Signal Processing with MATLAB: System Simulation and Real-Time Implementation

Gabriele Bunkheila
Senior Application Engineer
Session Highlights

- Efficient system simulations in MATLAB for DSP, communications, computer vision, and radar systems
- Latest capabilities to generate and customise embeddable C code directly from MATLAB
- Guidance and tools to aid preparation of MATLAB code for implementation
- FPGA/ASIC verification and implementation directly from MATLAB
MATLAB for Signal Processing – three key requirement areas

- Application-relevant algorithms
- Faster simulations
- Path to implementation on real-time targets
Agenda

- Workflow from MATLAB algorithms to embeddable C and simulation acceleration
- System simulation algorithms in the area of Signal Processing
- Workflow from algorithms to HDL
Example – tracking an acoustic source

- Simple example
- MATLAB validation harness for graphics and display, calling out to compiled code
- Algorithm coded in C
  - Auto-generated from MATLAB code
  - Portable to virtually any software platform
  - Comfortably running live on a PC
Application overview

- Sound source
- ALGORITHM
- Delay
- \( \theta \)
Applications

- Sniper detection
- Hands-free conferencing systems
- Speaker identification
- Gaming applications
- Other security applications
From MATLAB algorithms to real-time code

- Experiment with algorithm in MATLAB
- Architect/review/optimize MATLAB code
- Generate real-time source C/C++ code
- Verify/validate generated code
- Optimize generated code
From MATLAB algorithms to real-time code

- Experiment with algorithm in MATLAB
- Architect/review/optimize MATLAB code
- Generate real-time source C/C++ code
- Verify/validate generated code
- Optimize generated code
Step 1 – Explore algorithm in MATLAB

- Simplify:
  - Get a static source
  - Take a picture
  - Record the stereo audio input

- Understand your setup
  - What distance between microphones?
From MATLAB algorithms to real-time code

- Experiment with algorithm in MATLAB
- Architect/review/optimize MATLAB code
- Generate real-time source C/C++ code
- Verify/validate generated code
- Optimize generated code
Package algorithm as a function

- Minimum requirement for
  - Testability
  - Re-usability
  - C code generation
Reviewing our objectives – things to keep in mind

MATLAB Function

Real-time embeddable C

Efficiency

Accelerate execution through C

How you write MATLAB code

Greatly impacts

Performance and quality of generated C
Interpolation in a nutshell

- Includes
  - Up-sampling input signal (Inserting zeros between samples)
  - Filtering with a low-pass filter
How many ways to do this in MATLAB?

- How often to compute filter coefficients?
- How optimized a filtering algorithm
- … ?

<table>
<thead>
<tr>
<th>Insert zeros and filter conventionally</th>
<th>All times</th>
<th>Only once</th>
</tr>
</thead>
<tbody>
<tr>
<td>interp</td>
<td></td>
<td>persistent</td>
</tr>
<tr>
<td>upsample</td>
<td></td>
<td></td>
</tr>
<tr>
<td>filter(b, a, x)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Use efficient filtering structures (e.g. polyphase)</th>
<th>All times</th>
<th>Only once</th>
</tr>
</thead>
<tbody>
<tr>
<td>filter(mfilt)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>step(dsp.FIRInterpolator)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Investigate efficiency of MATLAB code

- tic/toc

```matlab
% start timer
tic

% execute code
out = myFunction(in);

% stop timer (and store % elapsed time)
et = toc;
```

- profile

```matlab
% turn on profiler
profile on

% execute code
out = myFunction(in);

% turn off profiler
profile off
% open html report
profile report
```

- How long did it take?
- Where are the bottlenecks?
Profile and review

- Before

```matlab
14    % Cross-correlate pair channels to get delay
2.46 7650   19    xc = xcorr(in(:,1), in(:,2));
16
17    % Interpolate to increase spatial resolution
7.45 7650   18    xclnt(:, 1) = interpSimple(xc, InterpFactor);
19    % xclnt(:, 1) = interp(xc, InterpFactor);
20
0.05 7650   21    [~, Index] = max(xclnt);
0.02 7650   22    delayVector(n) = (Index + InterpFactor - 1)/InterpFact
23    end
24
< 0.01 153   25    delay = median(delayVector);
```

- After

```matlab
14    % Cross-correlate pair channels to get delay
2.42 7650   15    xc = xcorr(in(:,1), in(:,2));
16
17    % Interpolate to increase spatial resolution
1.54 7650   16    xclnt = interpSeparated(xc, InterpFactor);
18
0.04 7650   20    [~, Index] = max(xclnt);
0.02 7650   21    delayVector(n) = (Index + InterpFactor - 1)/InterpFact
0.01 7650   22    end
0.01 153   24    delay = median(delayVector);
```

time  calls  line
1 function  odata = interpSeparated(idata,l)
0.03 7650   3   l = 4;
0.06 7650   4   cutoff = .5;
0.08 7650   6   persistent b
0.02 7650   8   if(isempty(b))
0.06 1   9     b = computeFilter(l,r,cutoff);
1.27 7650  12   odata = useFilter(b, idata, l, r);

- Interpolation: 5x faster
- Algorithm: > 2x faster
Lesson learnt

- Separate initialization and setup from recurring execution
From MATLAB algorithms to real-time code

- Experiment with algorithm in MATLAB
- Architect/review/optimize MATLAB code
- Generate real-time source C/C++ code
- Verify/validate generated code
- Optimize generated code
Translating MATLAB to C – Typical use cases

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.c, .cpp</td>
<td>Hand-off code to software engineers for implementation (e.g. on embedded processor)</td>
</tr>
<tr>
<td>.lib</td>
<td>Integrate MATLAB algorithms w/ existing C environment</td>
</tr>
<tr>
<td>.exe</td>
<td>Deploy MATLAB algorithms on Windows/Linux desktop PC</td>
</tr>
<tr>
<td>MEX</td>
<td>Accelerate MATLAB algorithms</td>
</tr>
</tbody>
</table>
Same algorithm, different performance

- tic/toc timing for different existing versions of the same algorithm

<table>
<thead>
<tr>
<th>Versions of the Algorithm</th>
<th>Elapsed Time (sec)</th>
<th>Acceleration Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Traditional use of MATLAB functions</td>
<td>7.3740</td>
<td>1.0000</td>
</tr>
<tr>
<td>2. Separating setup and execution</td>
<td>3.8605</td>
<td>1.9101</td>
</tr>
<tr>
<td>3. MATLAB Coder MEX version</td>
<td>0.8062</td>
<td>9.1462</td>
</tr>
</tbody>
</table>

- What did the trick?
  - Better-structured algorithms
  - Automatic C code generation
Functions supported for code generation

System objects supported for code generation
From MATLAB algorithms to real-time code

- Experiment with algorithm in MATLAB
- Architect/review/optimize MATLAB code
- Generate real-time source C/C++ code
- Optimize generated code
- Verify/validate generated code
Calling out to existing C code from MATLAB

Use cases:

- Optimized C/C++ code
- Legacy code
- Non-algorithmic code (e.g. peripherals, profiling)
- Interacting with larger C/C++ application
- Verifying C/C++ using existing MATLAB testbench

```
coder.ceval('cfun_name', arg1, arg2, ...)
```
Tools for target-specific C code optimization

- Operator level (e.g. fixed-point arithmetic)
  
  \[ c = a + b \]

  \[ *c = \_sadd(a, b) \]

- Routine level

  \[ \text{out} = \text{matlabFcn}(\text{in}) \]

  \[ \text{optimFcn}(\*\text{in}, \*\text{out}) \]

From MATLAB algorithms to real-time code

- Experiment with algorithm in MATLAB
- Architect/review/optimize MATLAB code
- Generate real/time source C/C++ code
- Verify/validate generated code
- Optimize generated code
MATLAB Coder resources

- MATLAB to C made easy (Recorded online webinar)

- MATLAB Coder quick-start guides
  - Preparing MATLAB code
  - Generating source C/C++
  - Simulation acceleration

- MATLAB language support for C code generation
Agenda

- Workflow from MATLAB algorithms to embeddable C and simulation acceleration
- System simulation algorithms in the area of Signal Processing
- Workflow from algorithms to HDL
System Toolboxes for Signal Processing

System Design & Implementation for MATLAB and Simulink

**DSP System Toolbox**

Design and simulate signal processing systems

**Communications System Toolbox**

Design and simulate the physical layer of communication systems

**Computer Vision System Toolbox**

Design and simulate computer vision and video processing systems

**Phased Array System Toolbox**

Design and simulate phased array signal processing systems
Lesson learnt

- Separate **initialization and setup** from **recurring execution**
Two entry points for MATLAB programming

- Data analysis
- Algorithm investigation

- System simulation
- Efficient C implementation

% All happens in one place
outImage = ...
    imfilter(inImage, [1 -1; 0 1])

% instatiate first
filt2D = vision.ImageFilter(...
    'Coefficients', [1 -1; 0 1]);
...
% iteratively process streamed data
while(...)
...
    % process
    outImage = filt2D.step(inImage);
...
end
Using System objects on our example

• Create objects and configure parameters
• Call **step** on each object to perform its computation

```matlab
% Create and initialize objects
Interpolator = dsp.FIRInterpolator('Numerator',b);
Correlator = dsp.Crosscorrelator;

% Process audio data in a loop
for k = 1:nFrames
    inInt = step(Interpolator, in);
    xc = step(Correlator, inInt(:,1), inInt(:,2));
    [~,maxIdx] = max(xc);
end
```
Phased Array System Toolbox

Tools and algorithms to model and simulate every aspect of a phased array signal processing system:

- Array design and analysis
- Waveform design and analysis
- Transmitter and receiver modeling
- Target and environment modeling
- Temporal processing
- Spatial processing
- Space-time adaptive processing
Phased Array Systems – Example of a System-Level Block Diagram

Waveform Generator → Transmitter → Transmit Array → Environment, Targets, and Interference → Receive Array → Receiver → Signal Processing
Example – modelling a beamscan radar

- Monostatic radar
- 30 x 30 rectangular array
- Beamscan for angle resolution
- Detect two moving targets
System Toolboxes

System Design & Implementation for MATLAB and Simulink

- Libraries of algorithms for system implementation

- Support both MATLAB and Simulink
  - Functions, System objects, Blocks

- Support Coder products
  - Execution acceleration
  - Embedded deployment

- Optionally include analysis tools for design-space exploration, verification
## Desktop Acceleration: Growing number of GPU implementations

- Addressing application-specific bottlenecks

<table>
<thead>
<tr>
<th>comm.gpu.AWGNChannel</th>
<th>Add white Gaussian noise to input signal with GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>comm.gpu.BlockDeinterleaver</td>
<td>Restore original ordering of block interleaved sequence with GPU</td>
</tr>
<tr>
<td>comm.gpu.BlockInterleaver</td>
<td>Create block interleaved sequence with GPU</td>
</tr>
<tr>
<td>comm.gpu.ConvolutionalDeinterleaver</td>
<td>Restore ordering of symbols using shift registers with GPU</td>
</tr>
<tr>
<td>comm.gpu.ConvolutionalEncoder</td>
<td>Convolutionally encode binary data</td>
</tr>
<tr>
<td>comm.gpu.ConvolutionalInterleaver</td>
<td>Permute input symbols using shift registers with GPU</td>
</tr>
<tr>
<td>comm.gpu.LDPCDecoder</td>
<td>Decode binary low-density parity-check data with GPU</td>
</tr>
<tr>
<td>comm.gpu.PSKDemodulator</td>
<td>Demodulate using M-ary PSK method with GPU</td>
</tr>
<tr>
<td>comm.gpu.PSKModulator</td>
<td>Modulate using M-ary PSK method with GPU</td>
</tr>
<tr>
<td>comm.gpu.TurboDecoder</td>
<td>Decode input signal using parallel concatenation decoding with GPU</td>
</tr>
<tr>
<td>comm.gpu.ViterbiDecoder</td>
<td>Decode convolutionally encoded data using Viterbi algorithm with GPU</td>
</tr>
</tbody>
</table>

**phased.gpu.ConstantGammaClutter**

- Constant gamma clutter simulation

**GPU acceleration for xcorr, xcorr2, fftfilt, xcov, and cconv functions**

Agenda

- Workflow from MATLAB algorithms to embeddable C and simulation acceleration
- System simulation algorithms in the area of Signal Processing
- Workflow from algorithms to HDL
From models to FPGAs and ASICs
Further challenges and desirable capabilities

- Fixed-point design, resource optimization
- Auto HDL code generation
- Auto elaboration of high-level design choices
- Re-using models for HDL verification
HDL Code Generation from MATLAB and Simulink

- Generate VHDL and Verilog code for FPGA and ASIC design
- One HDL code generation product for both MATLAB and Simulink
- Generates portable HDL code for ASIC, FPGA, and RP/HIL
HDL Coder (R2012a and R2012b)

MATLAB to HDL

- Floating-point to fixed-point conversion and HDL code generation
- HDL optimizations for MATLAB designs
- Integration with simulation and synthesis tools
- Inference, traceability, and resource reports

Simulink to HDL

- Over 20 new optimization, modeling, and HDL code generation enhancements
- Floating-point library support for Xilinx and Altera
- Xilinx System Generator Subsystem
(Generated) Fixed-Point MATLAB Code

```matlab
function [p, q] = mlhdlc_df2t_filter(x, y, z)

% Filter coefficients as constants
b = [0.2925077148375 0.585784912109375 0.2925077148375];
a = [1.0 0.0 0.171600341796875];

for j=1:length(x)
    y(j) = b(1)*x(j) + z(1);
    z(1) = b(2)*x(j) + z(2) - a(2)*y(j);
    z(2) = b(3)*x(j) - a(3)*y(j);
end
p = y;
q = z;
end
```

```matlab
function [p, q] = mlhdlc_df2t_filter_FixPt(x, y, z)

% Filter coefficients as constants
b = fi([0.2925077148375 0.585784912109375 0.2925077148375], 0, 14, 14, fm);
a = fi([1.0 0.0 0.171600341796875], 0, 14, 13, fm);

for j = 1:fixpoint_length(x)
    y(j) = fi(b(1)*x(j) + z(1), 1, 14, 12, fm);
    z(1) = fi(b(2)*x(j) + z(2)) - a(2)*y(j);
    z(2) = fi(b(3)*x(j) - a(3)*y(j), 1, 14, 13, fm);
end
p = fi(y, 1, 14, 12, fm);
q = fi(z, 1, 14, 21, fm);
end
```
Summary

- Flow from MATLAB to C/C++
  - Acceleration
  - Embeddable source code

- Signal Processing – new System Toolboxes
  - Application-specific
  - Efficient simulation
  - Implementation-ready

- New flow from MATLAB to HDL
  - Auto code generation
  - Verification