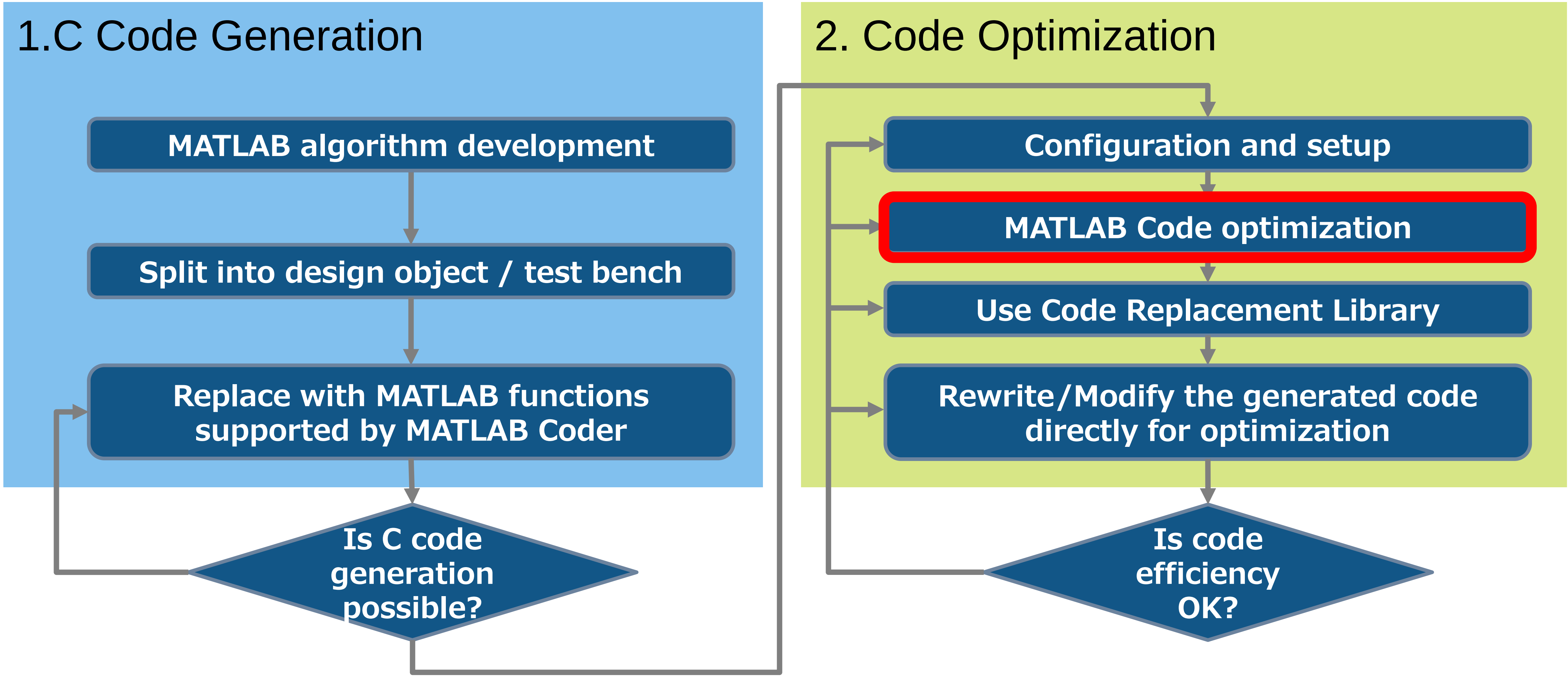
Key Takeaways

MATLAB Coder Basic

Coding techniques for efficient C code generation

Code Replacement Library and System Object

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

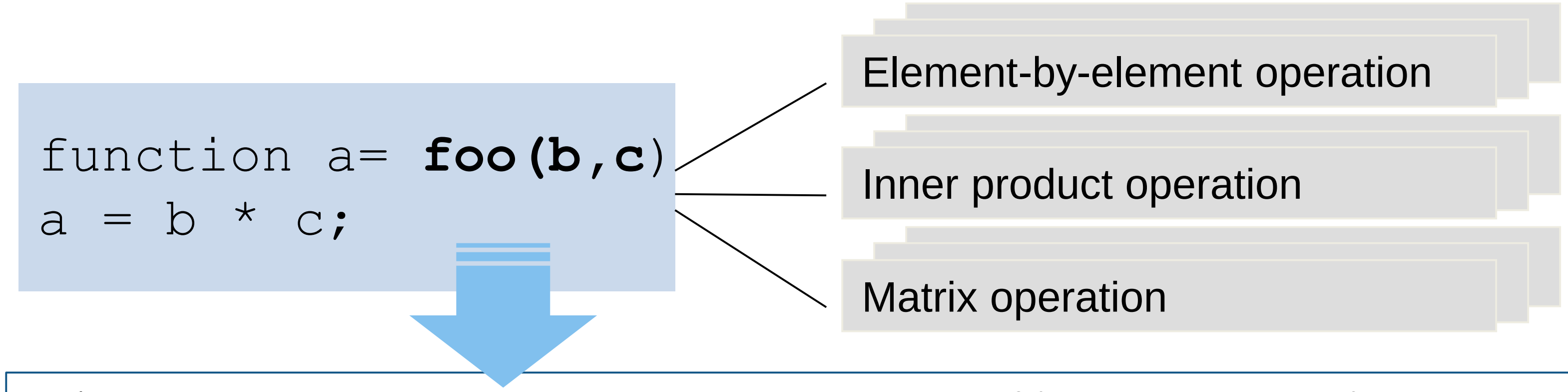
Use Code Replacement Library

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;
```

- Element-by-element operation
- Inner product operation
- Matrix operation

- Integer
- floating point number
- Fixed point number
- Real number
- complex number…

```
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

```
function C=SeriesFunc(A,B,n)

C=zeros(size(A));



for i=1:n
    C=C+inv(B)*A^i*B;
end
```

After improvement

```
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

```
function a = test_for(r)    %#codegen
a=ones(10,256);

for i=1:10
    a(i,:)=real(fft(r(i,:)));
end
```

After improvement

```
function a = test_for(r)    %#codegen
a=ones(10,256);

parfor i=1:10
    a(i,:)=real(fft(r(i,:)));
end
```

Run parallel with parfor

```
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++) {
  .....
}
```

Add pragmas for OpenMP

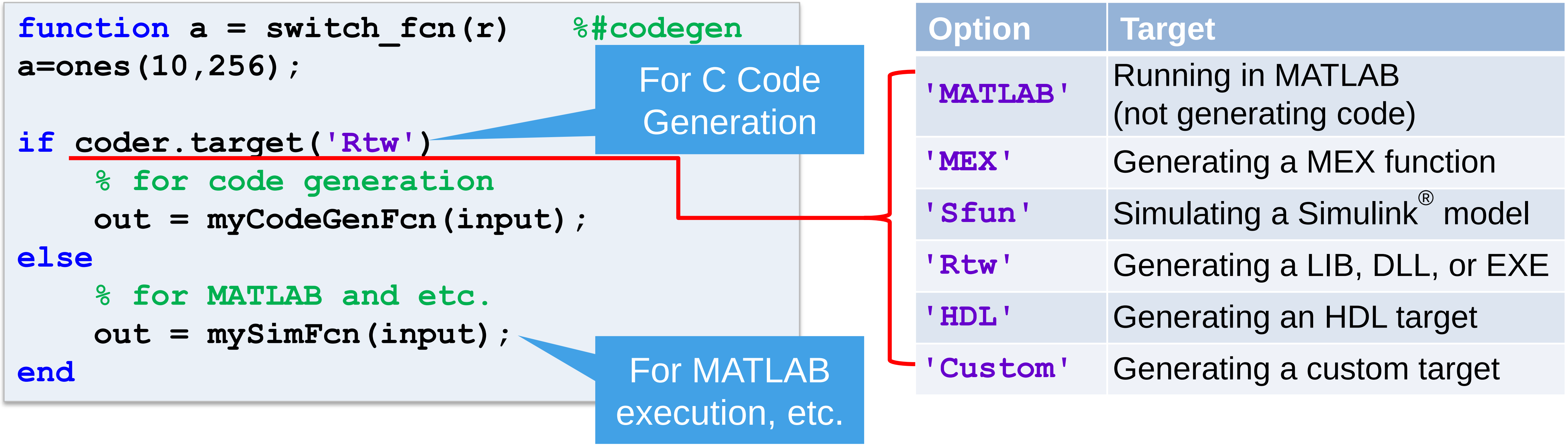
※ Parfor is a command for parallel operation of Parallel Computing Toolbox

※ OpenMP is an API for C / C ++ parallel computing

MATLAB code optimization

Differentiate simulation and code generation behavior

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



```
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
```

For C Code Generation

For MATLAB execution, etc.

Option	Target
'MATLAB'	Running in MATLAB (not generating code)
'MEX'	Generating a MEX function
'Sfun'	Simulating a Simulink® model
'Rtw'	Generating a LIB, DLL, or EXE
'HDL'	Generating an HDL target
'Custom'	Generating a custom target

MATLAB code optimization

Eliminate Redundant Copies of Function Inputs

- Reusing Variables ➔ Reduce memory usage and execution time

Before improvement

```
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

```
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

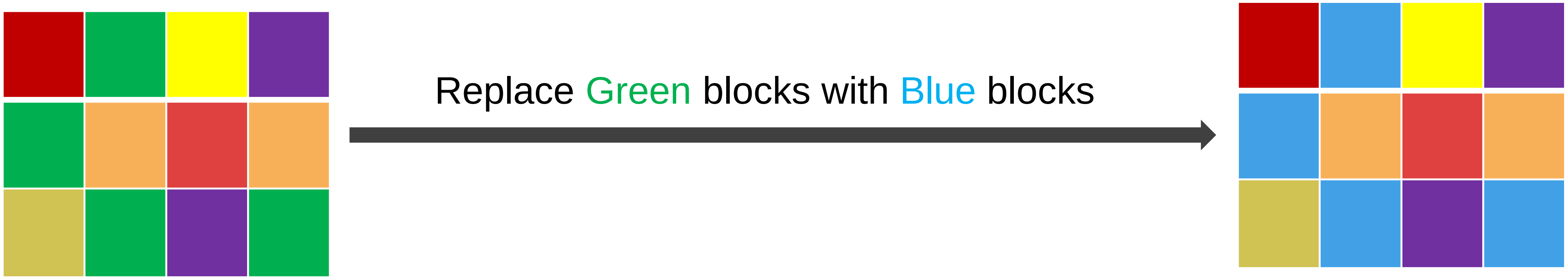
Key Takeaways

MATLAB Coder Basic

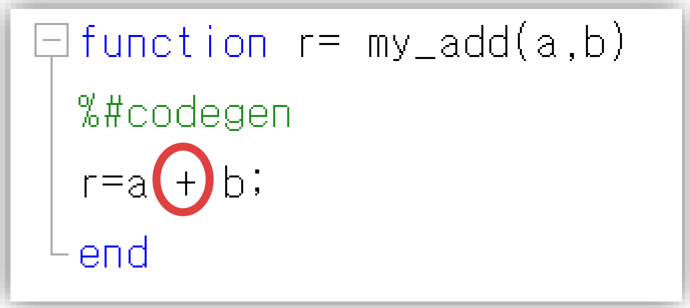
Coding techniques for efficient C code generation

Code Replacement Library and System Object

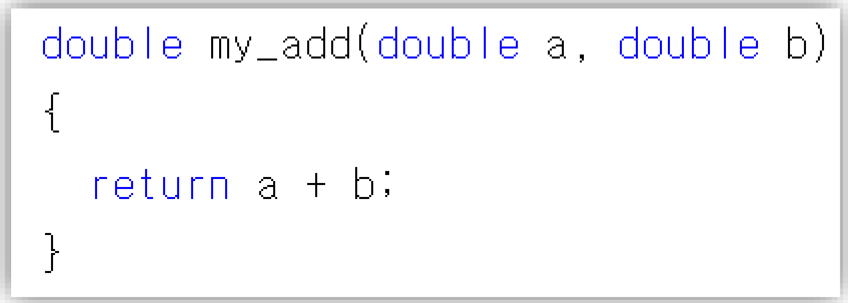
Concept of Code Replacement



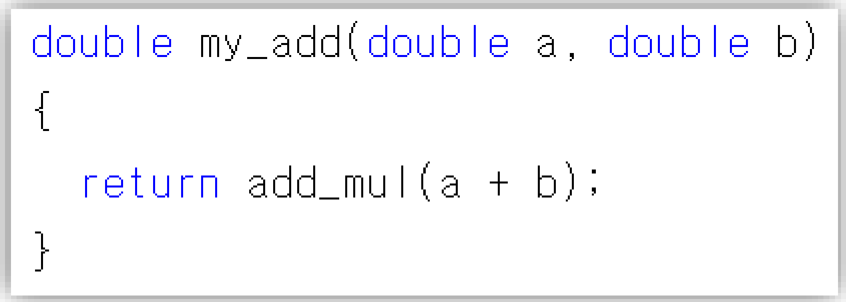
Replace Green blocks with Blue blocks



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



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



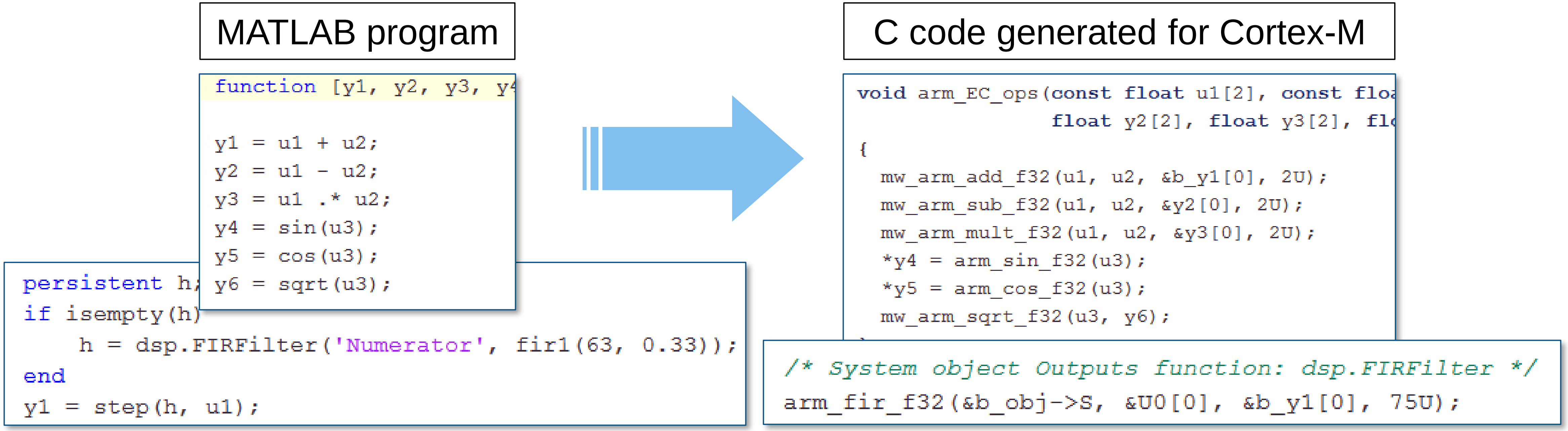
```
double my_add(double a, double b)
{
  return add_mul(a + b);
}
```

Replace ‘+’ with custom function ‘add_mul’

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

```
function [y1, y2, y3, y4

y1 = u1 + u2;
y2 = u1 - u2;
y3 = u1 .* u2;
y4 = sin(u3);
y5 = cos(u3);
y6 = sqrt(u3);
```

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

C code generated for Cortex-M

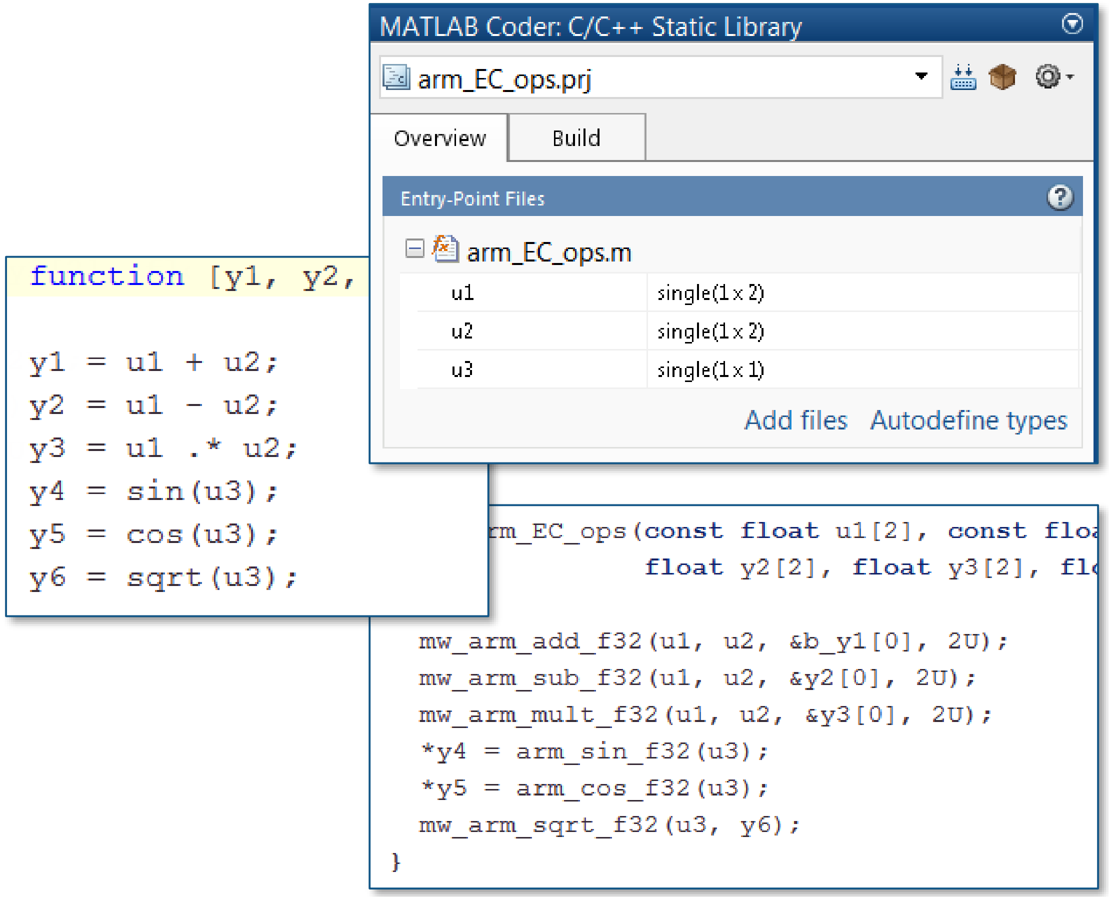
```
void arm_EC_ops(const float u1[2], const floa
                float y2[2], float y3[2], fl
{
  mw_arm_add_f32(u1, u2, &b_y1[0], 2U);
  mw_arm_sub_f32(u1, u2, &y2[0], 2U);
  mw_arm_mult_f32(u1, u2, &y3[0], 2U);
  *y4 = arm_sin_f32(u3);
  *y5 = arm_cos_f32(u3);
  mw_arm_sqrt_f32(u3, y6);
```

```
/* System object Outputs function: dsp.FIRFilter */
arm_fir_f32(&b_obj->S, &U0[0], &b_y1[0], 75U);
```

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()`



```
function [y1, y2,

y1 = u1 + u2;
y2 = u1 - u2;
y3 = u1 .* u2;
y4 = sin(u3);
y5 = cos(u3);
y6 = sqrt(u3);
```

MATLAB Coder: C/C++ Static Library

arm_EC_ops.prj

Overview | Build

Entry-Point Files

arm_EC_ops.m

u1	single(1 x 2)
u2	single(1 x 2)
u3	single(1 x 1)

Add files   Autodefine types

```
rm_EC_ops(const float u1[2], const floa
                    float y2[2], float y3[2], fl

  mw_arm_add_f32(u1, u2, &b_y1[0], 2U);
  mw_arm_sub_f32(u1, u2, &y2[0], 2U);
  mw_arm_mult_f32(u1, u2, &y3[0], 2U);
  *y4 = arm_sin_f32(u3);
  *y5 = arm_cos_f32(u3);
  mw_arm_sqrt_f32(u3, y6);
}
```

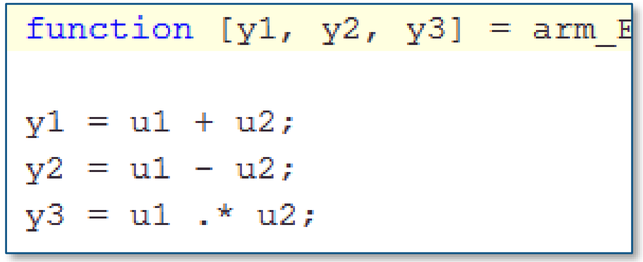
[Detailed listing here]

ARM Cortex-A Optimized Code

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

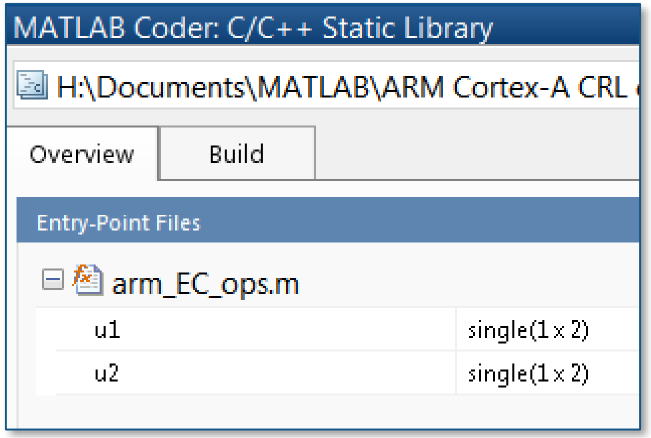
- Replace basic vector operations with calls to NEON SIMD code, such as:
  - `ne10_add_float_neon(), ne10_sub_float_neon(), ne10_mul_float_neon(), ne10_divc_float_neon(), ne10_abs_float_neon`

- Map code replacements to libraries: [Ne10 open software project]
  - *A set of common, heavily optimized functions for ARM Neon Architecture*



```
function [y1, y2, y3] = arm_E

y1 = u1 + u2;
y2 = u1 - u2;
y3 = u1 .* u2;
```



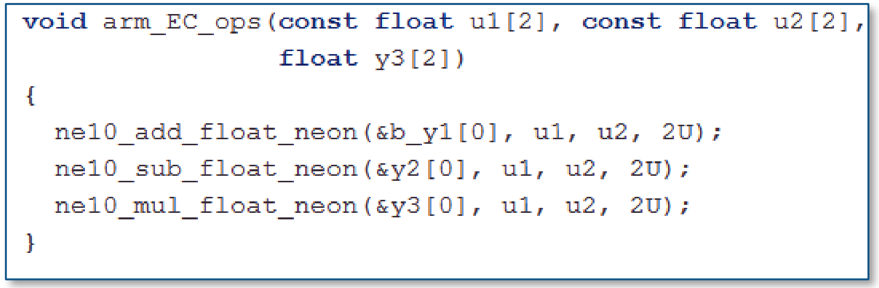
MATLAB Coder: C/C++ Static Library

H:\Documents\MATLAB\ARM Cortex-A CRL

Entry-Point Files

arm_EC_ops.m

u1	single(1 x 2)
u2	single(1 x 2)

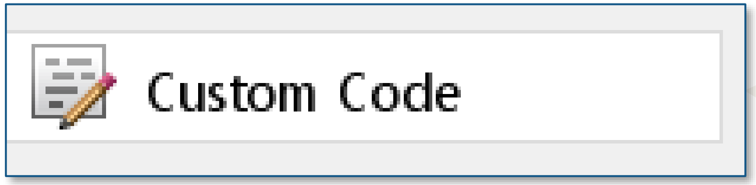


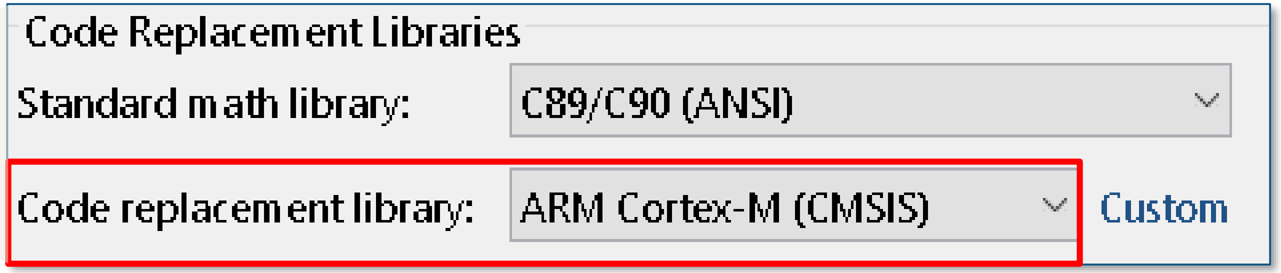
```
void arm_EC_ops(const float u1[2], const float u2[2],
                float y3[2])
{
  ne10_add_float_neon(&b_y1[0], u1, u2, 2U);
  ne10_sub_float_neon(&y2[0], u1, u2, 2U);
  ne10_mul_float_neon(&y3[0], u1, u2, 2U);
}
```

How to Apply Code Replacement Library

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

2. Set custom code in settings



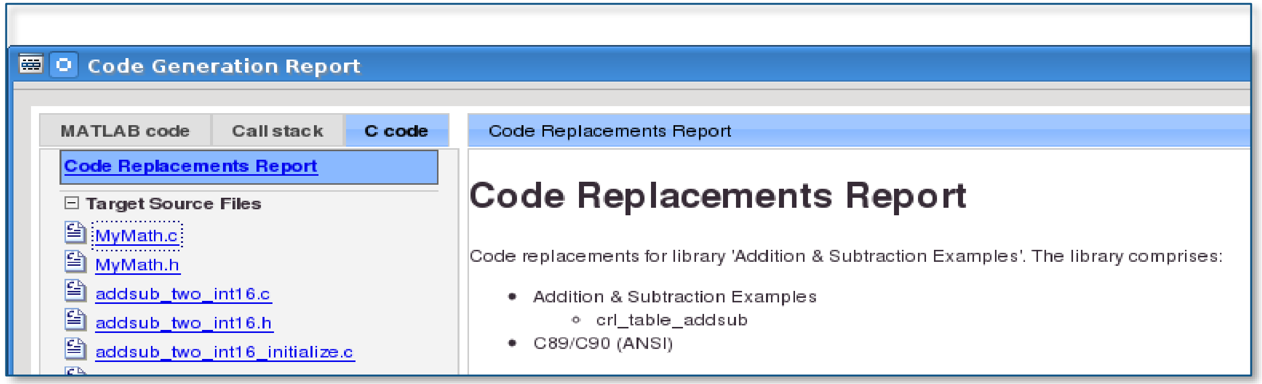


Code Replacement Libraries

Standard math library: C89/C90 (ANSI)

Code replacement library: ARM Cortex-M (CMSIS)

3. Confirm the Result of replacement from the report



Code Replacements Report

Code replacements for library 'Addition & Subtraction Examples'. The library comprises:

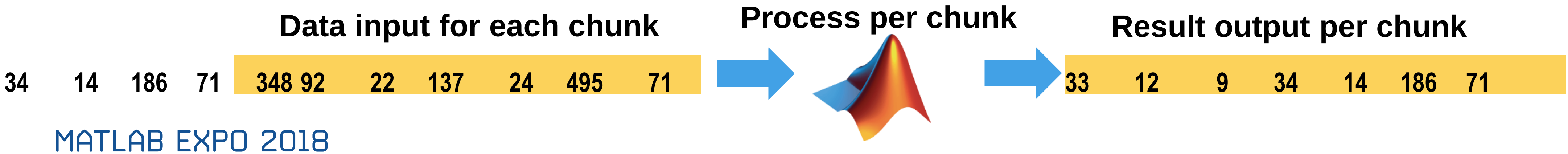
- Addition & Subtraction Examples
  - crl_table_addsub
- C89/C90 (ANSI)

What is System Object?

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

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

	System Object	MATLAB Function
Processing form	Streaming	Batch
Memory consumption	Small	Big
Simulink compatible	Almost all	Some
State management	Simple	User Setting
C Code Generation	Almost all	Some
HDL Code Generation	High abstraction level object correspondence	Not supported



Data input for each chunk

34 14 186 71 348 92 22 137 24 495 71

Process per chunk

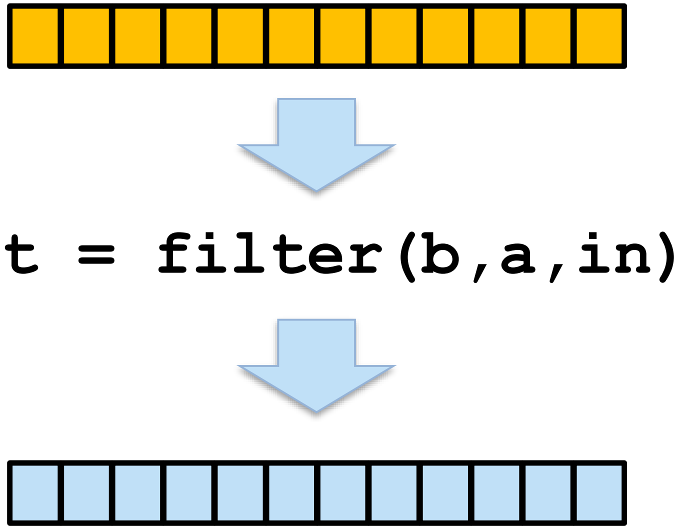
Result output per chunk

33 12 9 34 14 186 71

MATLAB, System Object and Simulink

MATLAB

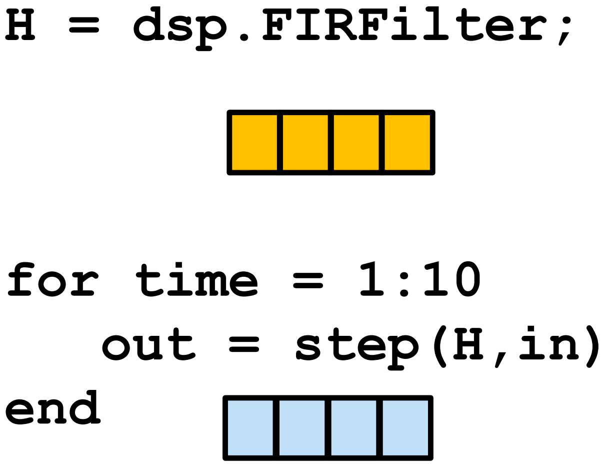
Frame data
Batch processing
code



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

System Object

Frame / sample data
Streaming process
Code / block

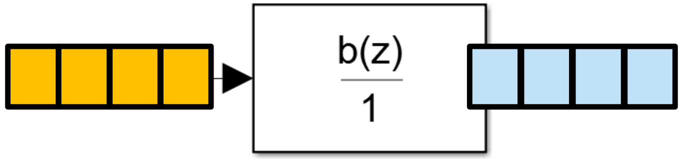


```
H = dsp.FIRFilter;

for time = 1:10
   out = step(H,in)
end
```

Simulink

Frame / sample data
Streaming process
block



Discrete FIR Filter

Supported System Objects for Code Replacement Library

CMSIS / Ne 10 library support

System Object	Cortex-M CMSIS	Cortex-A Ne10
dsp.FIRFilter	✓	✓
dsp.FIRDecimator	✓	✓
dsp.FIRInterpolator	✓	✓
dsp.LMSFilter	✓	
dsp.BiquadFilter	✓	
dsp.FFT	✓	✓
dsp.IFFT	✓	✓
dsp.Convolver	✓	
dsp.Crosscorrelator	✓	
dsp.Mean	✓	
dsp.RMS	✓	
dsp.StandardDeviation	✓	

System Object	Cortex-M CMSIS	Cortex-A Ne10
dsp.VariableBandwidthFIRFilter	✓	✓
dsp.FIRHalfbandInterpolator	✓	✓
dsp.FIRHalfbandDecimator	✓	✓
dsp.CICCompensationDecimator	✓	✓
dsp.CICCompensationInterpolator	✓	✓
dsp.DigitalDownConverter	✓	✓
dsp.DigitalUpConverter	✓	✓
dsp.SampleRateConverter	✓	✓
dsp.Variance	✓	

Display code replacement library list

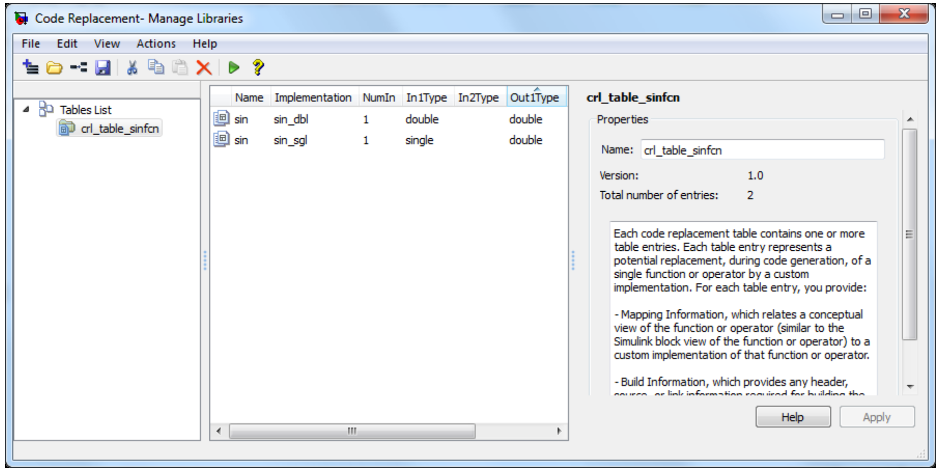
```
>> crviewer
```

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.



Name	Implementation	NumIn	In1Type	In2Type	Out1Type
sin	sin_dbl	1	double		double
sin	sin_sgl	1	single		double

crl_table_sinfcn

Properties

Name: crl_table_sinfcn

Version: 1.0

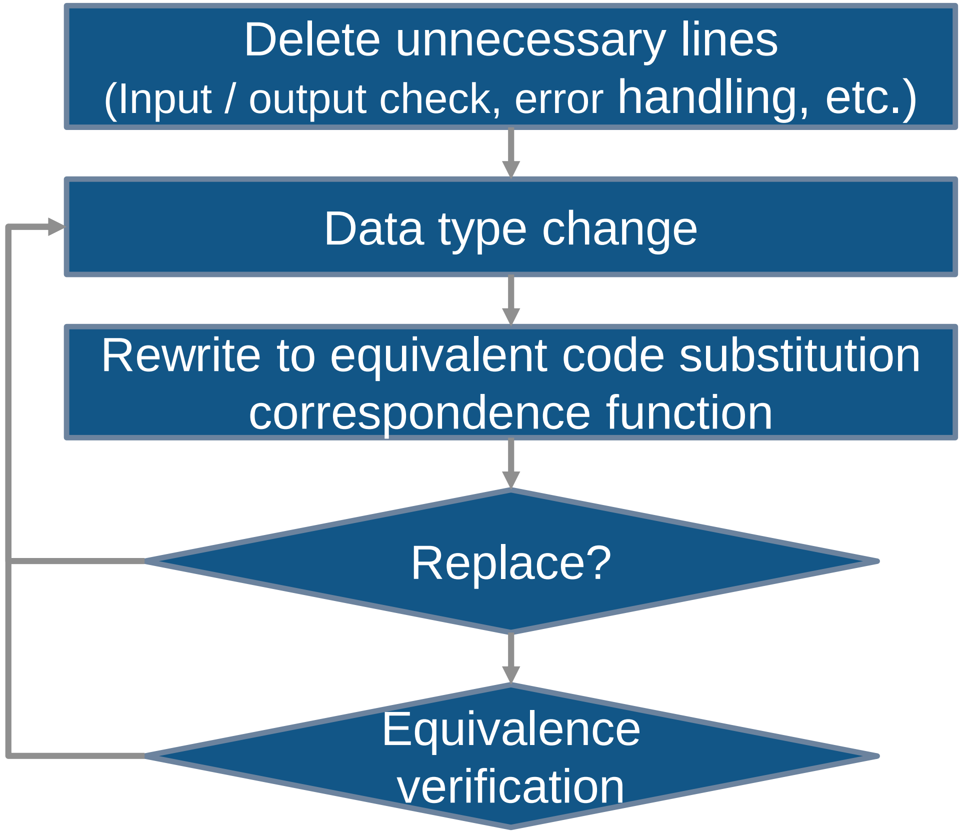
Total number of entries: 2

Each code replacement table contains one or more table entries. Each table entry represents a potential replacement, during code generation, of a single function or operator by a custom implementation. For each table entry, you provide:

- Mapping Information, which relates a conceptual view of the function or operator (similar to the Simulink block view of the function or operator) to a custom implementation of that function or operator.

- Build Information, which provides any header,

Customize MATLAB functions



Delete unnecessary lines
(Input / output check, error handling, etc.)

Data type change

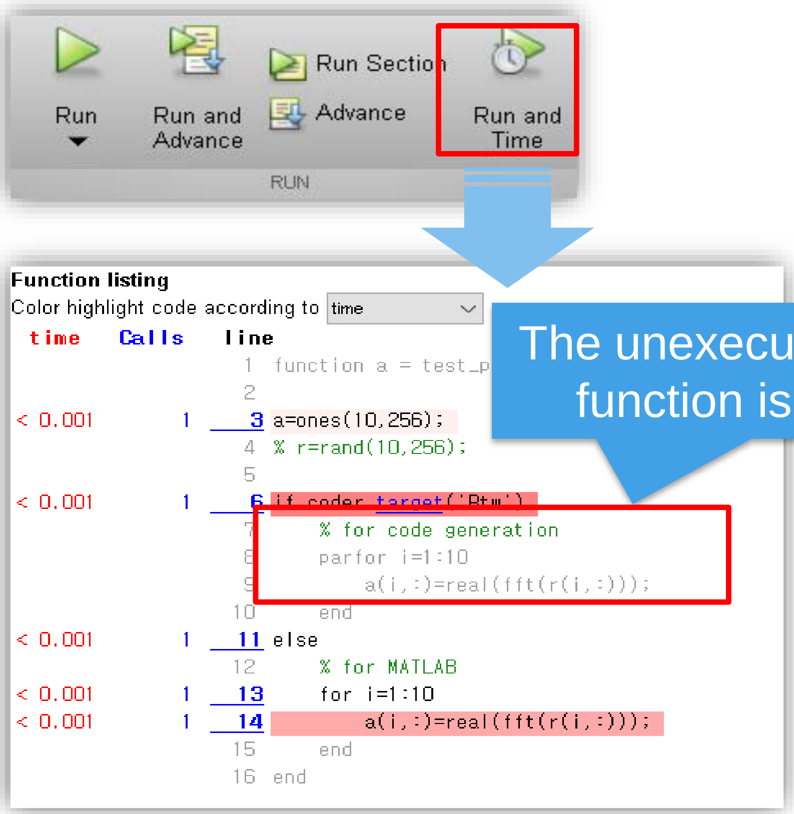
Rewrite to equivalent code substitution correspondence function

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

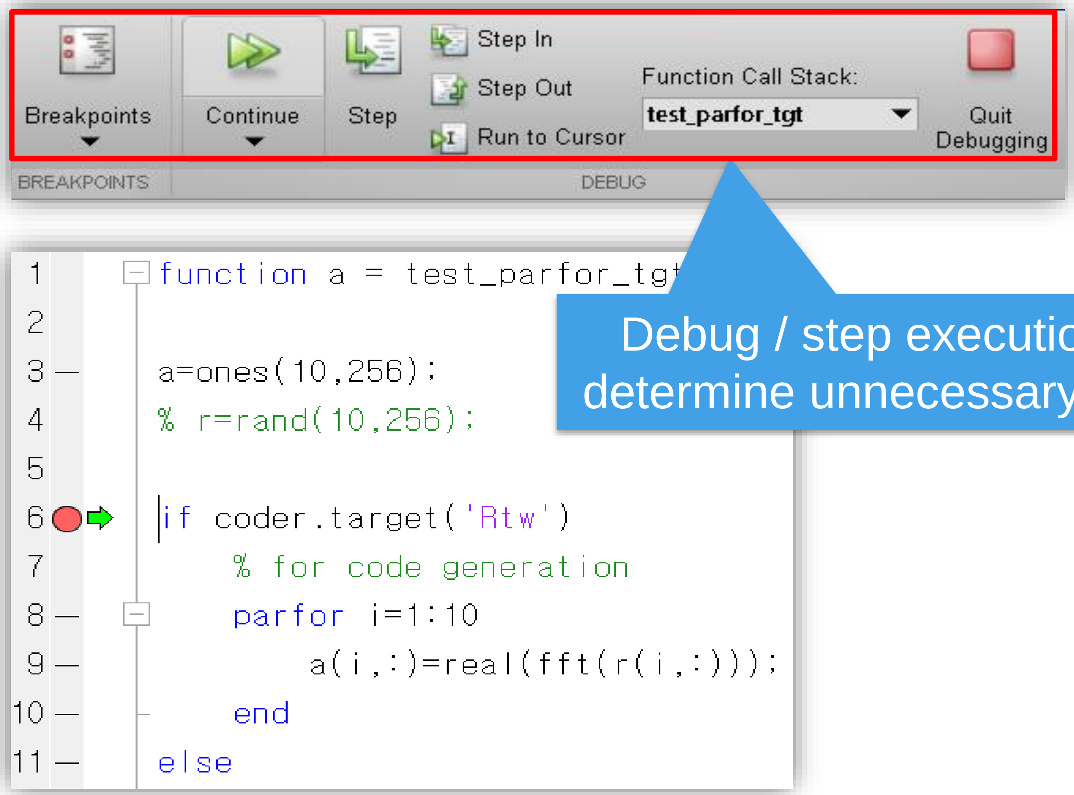


Function listing

Color highlight code according to time

```
 time      Calls    line
                     1  function a = test_p
                     2
< 0.001        1     3  a=ones(10,256);
                     4  % r=rand(10,256);
                     5
< 0.001        1     6  if coder.target('Rtw')
                     7      % for code generation
                     8      parfor i=1:10
                     9          a(i,:)=real(fft(r(i,:)));
                    10      end
< 0.001        1    11  else
                    12      % for MATLAB
< 0.001        1    13      for i=1:10
< 0.001        1    14          a(i,:)=real(fft(r(i,:)));
                    15      end
                    16  end
```

The unexecuted part in the function is grayed out



```
function a = test_parfor_tgt

a=ones(10,256);
% r=rand(10,256);

if coder.target('Rtw')
    % for code generation
    parfor i=1:10
        a(i,:)=real(fft(r(i,:)));
    end
else
```

Debug / step execution to determine unnecessary parts

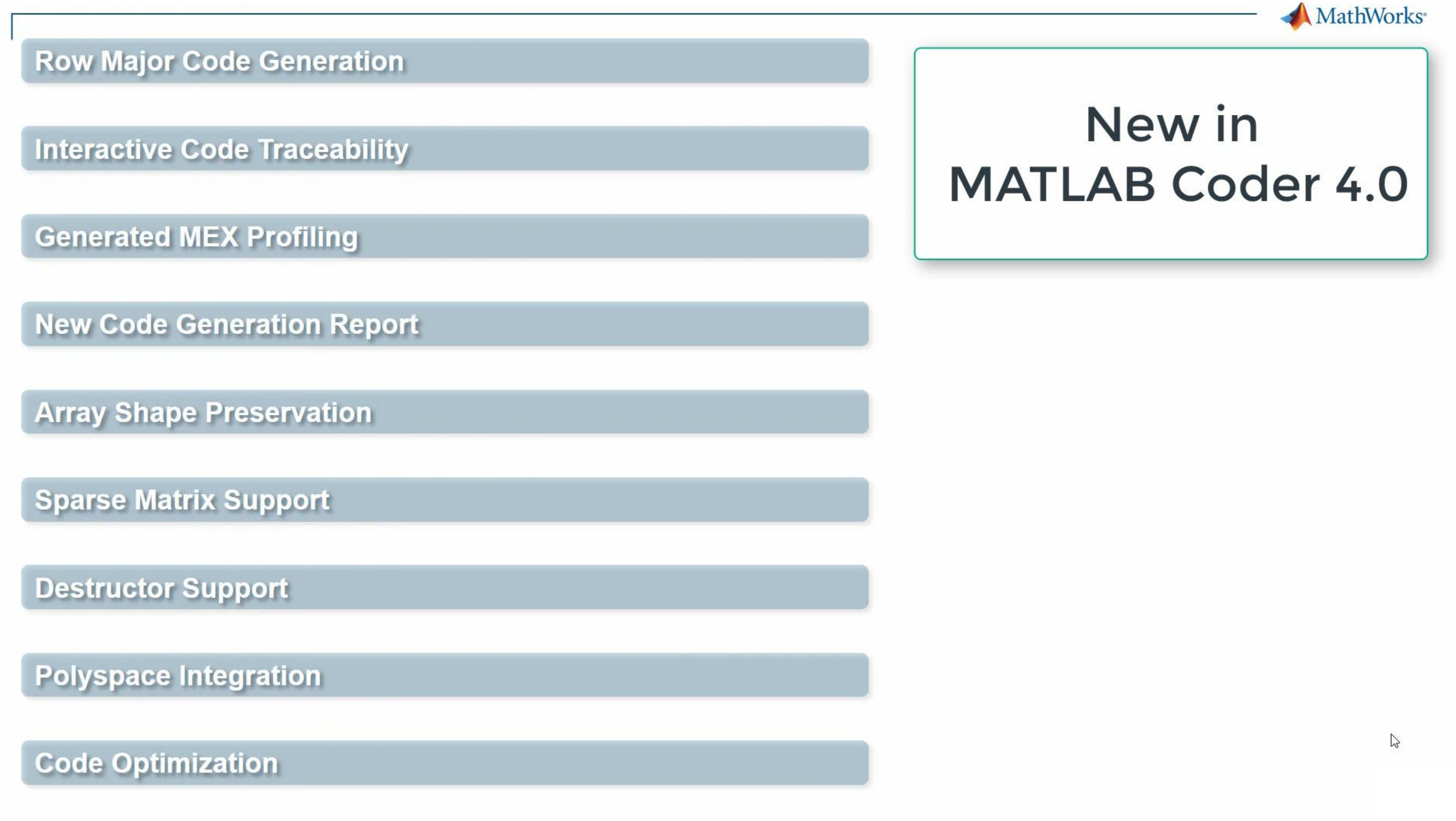
Key Takeaways

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