MATLAB to C Made Easy

Generating readable and portable C code from your MATLAB algorithms

Bill Chou

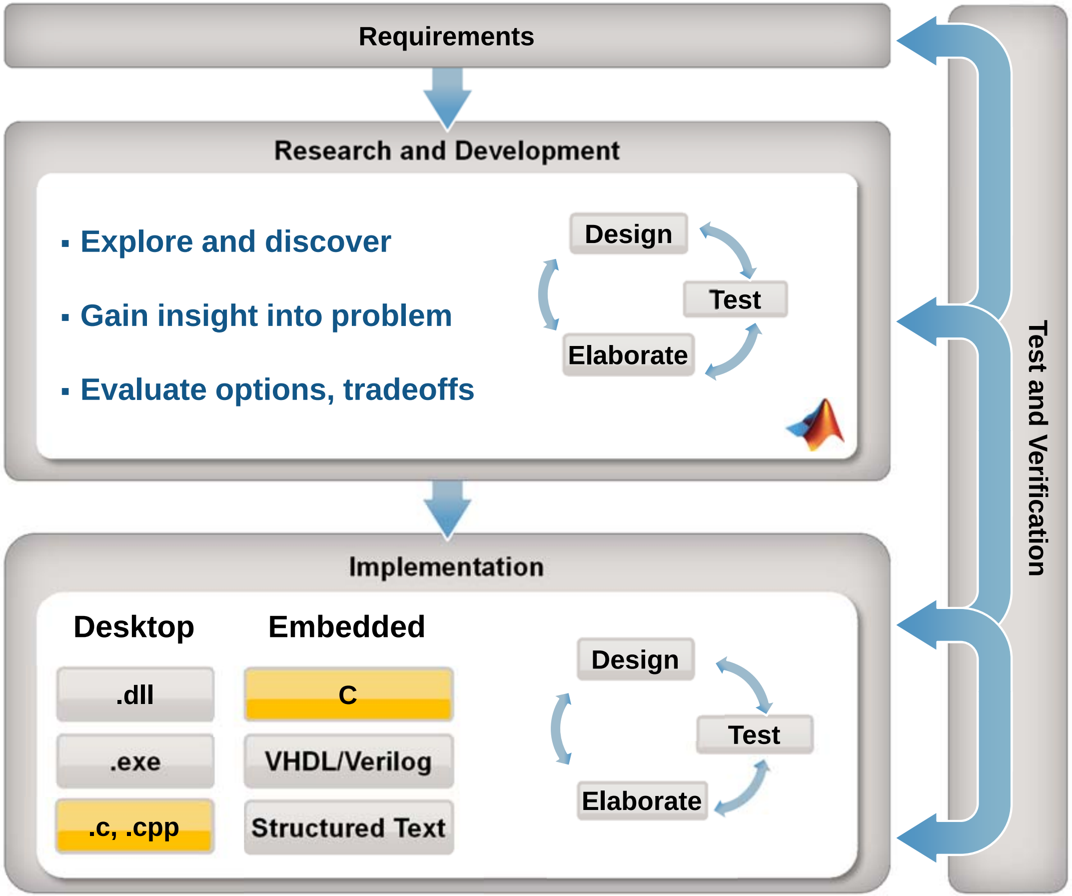
Agenda

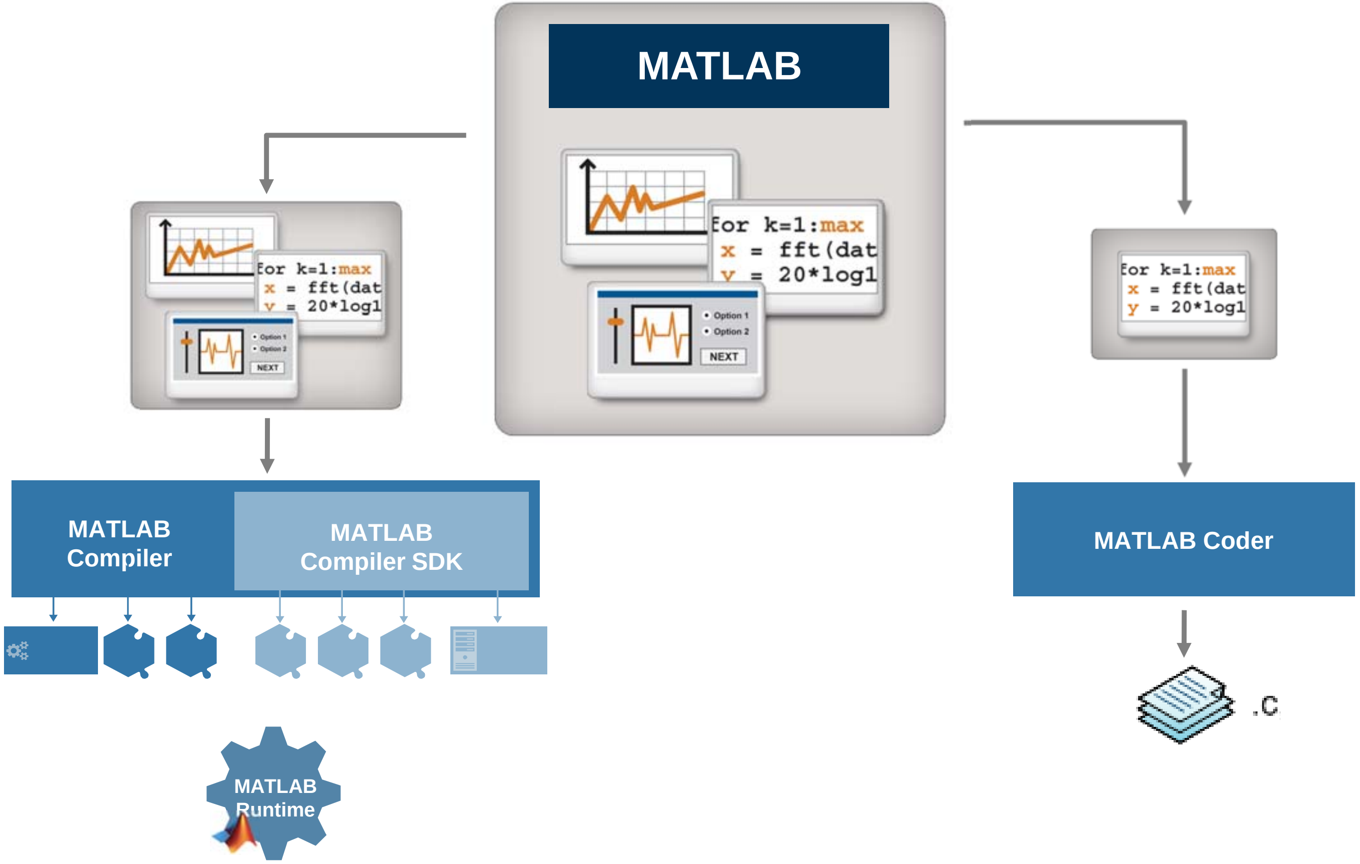
- Motivation
  - Why translate MATLAB to C?
  - Challenges of manual translation
- Using MATLAB Coder
  - Three-step workflow for generating code
- Use cases
  - Integrate algorithms using source code/libraries
  - Accelerate through MEX
  - Prototype by generating EXE
- Conclusion
  - Integration with Simulink and Embedded Coder
  - Other deployment solutions

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

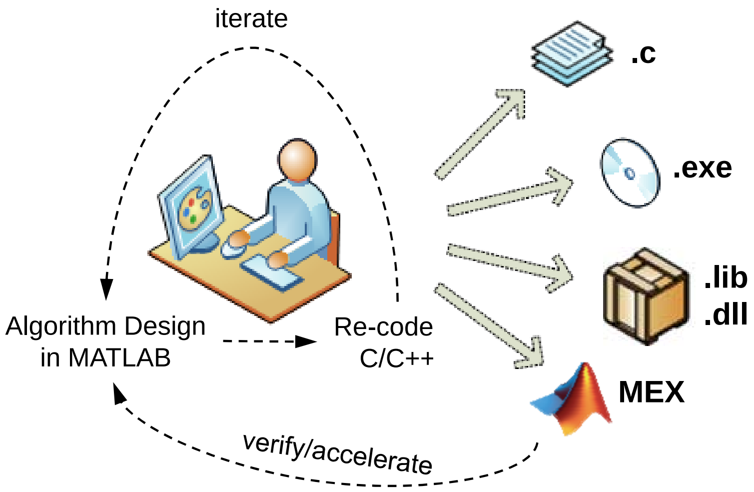
Algorithm Development Process





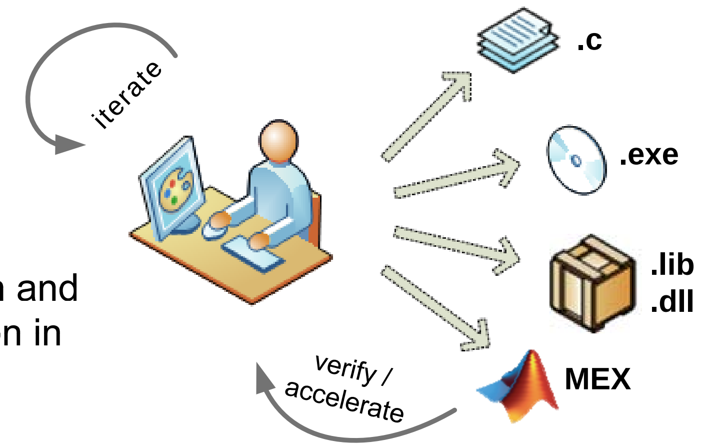
MATLAB
for k=1:max
x = fft(dat
y = 20*log1
Option 1
Option 2
NEXT
MATLAB Compiler
MATLAB Compiler SDK
MATLAB Runtime
MATLAB Coder
.C

Challenges with Manual Translation
from MATLAB to 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 process

Automatic Translation of MATLAB to C



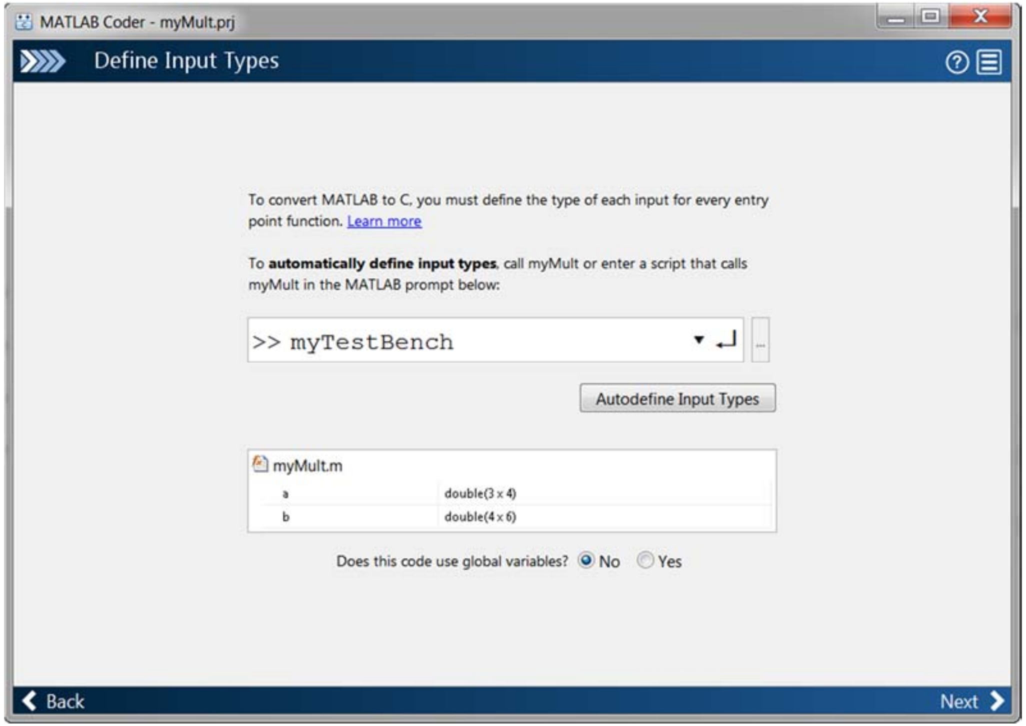
Algorithm Design and Code Generation in MATLAB

With MATLAB Coder, design engineers can:

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

Simple Demo
c = a*b

- MATLAB Coder app
- Autodefine input type
- Check for Run-Time issues
- Code generation report

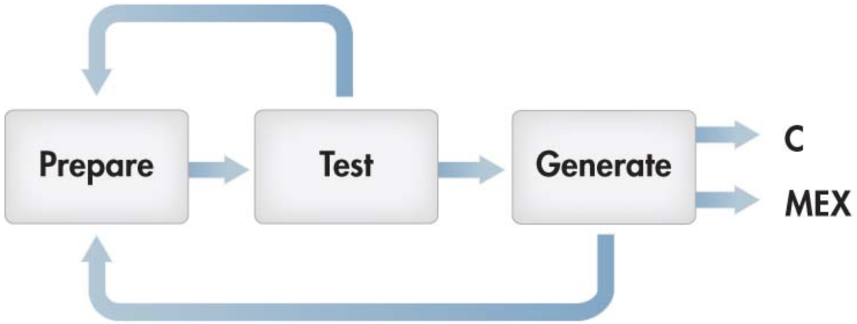


>> Demo

Agenda

- Motivation
  - Why translate MATLAB to C?
  - Challenges of manual translation
- Using MATLAB Coder
  - Three-step workflow for generating code
- Use cases
  - Integrate algorithms using source code/libraries
  - Accelerate through MEX
  - Prototype by generating EXE
- Conclusion
  - Integration with Simulink and Embedded Coder
  - Other deployment solutions

Using MATLAB Coder: Three-Step Workflow

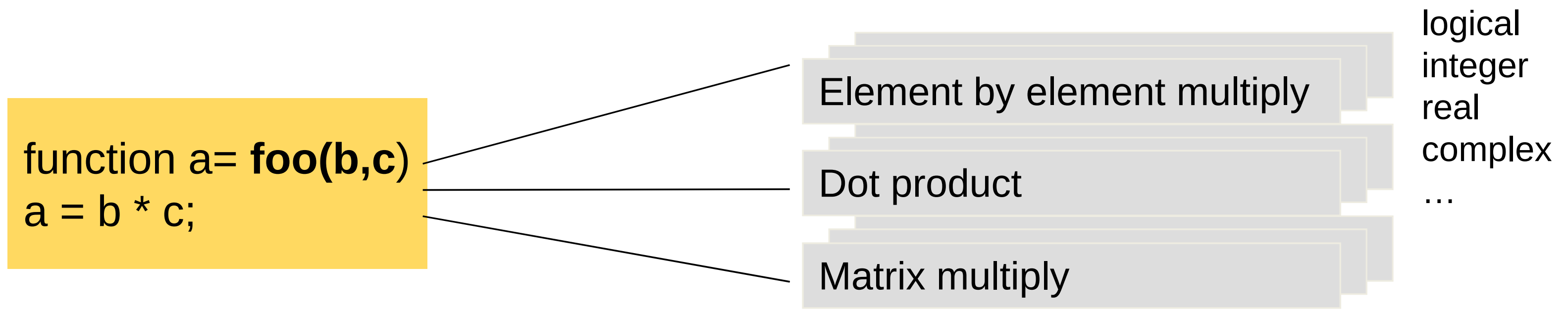


Prepare your MATLAB algorithm for code generation
- Make implementation choices
- Use supported language features

Test if your MATLAB code is ready for code generation
- Validate that MATLAB program generates code
- Accelerate execution of user-written algorithm

Generate source code or MEX for final use
- Iterate your MATLAB code to optimize
- Implement as source, executable, or library

Implementation Considerations



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 Considerations

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

7 Lines of MATLAB
105 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 do
                     const do
                     double p
{
  double klm_gain[12];
  int r1;
  int r2;
  int k;
  double S[4];
  double a21;
  double B[12];
  double a22;
  double Y[12];
  double b_z[2];
  double b_klm_gain[36];
  for (r1 = 0; r1 < 2; r1++) {
    for (r2 = 0; r2 < 6; r2++) {
      klm_gain[r1 + (r2 << 1)] = 0.0;
```

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

  for (r1 = 0; r1 < 2; r1++) {
    for (r2 = 0; r2 <
      a21 = 0.0;
      for (k = 0; k <
        a21 += klm_ga
      }

      S[r1 + (r2 << 1
    }
  }

  for (r1 = 0; r1 < 2
    for (r2 = 0; r2 <
```

```
  if (fabs(S[1]) > fabs(S[0])) {
    r1 = 1;
    r2 = 0;
  } else {
    r1 = 0;
    r2 = 1;
  }

  a21 = S[r2] / S[r1];
  a22 = S[2 + r2] - a21 * S[2 + r1];
  for (k = 0; k < 6; k++) {
    Y[1 + (k << 1)] = (B[r2 + (k << 1)] - B[r1 + (k << 1)] * a21) / a22;
    Y[k << 1] = (B[r1 + (k << 1)] - Y[1 + (k << 1)] * S[2 + r1]) / S[r1];
  }

  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 < 6; r1++) {
    for (r2 = 0; r2 < 6; r2++) {
      b_klm_gain[r1 + 6 * r2] = 0.0;
      for (k = 0; k < 2; k++) {
        b_klm_gain[r1 + 6 * r2] += klm_gain[r1 + 6 * k] * H[k + (r2 << 1)];
      }
    }
  }

  for (r1 = 0; r1 < 6; r1++) {
    for (r2 = 0; r2 < 6; r2++) {
      a21 = 0.0;
      for (k = 0; k < 6; k++) {
        a21 += b_klm_gain[r1 + 6 * k] * p_prd[k + 6 * r2];
      }

      p_est[r1 + 6 * r2] = p_prd[r1 + 6 * r2] - a21;
    }
  }
}
```

Example: Newton/Raphson Algorithm

Preparing your MATLAB code

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

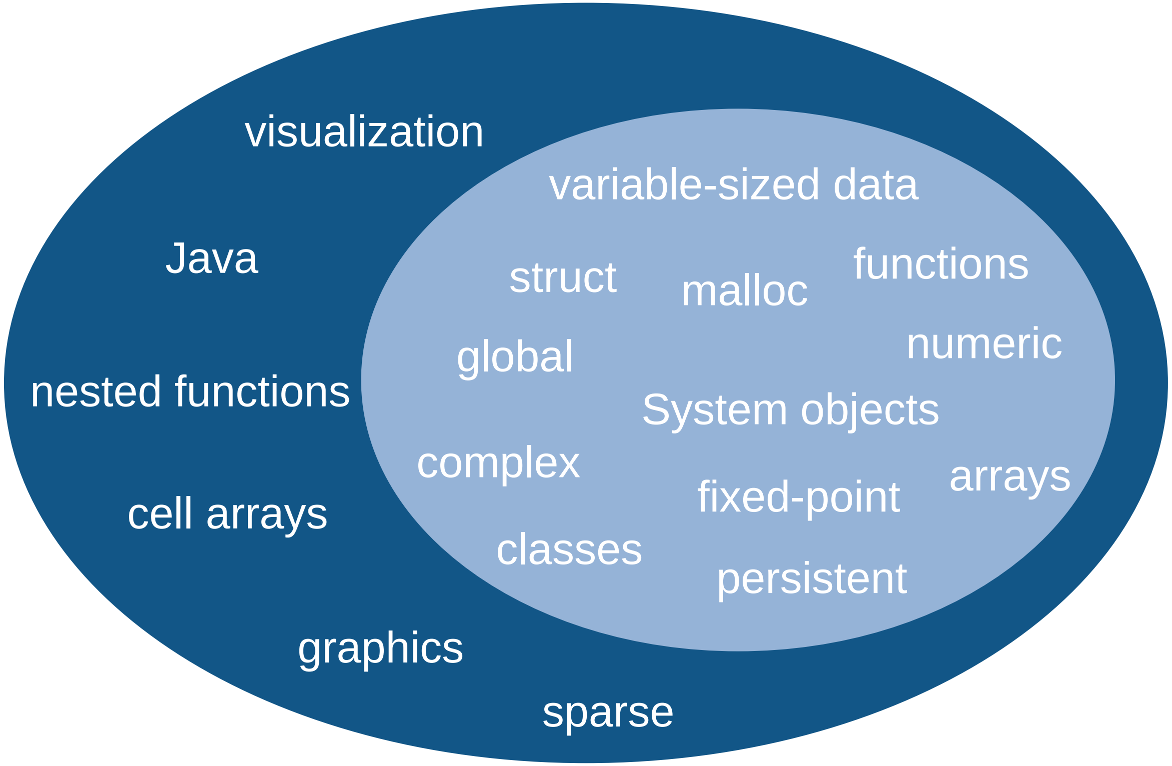
- Pre-allocate
- Identify more efficient constructs
- Select code generation options

```
function [x,h] = newtonSearchAlgorithm(b,n,tol)
% Given, "a", this function finds the nth root of a
% number by finding where: x^n-a=0.

    notDone = 1;
    aNew    = 0; %Refined Guess Initialization
    a       = 1; %Initial Guess
    cnt     = 0;
    h(1)=a;
    while notDone
        cnt = cnt+1;
        [curVal,slope] = f_and_df(a,b,n); %square
        yint = curVal-slope*a;
        aNew = -yint/slope; %The new guess
        h(cnt)=aNew;
        if (abs(aNew-a) < tol) %Break if it's converged
            notDone = 0;
        elseif cnt>49 %after 50 iterations, stop
            notDone = 0;
            aNew = 0;
```

>> Demo

MATLAB Language Support for Code Generation



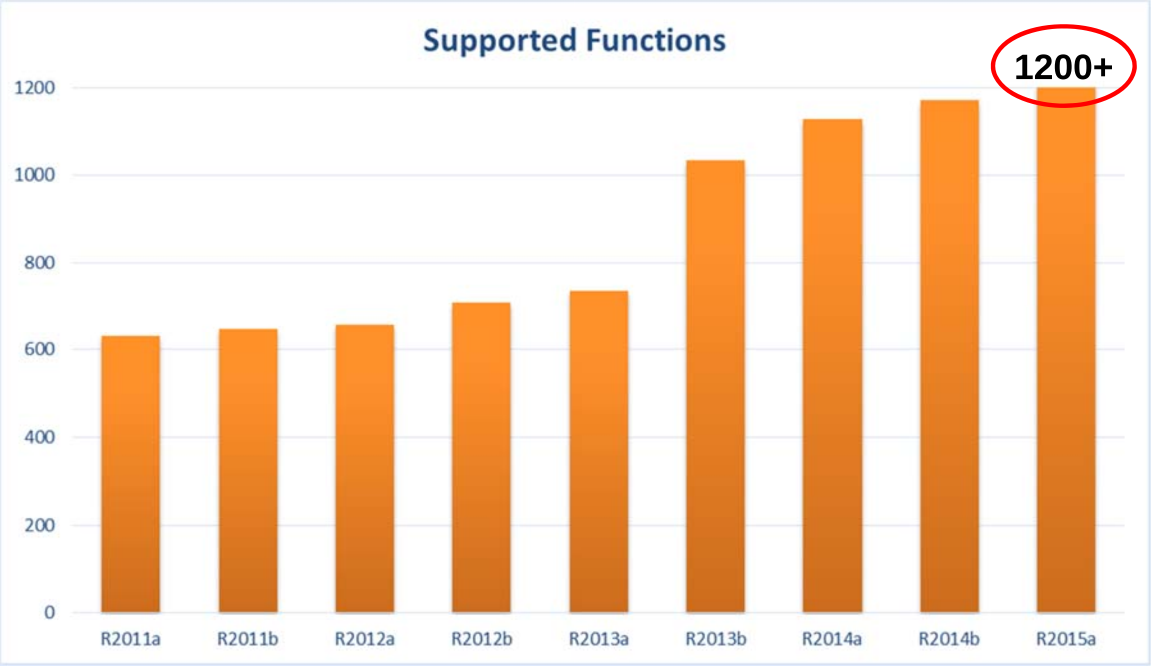
Supported MATLAB Language Features and Functions

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

Matrices and Arrays	Data Types	Programming Constructs	Functions

• Matrix operations
• N-dimensional arrays
• Subscripting
• Frames
• Persistent variables
• Global variables
	• Complex numbers
	• Integer math
	• Double/single-precision
	• Fixed-point arithmetic
	• Characters
	• Structures
	• Numeric class
	• Variable-sized data
	• MATLAB Class
	• System objects
		• Arithmetic, relational, and logical operators
		• Program control (`if, for, while, switch`)
			• MATLAB functions and subfunctions
			• Variable-length argument lists
			• Function handles
			
			Supported algorithms
			• More than 1100 MATLAB operators, functions, and System objects for:
			  • Communications
			  • Computer vision
			  • Image processing
			  • Phased Array signal processing
			  • Signal processing
			  • Statistics

Supported Functions

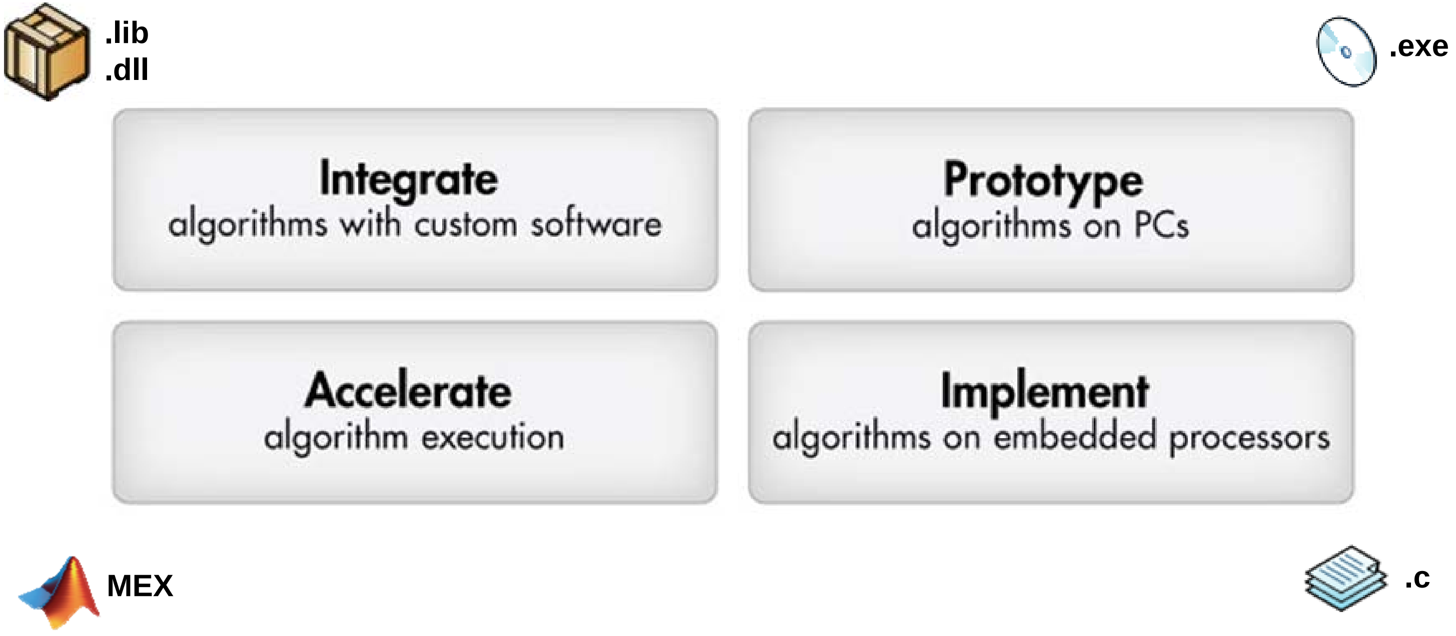


- Aerospace Toolbox
- Communications System Toolbox
- Computer Vision System Toolbox
- DSP System Toolbox
- Image Processing Toolbox
- Neural Networks Toolbox
- Phased Array System Toolbox
- Signal Processing Toolbox
- Statistics and Machine Learning Toolbox

Agenda

- Motivation
  - Why translate MATLAB to C?
  - Challenges of manual translation
- Using MATLAB Coder
  - Three-step workflow for generating code
- Use cases
  - Integrate algorithms using source code/libraries
  - Accelerate through MEX
  - Prototype by generating EXE
- Conclusion
  - Integration with Simulink and Embedded Coder
  - Other deployment solutions

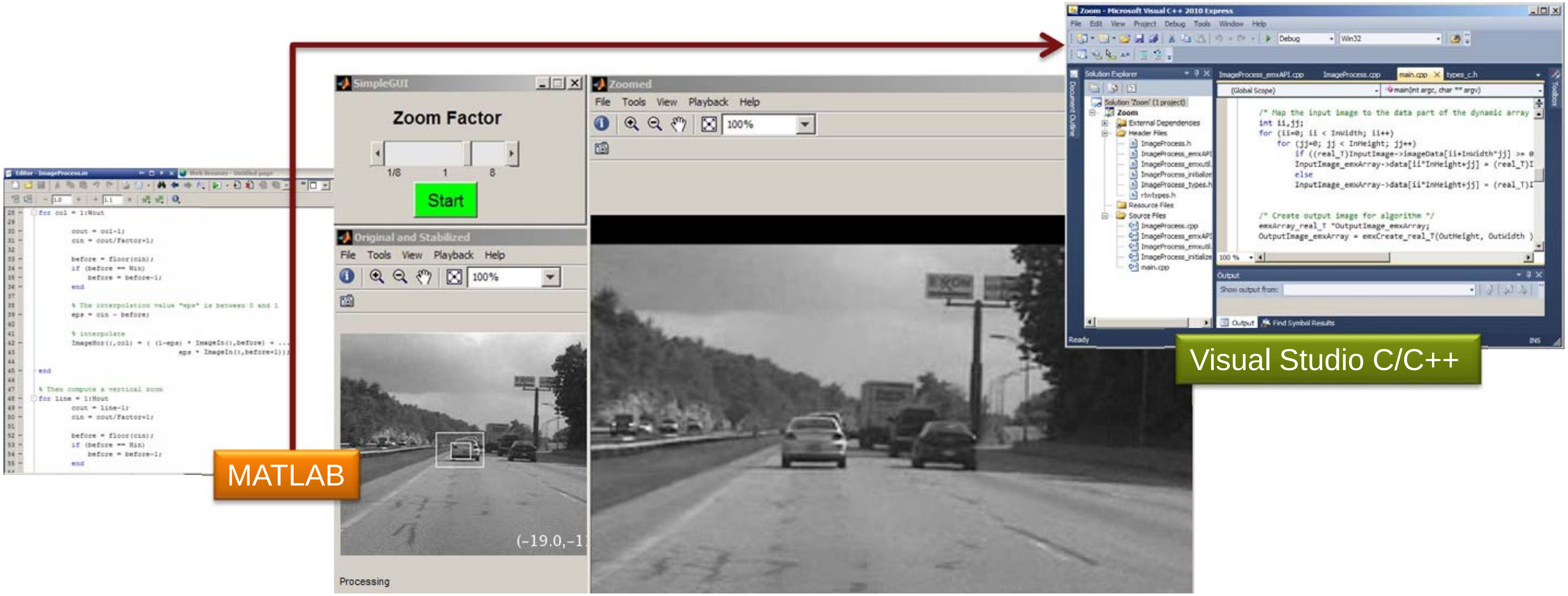
MATLAB Coder Use Cases



Example: Code Integration for Zoom Algorithm

with OpenCV Visual Studio Parent Project

Integrate
algorithms with custom software



>> Demo

Acceleration Strategies

Accelerate
algorithm execution

- Better algorithms

  Matrix inversion vs. QR or SVD

  – Different approaches to solving the same problem

- More efficient implementation

  Hand-coded vs. optimized library (BLAS and LAPACK)

  – Different optimization of the same algorithm

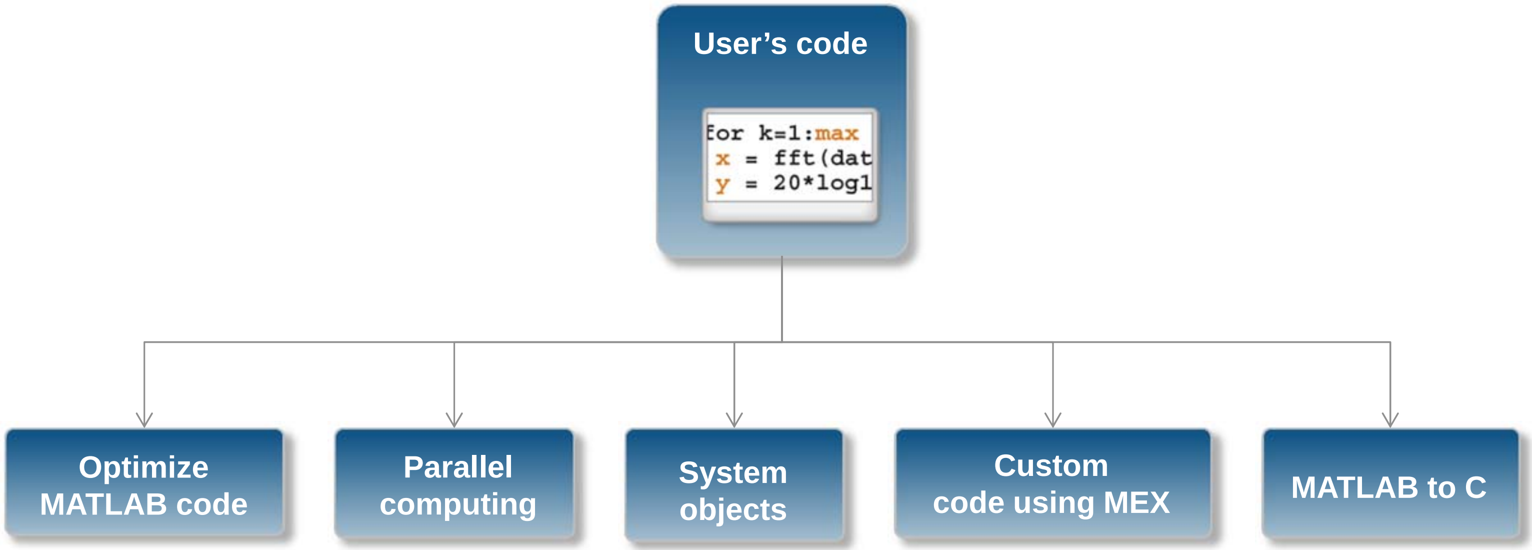
- More computational resources

  Single-threaded vs. multithreaded (multithreaded BLAS)

  – Leveraging additional processors, cores, GPUs, FPGAs, etc.

Accelerating Algorithm Execution

Accelerate
algorithm execution

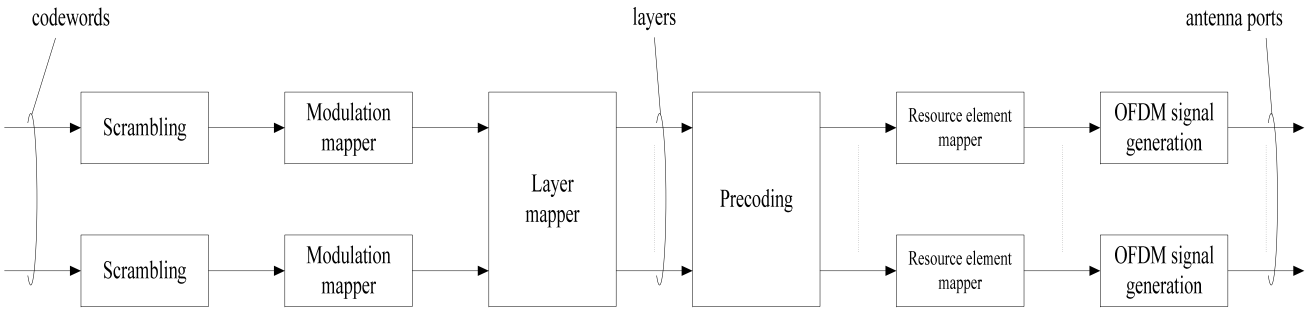


Example: Simulate Downlink physical layer of LTE

Accelerate algorithm execution

Optimize MATLAB code

System objects



Parallel computing

MATLAB to C

>> Demo

Acceleration Using MEX

Accelerate
algorithm execution

- Speed-up factor will vary

- When you **may** see a speedup:
  - Often for communications and signal processing
  - Always for fixed point
  - Likely for loops with states or when vectorization isn't possible

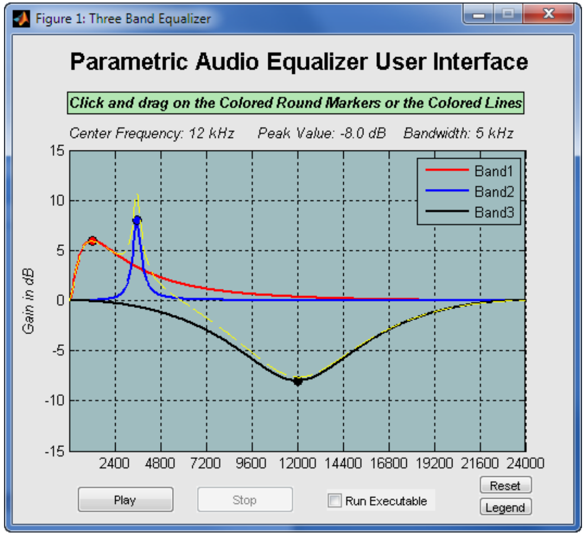
- When you **may not** see a speedup:
  - MATLAB implicitly multithreads computation.
  - Built-functions call IPP or BLAS libraries.

Example: Creating Standalone Executable

Three-Band Parametric Equalizer

Prototype
algorithms on PCs

- Need to provide `main.c` for entry point
- Use System objects from DSP System Toolbox to stream live audio to/from sound card
- Use UDP objects to pass filter coefficients between GUI in MATLAB and generated EXE



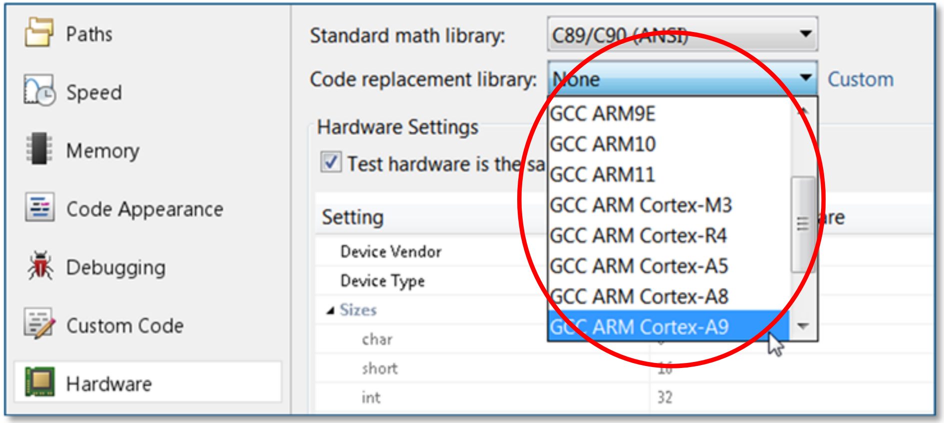
>> Demo

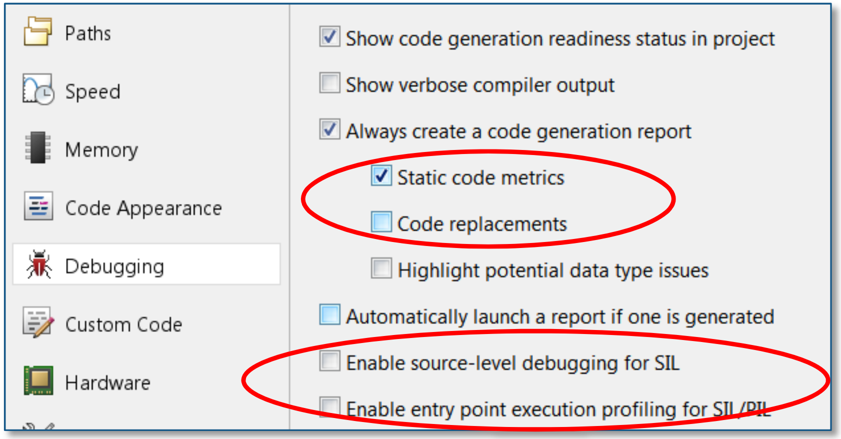
Agenda

- Motivation
  - Why translate MATLAB to C?
  - Challenges of manual translation
- Using MATLAB Coder
  - Three-step workflow for generating code
- Use cases
  - Integrate algorithms using source code/libraries
  - Accelerate through MEX
  - Prototype by generating EXE
- Conclusion
  - Integration with Simulink and Embedded Coder
  - Other deployment solutions

Working with Embedded Coder

- Advanced support for MATLAB Coder, including:
  - Speed & Memory
  - Code appearance
  - Hardware-specific optimization
  - Software/Processor-in-the-loop verification
  - Execution profiling





2. Global Variables [hide]

Global variables defined in the generated code.

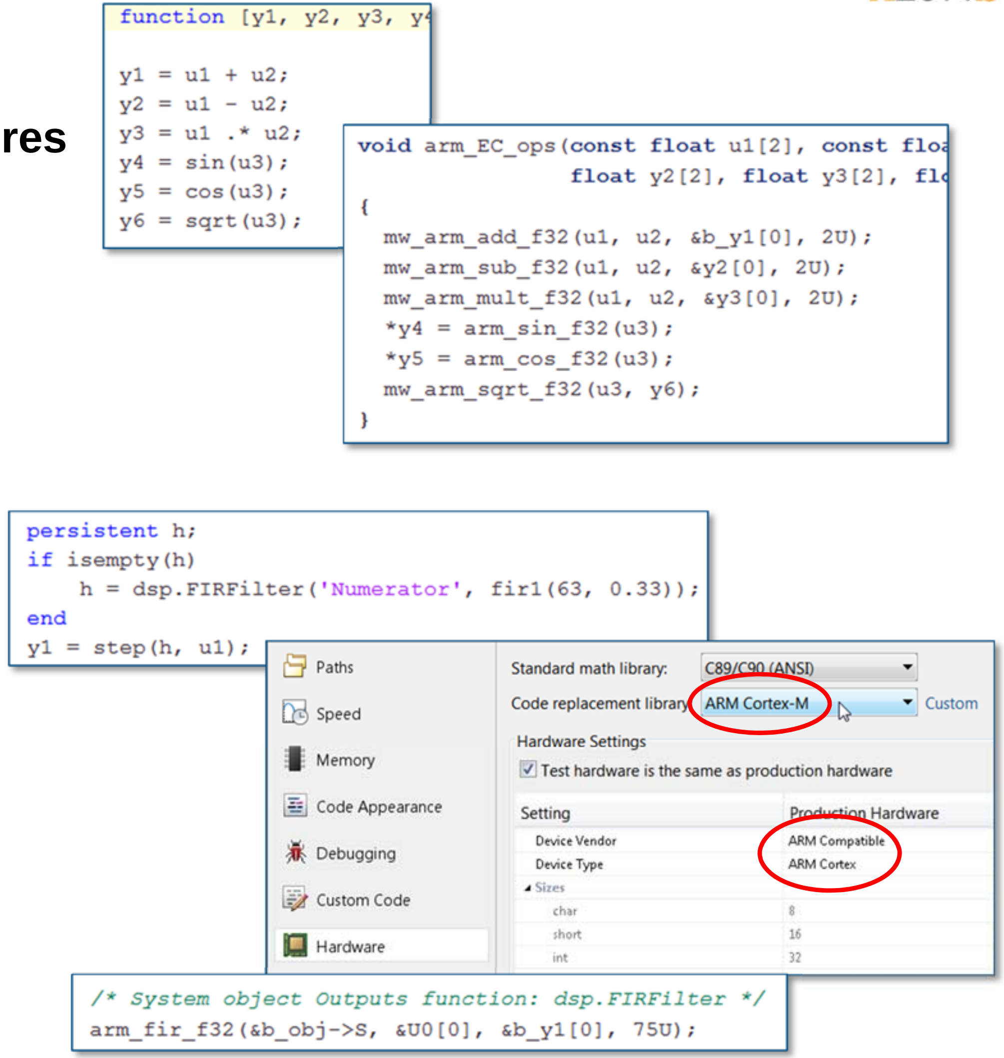
Global Variable	Size (bytes)
myglobal	240
Total	240

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_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()`
- Replace System objects including:
  - `dsp.FIRFilter, dsp.BiquadFilter, dsp.FFT, dsp.IFFT, dsp.Convolver, dsp.CICCompensationInterpolator, dsp.DigitalUpConverter,`

  With CMSIS functions such as:
  - `arm_fir_f32(), arm_lms(), arm_biquad_cascade_df1(), arm_cfft_radix2_f32(), arm_conv()`



ARM Cortex-A Optimized Code

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

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

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

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

```
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);
}
```

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

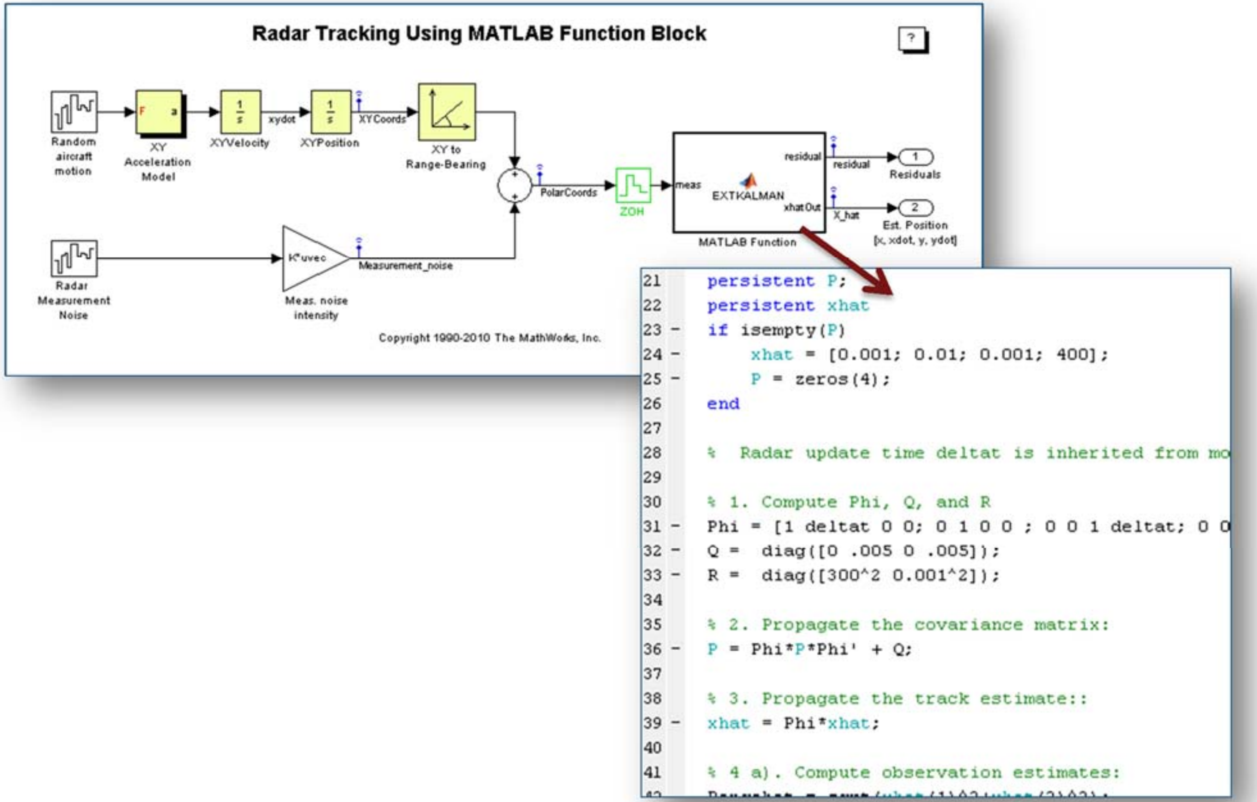
  With Ne10 functions such as:
  - `ne10_fir_init_float()`, `ne10_fft_c2c_1d_float32_neon()`, `ne10_fir_interpolate_float_neon()`, `ne10_fir_decimate_float_neon()`

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

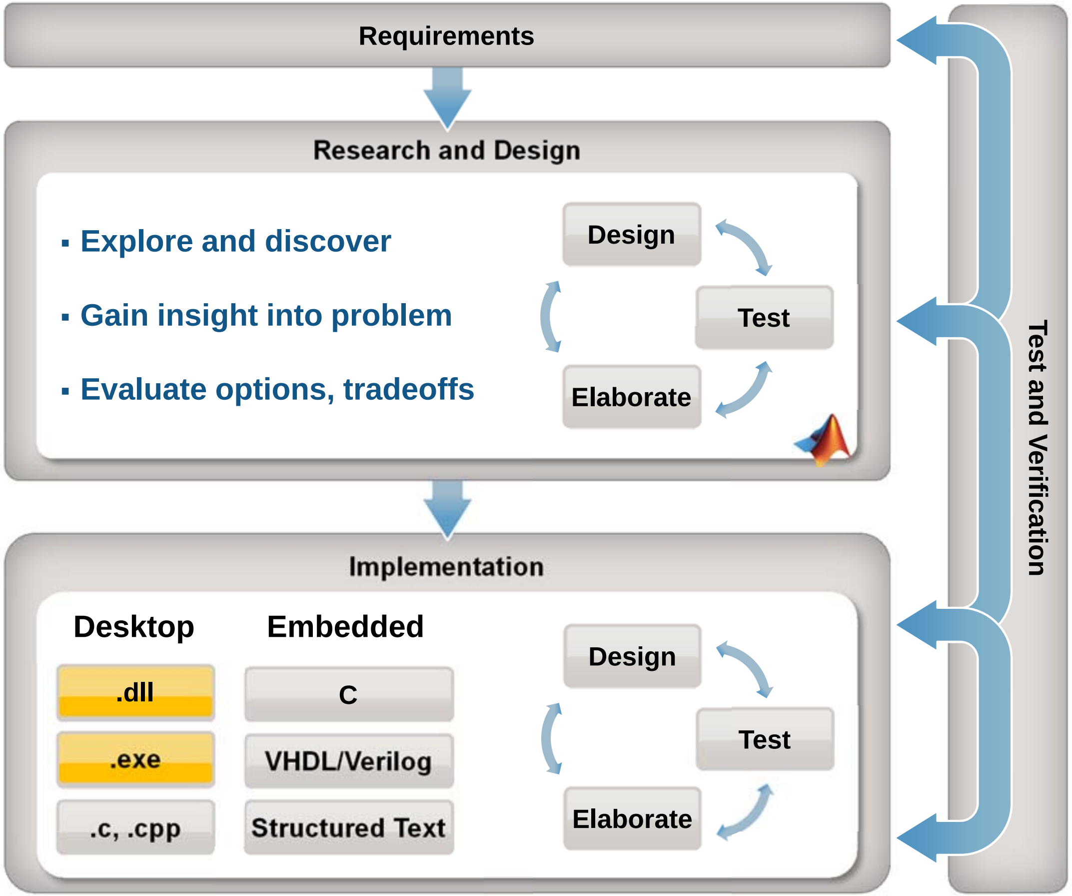
```
/* System object Outputs function: dsp.FIRFilter */
ne10_fir_float_neon(&obj->cSFunObject.S, &U0[0], &b_y1[0], 76U);
```

Working with Simulink and Embedded Coder

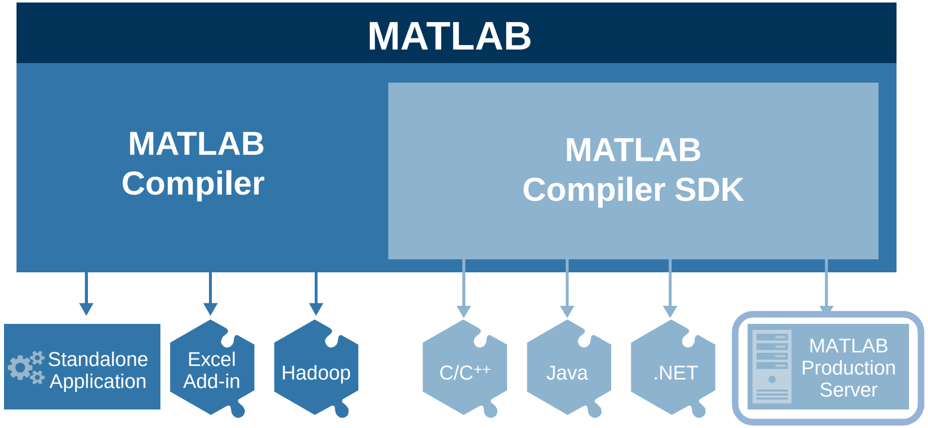
MATLAB Function block in Simulink



Other Desktop Deployment Options



Other Deployment Options

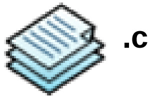


MATLAB Compiler for sharing MATLAB programs without integration programming

MATLAB Compiler SDK provides implementation and platform flexibility for software developers

MATLAB Production Server provides the most efficient development path for secure and scalable web and enterprise applications

Choosing the Right Deployment Solution



	MATLAB Coder	MATLAB Compiler MATLAB Compiler SDK
Output	Portable and readable C source code	Software components
MATLAB support	Subset of language Some toolboxes	Full language Most toolboxes Graphics
Additional libraries	None	MATLAB Runtime
License model	Royalty-free	Royalty-free
Extensions	Embedded Coder	MATLAB Production Server

More Information

- To learn more, visit the product page:
  mathworks.com/products/matlab-coder

- To request a trial license:
  - Talk to your MathWorks account manager to request a trial license and set up a guided evaluation with an application engineer

Q&A