

MATLAB & SIMULINK DAY 2013

one place. one platform. endless possibilities.

Automatic C Code Generation from MATLAB

Marc Barberis – Application Engineering Group, MathWorks Inc.

Supported by



Southeast Asia's sole distributor of

**MATLAB®
& SIMULINK®**



AGENDA

- Quick Demo
- Benefits of Automatic C Code Generation
- In-Depth Example
- Comparison between MATLAB Coder and MATLAB Compiler
- Fixed-Point Design
- Conclusion

Demo: Using Generated C Code in a Stand-Alone C Project

The image displays two screenshots related to a C++ project. The left screenshot shows a SimpleGUI application window titled "SimpleGUI" with a "Zoom Factor" slider ranging from 1/4 to 4 and a green "Start" button. Below it is a "Zoomed" window showing a zoomed-in image of a road scene. The right screenshot shows the Microsoft Visual C++ 2010 Express IDE. The Solution Explorer on the left shows a project named "Zoom" with source files: ImageProcess.cpp, ImageProcess_emxAPI.cpp, ImageProcess_emxutil.cpp, ImageProcess_initialize.cpp, and main.cpp. The main editor window shows the code for ImageProcess.cpp, which includes a for loop for processing image data. The Output window at the bottom shows the build process, including the message "Generating Code..." and a linker error: "error LNK2019: unresolved external symbol...".

Why translate MATLAB to C?



Integrate MATLAB algorithms w/ existing C environment using source code or static libraries



Prototype MATLAB algorithms on desktops as standalone executables

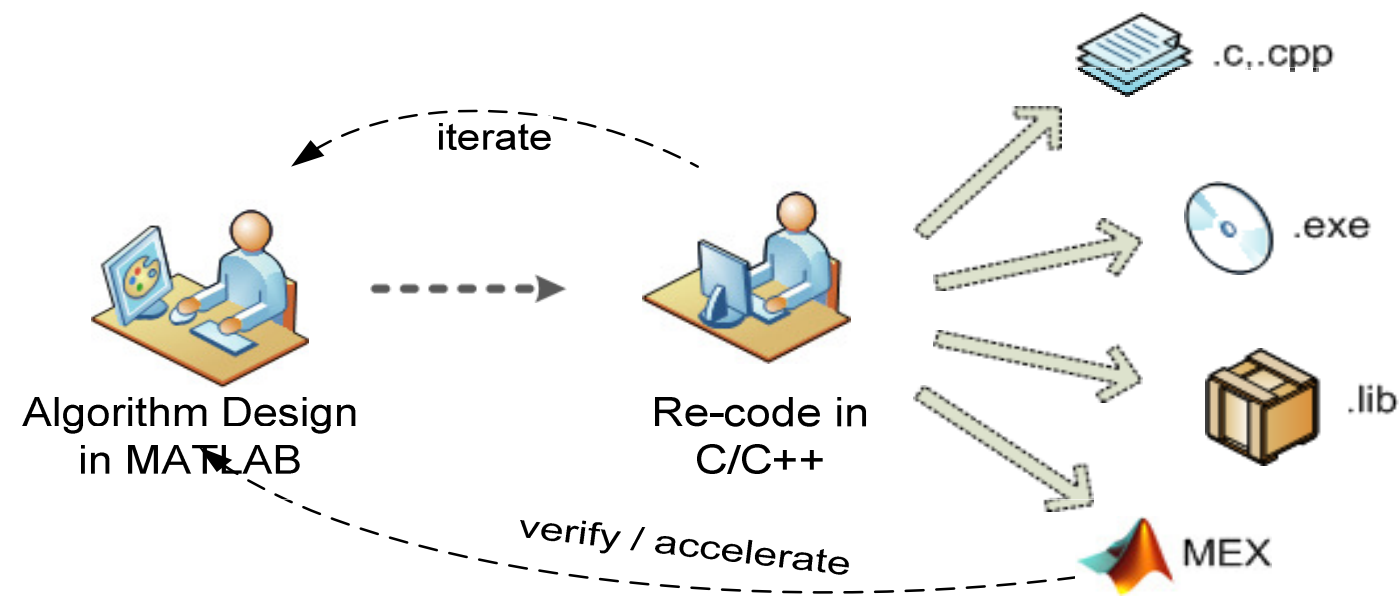


Accelerate user-written MATLAB algorithms



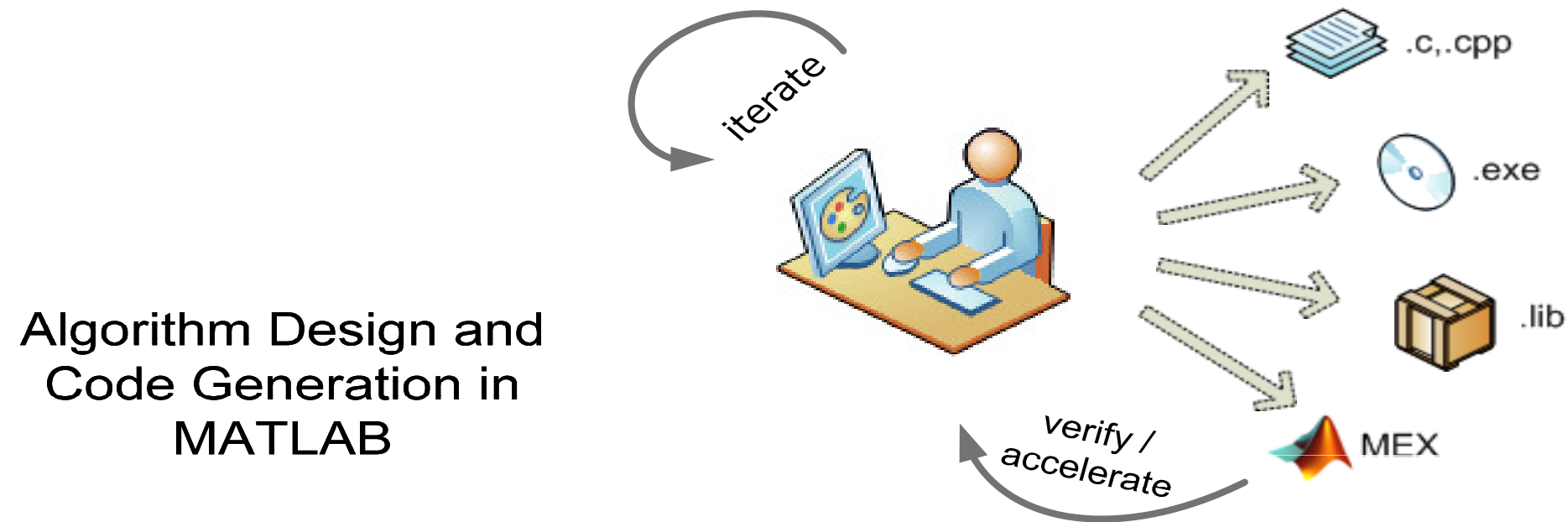
Implement C/C++ code on processors or hand-off to software engineers

Challenges with Manual Translation from MATLAB to C/C++



- Separate functional and implementation specification
 - Leads to multiple implementations that are inconsistent
 - Hard to modify requirements during development
 - Difficult to keep reference MATLAB code and C code in-sync
- Manual coding errors
- Time consuming and expensive

Automatic Translation of MATLAB to C



With MATLAB Coder, design engineers can

- 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

Implementation Constraints

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

Element by element multiply

Dot product

Matrix multiply

logical
integer
real
complex
...

C

```
double foo(double b, double c)
{
    return 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];
            }
        }
    }
}
```

Implementation Constraints

- Polymorphism
- Memory allocation
- Processing matrices & arrays
- Fixed-point data types

7 Lines of MATLAB
107 Lines of C

```
function [x_est p_est] = kalman_estimate(R,H,x_prd,p_prd,z)
S = H * p_prd' * H' + R;
B = H * p_prd';
klm_gain = (S \ B)';
x_est = x_prd + klm_gain * (z - H * x_prd);
p_est = p_prd - klm_gain * H * p_prd;
```

```
#include "kalman_estimate.h"
void kalman_estimate(const real_T R[4],
                    const real_T p_prd[6],
                    real_T x_est[6], real_T z[6])
{
    int32_T r1;
    int32_T r2;
    real_T klm_gain[12];
    int32_T k;
    real_T a21;
    real_T S[4];
    real_T B[12];
    real_T a22;
    real_T Y[12];
    real_T b_z[2];
    real_T b_klm_gain[36];
    for (r1 = 0; r1 < 2; r1++) {
        for (r2 = 0; r2 < 6; r2++) {
            klm_gain[r1 + (r2 << 1)] = 0.0;
```

```
for (r1 = 0; r1 < 2; r1++) {
    a21 = 0.0;
    for (r2 = 0; r2 < 6; r2++) {
        a21 += H[r1 + (r2 << 1)] * x_prd[r2];
    }
}
```

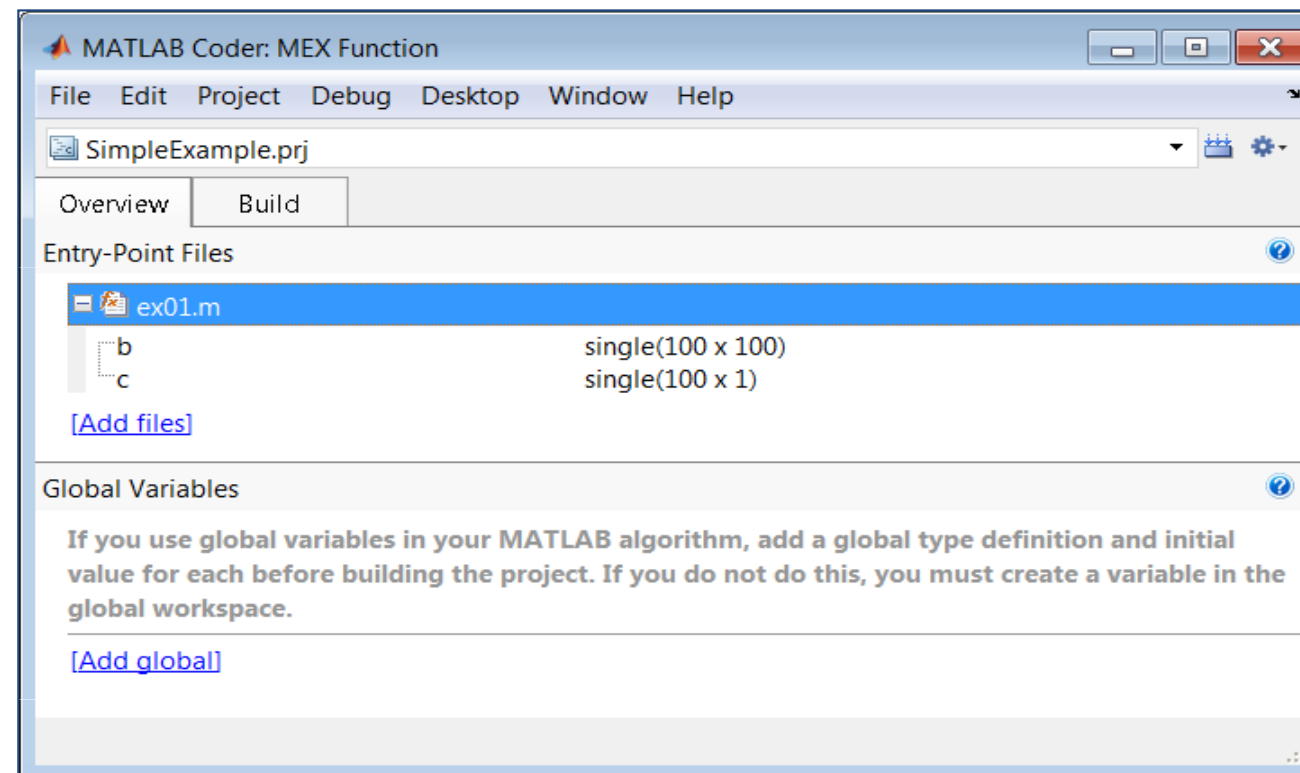
```
b_z[r1] = z[r1] - a21;
}
if (fabs(S[1]) > fabs(S[0])) {
    r1 = 2;
    r2 = 1;
} else {
    r1 = 1;
    r2 = 2;
}
a21 = S[r2 - 1] / S[r1 - 1];
a22 = S[r2 + 1] - a21 * S[r1 + 1];
for (k = 0; k < 6; k++) {
    Y[1 + (k << 1)] = (B[(r2 - 1) + (k << 1)] +
                    a22 *
                    Y[k << 1] - (B[(r1 - 1) + (k << 1)] +
                    1);
}
```

```
for (r1 = 0; r1 < 2; r1++) {
    for (r2 = 0; r2 < 6; r2++) {
        klm_gain[r2 + 6 * r1] = Y[r1 + (r2 << 1)];
    }
}
```

```
for (r1 = 0; r1 < 2; r1++) {
    for (r2 = 0; r2 < 2; r2++) {
        a21 = 0.0;
        for (k = 0; k < 6; k++) {
            a21 += klm_gain[r1 + (k << 1)] * H[r2 + (k << 1)];
        }
        S[r1 + (r2 << 1)] = a21 + R[r1 + (r2 << 1)];
    }
}
for (r1 = 0; r1 < 2; r1++) {
    for (r2 = 0; r2 < 6; r2++) {
        B[r1 + (r2 << 1)] = 0.0;
        for (k = 0; k < 6; k++) {
            B[r1 + (r2 << 1)] += H[r1 + (k << 1)] * p_prd[r2 + 6 * k];
        }
    }
}
```


In-Depth Demo of MATLAB Coder

- Coder UI
- Code Generation options
- Generate code
- Browse through report



Supported MATLAB Language Features and Functions

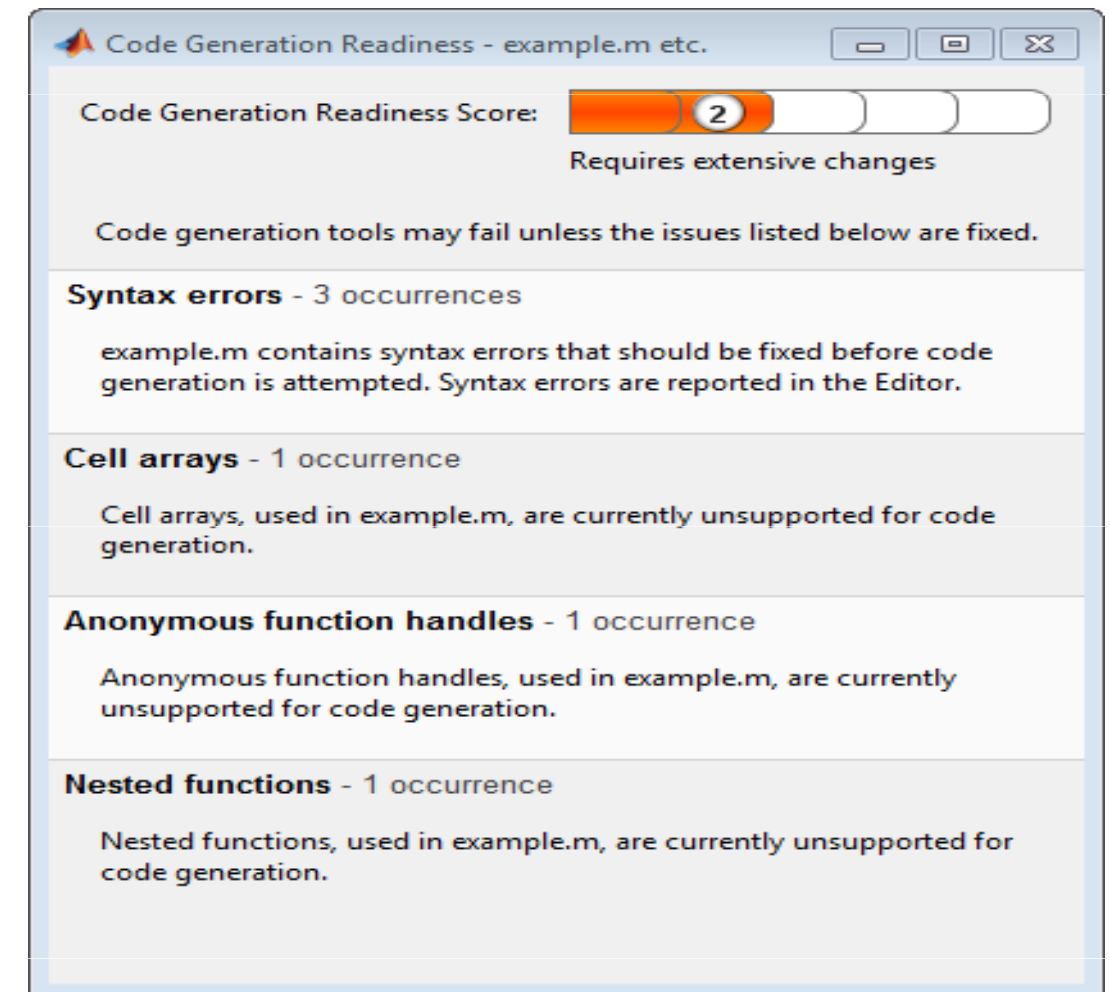
- Broad set of language features and functions/system objects supported for code generation

Matrices and Arrays	Data Types	Programming Constructs	Functions
<ul style="list-style-type: none"> • Matrix operations • N-dimensional arrays • Subscripting • Frames • Persistent variables • Global variables 	<ul style="list-style-type: none"> • Complex numbers • Integer math • Double/single-precision • Fixed-point arithmetic • Characters • Structures • Numeric classes • Variable-sized data • System objects • Classes 	<ul style="list-style-type: none"> • Arithmetic, relational, and logical operators • Program control (if, for, while, switch) 	<ul style="list-style-type: none"> • MATLAB functions and sub-functions • Variable length argument lists • Function handles <p>Supported algorithms</p> <ul style="list-style-type: none"> • > 400 MATLAB operators and functions • > 200 System objects for <ul style="list-style-type: none"> • Signal processing • Communications • Computer vision

Code Generation Readiness Tool

Instant feedback on code generation compliance of your MATLAB code

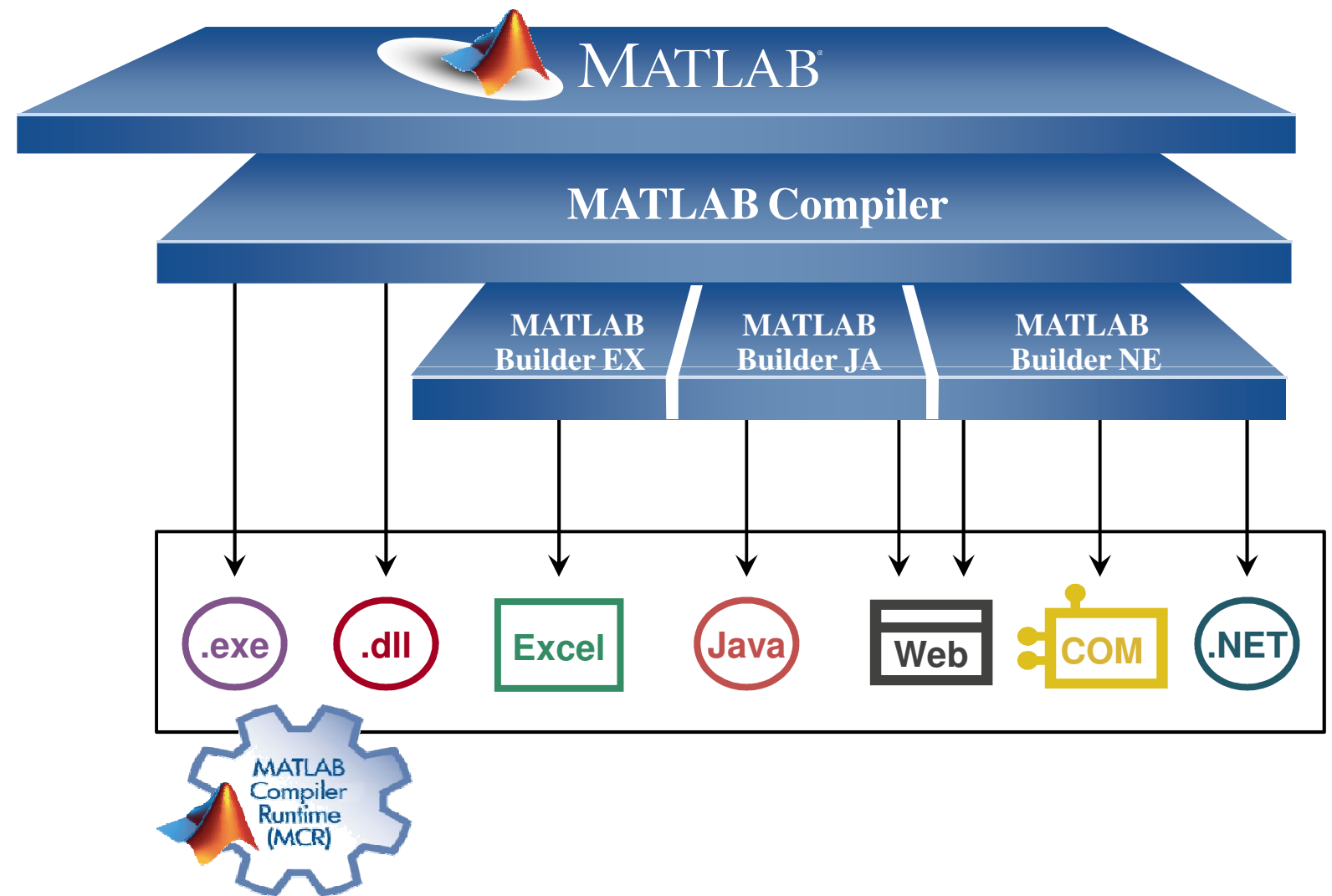
- Provides estimate of effort needed to generate C code from your MATLAB code on a scale of 1 to 5
- Provides a list of issues that need to be resolved in one report
- Gives detailed information on unsupported functions



Other Deployment Options

Deploying Applications with MATLAB Compiler

- Share applications
 - Desktop or Web software components
 - Supports full MATLAB language and most toolboxes
 - Requires MCR
 - Free run-time library
 - Royalty-free deployment

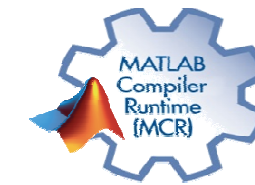


Choosing the Right Deployment Solution

MATLAB Coder and MATLAB Compiler



MATLAB Coder



MATLAB Compiler

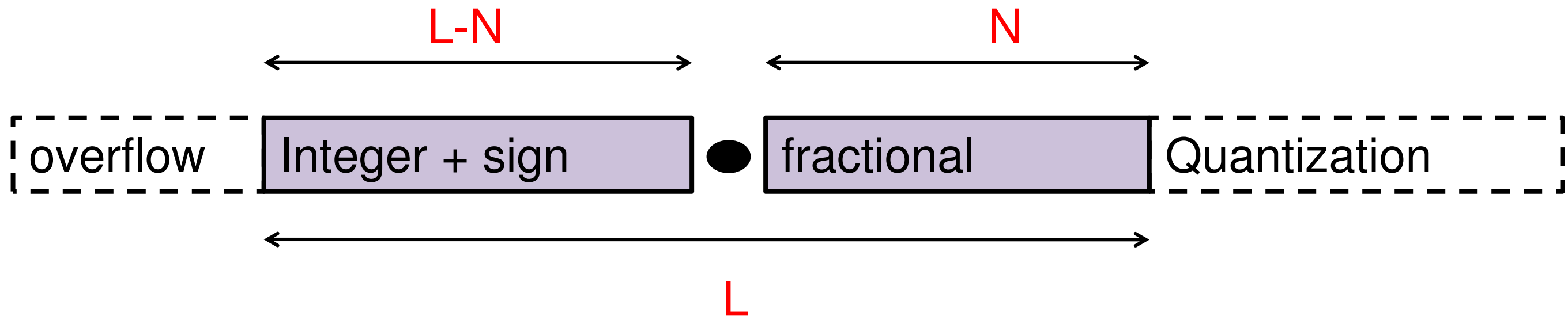
Output	Portable and readable C source code	Executable or software component/library
MATLAB support	Subset of language Some toolboxes	Full language Most toolboxes Graphics
Runtime requirement	None	MATLAB Compiler Runtime (MCR)
License model	Royalty-free	Royalty-free

Fixed Point Design: Motivation

Consideration	Fixed Point	Floating Point
RAM and ROM consumption	Small	Large
Execution time	Faster	Slower
Hardware power consumption	Low	High
Development time	Long	Short
Implementation complexity	More complex. Control of word length, rounding mode, saturation...	Less
Error Prone	Harder to develop. More prone to programming errors	Easier to develop

Fixed Point Design: Pitfalls

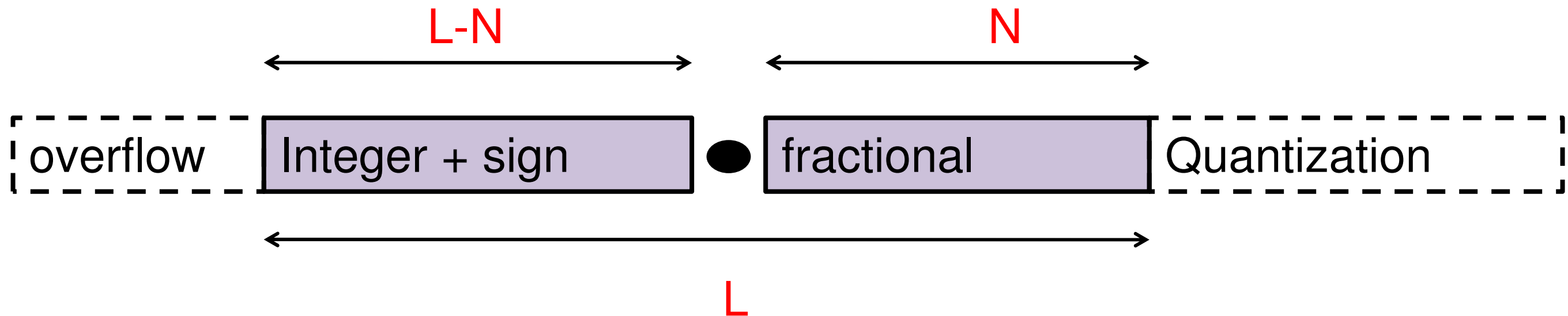
- Arithmetic Pitfalls
 - Introduces quantization errors
 - Word length and Fraction Length must be specified
 - For every variable
 - Degradation must be analyzed



Fixed Point Design: Pitfalls

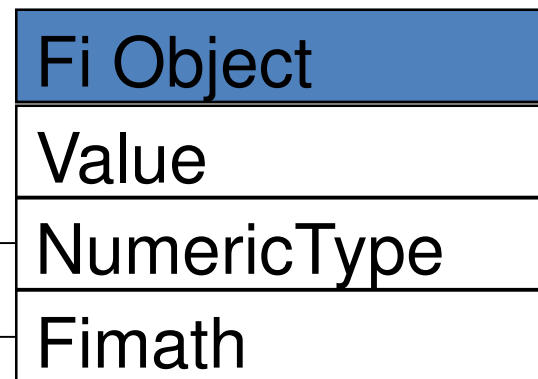
- Fixed Point C Pitfalls

- No native fixed-point math libraries
- No built-in *overflow / underflow* checks
- No tools to determine optimal *integer* and *fractional* bits
- No visualization of floating and fixed-point representations



Fixed-Point Toolbox: MATLAB Fixed-Point Object

Signed: true
WordLength: 16
FractionLength: 13



RoundMode: round
OverflowMode: saturate
ProductMode: FullPrecision
MaxProductWordLength: 128
SumMode: FullPrecision
MaxSumWordLength: 128
CastBeforeSum: true

1. Controls output type of operations
2. Allows natural operator syntax

`A*B, A+B, pow2(A,3)`

Fixed Point Design in MATLAB

The screenshot displays the MATLAB Code Generation Report interface. The main window shows MATLAB code for a function named 'ImageProcess'. A call stack on the left indicates the function is running. A 'NumericTypeScope' window is overlaid on the code, showing a histogram of the variable 'eps'. The histogram is titled 'Unsigned numerictype(false,8,7)' and shows the distribution of absolute data values. The x-axis is labeled 'Absolute Data Values' and ranges from 2^0 to 2^{-8} . The y-axis is labeled 'Occurrences (%)' and ranges from 0 to 60. The histogram shows a peak at 2^0 and a secondary peak at 2^{-1} . The window also displays 'WL=8' and 'FL=7'. A legend on the right indicates 'Outside range' (red), 'In range' (blue), and 'Below precision' (yellow). Below the histogram, a table provides 'Information for the selected variables'.

Size	Class	Complex	Always Whole Number	SimMin	SimMax
1 x 1	double	No	No	0	1.288084984867421

At the bottom of the report, a 'Summary' table lists simulation results for all variables:

Order	Variable	Type	Size	Class	Complex	Always Whole Number	SimMin	SimMax
1	ImageOut	Output	1920 x 2560	double	No	Yes	0	1
2	ImageIn	Input	240 x 320	double	No	No	0	1
3	Factor	Input	1 x 1	double	No	No	1	1.4563650
4	Hin	Local	1 x 1	double	No	Yes	240	240
5	Win	Local	1 x 1	double	No	Yes	320	320
6	Wout	Local	1 x 1	double	No	Yes	240	240

Collect histograms for signals

Run MATLAB code with floating point data types

Simulation results for all variables

Analyze simulation min/max

Benefits of C Code Generation with MATLAB Coder

- Generate C code directly
 - Automatically generated C code is correct by construction
 - Reduce verification effort and cost
- Maintain floating and fixed-point designs in a unified environment
 - Run simulations in double precision or fixed-point as needed
 - Validate fixed-point effects during system design phase