Algorithm to Implementation

김용정 부 장  Senior Applications Engineer
<table>
<thead>
<tr>
<th>Agenda</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Overview</strong></td>
</tr>
<tr>
<td><strong>Signal Processing Systems Design with MATLAB</strong></td>
</tr>
<tr>
<td>• Modeling and Simulation</td>
</tr>
<tr>
<td>• DSP/Communication/Phased Array System Toolbox</td>
</tr>
<tr>
<td><strong>Conversion to Fixed Point Conversion</strong></td>
</tr>
<tr>
<td>• Fixed Point Design</td>
</tr>
<tr>
<td><strong>Automatic Code Generation</strong></td>
</tr>
<tr>
<td>• MATLAB/Simulink to C/HDL</td>
</tr>
<tr>
<td><strong>Summary and Wrap-up</strong></td>
</tr>
</tbody>
</table>
Challenges in Signal Processing System Design

- Algorithm development and visualization difficult with generic programming languages
- Need access to ready-to-use libraries of signal processing and communications algorithms and design tools
- Need more options for faster simulations of complex systems
- Need efficient and easier access to real-world data
- Need easier options for test-bench creation and verification
Signal Processing System Design with MATLAB
From Algorithm Exploration to System Design

Algorithm

- Filtering…
- Spectral analysis…
- Resampling…

Behavioral trade-offs

- Do I get the right numerical results?
- Does it meet design specifications?
- Is this the right algorithm compared to other choices?

System Model

- Encoding Algorithm
- Filtering Algorithm
- Multirate Algorithm

Implementation trade-offs

- Do the algorithms maintain behavior in the integrated system?
- Does the system meet performance requirements under realistic conditions?
- How can I minimize cost on hardware?
System Toolboxes

Filter Design Toolbox + Signal Processing Blockset → DSP System Toolbox

Communications Toolbox + Communications Blockset → Communications System Toolbox

Video and Image Processing Blockset + New Features → Computer Vision System Toolbox

New Product → Phased Array System Toolbox
System Toolboxes

- Rich set of algorithms for design in MATLAB or Simulink
  - Stream and batch signal processing
- Fixed-point and floating point support
- C code generation from MATLAB or Simulink
- Growing HDL code generation support
- Object-based design flow for design reuse
# Basic Tools for Signal Processing Design

## For Analysis and Algorithms: Signal Processing Toolbox

- Time and frequency domain signal analysis and visualization
- Waveforms, linear prediction, time-series, and spectral estimation
- Industry-standard analog and digital FIR and IIR filter design

## For Systems: DSP System Toolbox

- Streaming and multi-rate algorithms for real-time DSP systems
- Fixed-point, C/HDL code generation, and hardware connectivity
- Multi-rate and adaptive filter design; signal and spectrum scopes
Modeling real-time signal processing systems

- Streaming algorithms in DSP System Toolbox provide
  - Implicit data buffering, state management and indexing
  - Simulation speed-up by reducing overhead
DSP System Toolbox

Over 300 algorithms for
- Advanced filter design
- FFTs
- Multirate DSP
- Linear algebra routines

Algorithm libraries in MATLAB

Algorithm libraries in Simulink
Model Dynamic Systems in MATLAB: System Toolboxes and System objects

Represent dynamic systems

- Multiple tasks: validation, initialize, output, reset, terminate

Separate Declaration from Execution

- Declaration: State initializations, Parameter validation
- Execution: Output computation (step), reset and eventual termination

```matlab
%% Initialization
hFilter = dsp.DigitalFilter('Numerator',fir1(256,0.5));

%% In-loop processing
for n=1:1e6
    % Output computation
    y=step(hFilter,x);

    % Reset states (optionally)
    if mod(n,100)==0, reset(hFilter); end
end

%% Termination
release(hFilter);
```
Communications System Toolbox

Over 100 algorithms for
- Modulation, Interleaving, Channels, Source Coding
- Error Coding and Correction
- MIMO, Equalizers, Synchronization
- Sources and Sinks, SDR hardware

Algorithm libraries in MATLAB

Algorithm libraries in Simulink
LTE Downlink processing

(a) Transport channel processing for DL-SCH
(b) Overview of downlink physical channel processing
Demo: 4G Building blocks

- QPSK Modulation
- Turbo Code (Soft Decision)
- OFDM
- MIMO (Space Time Block Codes)

>> Demo selector: Communications -> 4G Comms system
LTE Downlink – Putting It All Together

Downlink Shared Channel Processing

Transmitter (eNodeB)

Mode 4:
16QAM, full band-width,
Multi-Codeword Spatial Multiplexing Tx.
Direct Matrix Inversion MIMO Receiver,
CRC-enabled Turbo Decoder.

Receiver (UE)

Selected References:
3GPP TS 36.211 v10.0.0 (2010-12),
3GPP TS 36.221 v10.0.0 (2010-12),
3GPP TS 36.213 v10.0.0 (2010-12)

Copyright 2011 The MathWorks, Inc.
Simulink Library Support for HDL

Core Simulink Blocks
- Basic and Array Arithmetic, Look-Up Tables, Signal Routing (Muxes, Delays, Selectors), Logic & Bit Operations, FIFOs, CORDIC, Single- and Dual-port RAMs

Signal Processing Blocks
- NCOs, DDCs, FFTs, Counters, Digital Filters (FIR, IIR, Multi-rate, Adaptive), Rate Changes (Up & Down Sample)

Communications Blocks
- Pseudo-random Sequence Generators, Modulators / Demodulators, Interleavers / Deinterleavers, Viterbi Decoder

Stateflow
- Mealy and Moore Finite State Machines
FPGA-in-the-Loop (FIL) Prototyping

- Part of HDL Verifier
- Easy to setup using FIL Wizard
- Fast simulation
  - HDL runs on FPGA
  - Gigabit Ethernet data transfer

Supported Xilinx boards

- ML605
- ML505
- ML506
- ML507
- XUP Atlys
- ML401
- ML402
- ML403
- XUP-V5
Phased Array System Design with MATLAB
Applications of Phased Array Systems

Source array → Medium → Target → Receiver array and processing → Result

Communications
Imaging
Measurement and Tracking
Phased Array Analysis and Visualization

- Array gain
- Array response
- Delay between elements
- Steering vector
- Element response
Demo: Uniform Linear Array (ULA)

½ wavelength spacing
Conformal Arrays

- Specify arbitrary arrays
  - Position for each element \((x,y,z)\)
  - Normal vector to specify the orientation for each element (azimuth, elevation)

>> Demo selector: PhAST -> Array Patterns
Phased Array Systems: Block Diagram View
# Phased Array System Toolbox

<table>
<thead>
<tr>
<th>Phased array design and analysis</th>
<th>Temporal processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Linear, rectangular, conformal geometries</td>
<td>• Time varying gain, pulse compression</td>
</tr>
<tr>
<td>• Shading, tapering</td>
<td>• Coherent, non-coherent integration</td>
</tr>
<tr>
<td>• Element position and orientation</td>
<td>• Signal detection and ROC curves</td>
</tr>
<tr>
<td>• Gain, delay, steering vector</td>
<td>• CFAR processing, range/Doppler estimation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Waveform design and analysis</th>
<th>Spatial processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Pulsed CW</td>
<td>• Digital beamforming: narrowband &amp; broadband, Conventional, MVDR, LCMV, Frost, time delay, time delay LCMV, subband phase shift</td>
</tr>
<tr>
<td>• LFM and stepped FM</td>
<td>• DOA processing: Monopulse, MVDR, beamscan, ESPRIT, Root-MUSIC</td>
</tr>
<tr>
<td>• Staggered PRF</td>
<td>• Ambiguity function</td>
</tr>
<tr>
<td>• Ambiguity function</td>
<td>• Matched filter</td>
</tr>
<tr>
<td>• Matched filter</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Signal modeling framework</th>
<th>Space-time adaptive processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Monostatic and multistatic scenarios</td>
<td>• Displaced phase center array (DPCA)</td>
</tr>
<tr>
<td>• Point target and Swerling target models</td>
<td>• Adaptive DPCA</td>
</tr>
<tr>
<td>• Narrowband and broadband modeling</td>
<td>• Sample matrix inversion (SMI)</td>
</tr>
<tr>
<td>• Platform motion</td>
<td>• Angle-Doppler response</td>
</tr>
</tbody>
</table>
Summary: Modeling and Simulation

- System Toolboxes provide system design and modeling techniques in MATLAB
  - Pre-defined algorithms enable faster design iterations
  - Streaming simulation techniques enable modeling of real-time signal processing systems
  - Object-oriented implementation of algorithms speed up some system simulations
  - Support for fixed-point and C code generation enable design flow continuity in MATLAB
<table>
<thead>
<tr>
<th>Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Signal Processing Systems Design with MATLAB</strong></td>
</tr>
<tr>
<td>• Modeling and Simulation</td>
</tr>
<tr>
<td>• DSP/Communication/Phased Array System Toolbox</td>
</tr>
<tr>
<td><strong>Conversion to Fixed Point Conversion</strong></td>
</tr>
<tr>
<td>• Fixed Point Design</td>
</tr>
<tr>
<td><strong>Automatic Code Generation</strong></td>
</tr>
<tr>
<td>• MATLAB/Simulink to C/HDL</td>
</tr>
<tr>
<td><strong>Summary and Wrap-up</strong></td>
</tr>
</tbody>
</table>
Fixed Point Design
# Fixed Point Design: Motivation

<table>
<thead>
<tr>
<th>Consideration</th>
<th>Fixed Point</th>
<th>Floating Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM and ROM consumption</td>
<td>Small</td>
<td>Large</td>
</tr>
<tr>
<td>Execution time</td>
<td>Faster</td>
<td>Slower</td>
</tr>
<tr>
<td>Hardware power consumption</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Development time</td>
<td>Long</td>
<td>Short</td>
</tr>
<tr>
<td>Implementation complexity</td>
<td>More complex. Control of word length, rounding mode, saturation...</td>
<td>Less</td>
</tr>
<tr>
<td>Error Prone</td>
<td>Harder to develop. More prone to programming errors</td>
<td>Easier to develop</td>
</tr>
</tbody>
</table>
Fixed Point Design: Pitfalls

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

![Quantization Diagram]

- Overflow
- Integer + sign
- Fractional
- Quantization
- L
- L-N
- N
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)
Summary:
Fixed-Point C Code Generation

- Perform fixed-point system design and prototyping activities directly in MATLAB

- 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

- Generate fixed-point C code directly
  - Automatically generated C code is correct by construction
  - Reduce verification effort and cost
## Agenda

<table>
<thead>
<tr>
<th>Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Signal Processing Systems Design with MATLAB</strong></td>
</tr>
<tr>
<td>• Modeling and Simulation</td>
</tr>
<tr>
<td>• DSP/Communication/Phased Array System Toolbox</td>
</tr>
<tr>
<td><strong>Conversion to Fixed Point Conversion</strong></td>
</tr>
<tr>
<td>• Fixed Point Design</td>
</tr>
<tr>
<td><strong>Automatic Code Generation</strong></td>
</tr>
<tr>
<td>• MATLAB/Simulink to C/HDL</td>
</tr>
<tr>
<td><strong>Summary and Wrap-up</strong></td>
</tr>
</tbody>
</table>
Automatic Code Generation
**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
Algorithm Design and Code Generation in MATLAB

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

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

```
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 \
        \[Image\]

7 Lines of MATLAB
107 Lines of C
```
Fixed Point Design in MATLAB

- Run MATLAB code with floating point data types
- Collect histograms for signals
- Simulation results for all variables
- Analyze simulation min/max
Choosing the Right Deployment Solution
MATLAB Coder and MATLAB Compiler

<table>
<thead>
<tr>
<th>Feature</th>
<th>MATLAB Coder</th>
<th>MATLAB Compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>Portable and readable C source code</td>
<td>Executable or software component/library</td>
</tr>
<tr>
<td>MATLAB support</td>
<td>Subset of language</td>
<td>Full language</td>
</tr>
<tr>
<td></td>
<td>Some toolboxes</td>
<td>Most toolboxes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Graphics</td>
</tr>
<tr>
<td>Runtime requirement</td>
<td>None</td>
<td>MATLAB Compiler Runtime (MCR)</td>
</tr>
<tr>
<td>License model</td>
<td>Royalty-free</td>
<td>Royalty-free</td>
</tr>
</tbody>
</table>
Algorithm to HDL Workflows

1. Simulink to HDL (with MATLAB and Stateflow)
2. MATLAB to HDL
3. Hybrid workflow

VHDL & Verilog
MATLAB to HDL workflow

- Automated Floating to Fixed-Point conversion
- Generate optimized HDL code
- Integrated verification and implementation

Easily connect to Simulink and Xilinx System Generator
Fixed-Point Analysis

Corner Detection

- Convert floating point to **optimized** fixed-point models
  - Automatic tracking of signal range for both Simulink blocks and MATLAB function block
  - **Mathematically** calculate signal range
  - **Word / Fraction** length scaling
Summary

- Pre-defined algorithms enable faster design iterations
- Object-oriented implementation of algorithms with simulation speed-up
- Automatic fixed-point code generation with effort and cost reduction
- Reduce verification time with HDL/FPGA Co-simulation