Moving MATLAB® Algorithms into Complete Designs with Fixed-Point Simulation and Code Generation

Houman Zarrinkoub, PhD.
Product Manager
Signal Processing Toolboxes
The MathWorks Inc.
Outline

- Challenges in fixed-point signal processing
- Traditional design workflow
- Advantages of Model-Based Design workflow
  - Streamlined Float-to-fixed conversion
  - Acceleration of fixed-point simulation
  - Automatic C code generation
- Demo
- Summary
Fixed-point signal processing applications

- **Tasks**
  - Design and analyze fixed-point algorithms
  - Verify fixed point implementations

- **Hardware targets**
  - FPGA or ASIC
  - Fixed-point DSP chip
Traditional Development flow

Requirements and Specifications
- Text-based
  - Prevents rapid iteration

Design
- Physical prototypes
  - Incomplete and expensive

Implementation
- Manual coding
  - Introduces human error

Test and Verification
- Traditional testing
  - Errors found too late in the process
Multiple truths in traditional workflows

Re-implement as you go down the level of abstraction
Examine a fixed-point algorithm design

Traditional workflow

1. Set-up simulation flow
2. Express your floating-point algorithm
   - Focus on algorithmic integrity, proof of concept
3. Simulate (floating-point)
   - Iterate on algorithm trade-offs
   - Validate against requirements

4. Convert design to fixed-point
   - Focus of design viability based on implementation constraints
5. Simulate (fixed-point)
   - Iterate on implementation trade-offs
   - Validate against original requirements

6. Generate code for implementation
7. Validate and verify design after deployment

MATLAB

C/C++

Assembly or HDL

MathWorks
Aerospace and Defense Conference '07
Problems with traditional workflow

- **Multiple truths (Copies of same algorithm)**
  - Floating-point M code
  - Floating-point C code
  - Fixed-point C code
  - Assembly code
  - Verilog/VHDL code

- **Error-prone process**
  - Using different tools
  - Exchange data across tools
  - Multiple update/test of code
Model-Based Design Workflow

Requirements and Specifications

Design

Implementation

Test and Verification

Executable models
- Unambiguous
- Only “one truth”

Simulation
- Reduces “real” prototypes
- Systematic “what-if” analysis

Automatic code generation
- Minimizes coding errors

Test with Design
- Detects errors earlier

Continuous Verification

Model Elaboration

Aerospace and Defense Conference ’07
Examine a fixed-point algorithm design

Model-based Design workflow

1. Set-up simulation flow
2. Express your floating-point algorithm
   - Focus on algorithmic integrity, proof of concept
3. Simulate (floating-point)
   - Iterate on algorithm trade-offs
   - Validate against requirements
4. Convert design to fixed-point
   - Focus on design viability based on implementation constraints
5. Simulate (fixed-point)
   - Iterate on implementation trade-offs
   - Validate against original requirements
6. Generate code for implementation
7. Validate and verify design after deployment
Advantages of Model-Based Design workflow

- Maintain **One Truth**
- **One integrated design environment**
- MATLAB benefit:
  - Integrated visualization, analysis & design
- No sacrifice of **simulation speed**
- Automatic path to implementation
How does Model-Based Design make fixed-point design faster and easier?

- Streamline process of converting your MATLAB algorithms to fixed-point

- Simulate fixed-point algorithms with large data sets at compiled-C-code speed

- Integrate with system-level design in Simulink

- Generate embeddable C code for implementation with Real-Time Workshop®
Streamlined floating-to-fixed conversion: introducing Data-type override

- Turn on the logging mode
- Set data type override parameters
- Observe dynamic range of variables in your M-code
- Set the best fixed-point attributes to avoid overflow/underflow & large quantization errors
Tools for scaling a fixed-point variable

Data logging

Steps involved with dynamic range analysis to convert a design into fixed-point

1. Compute the range based on the min/max logs
2. Compute the integer part to fit variable within range
3. Compute the fraction length as the rest of bit budget
4. Construct the fixed-point numeric type object

```matlab
>> % Configuration
>> Simulink.set_param('loggingmode','on');

% Data Logging
>> A = max(abs(double(minlog(x))),abs(double(maxlog(x))));
>> integer_part = ceil(log2(A));
>> word_length = 32; is_signed = 1;
>> fraction_length = word_length - integer_part - 16; % 16-bit signed
>> T = numerictype(is_signed, word_length, fraction_length);
```
Fixed-point acceleration: introducing new `emlmex` function

- Fast simulation through code generation
- Automatic generation of C-MEX function from M-function
- M-code confined to embedded MATLAB language subset
- Compile C-code execution speed (beyond 100x acceleration in MATLAB)
Integrate MATLAB design with Simulink

Embedded MATLAB Function block

Change parameters and run Simulink® simulations from MATLAB

Integration of Embedded MATLAB Functions in Simulink

MathWorks
Aerospace and Defense Conference ’07
Automatic fixed-point C code generation
Real-Time Workshop

Enabled via Simulink Fixed-point
Real-Time Workshop®
Real-Time Workshop®
Embedded Coder
Supports up to 32-bit fixed-point numbers
Uses only native C integer data types
Hands-on Demonstration

1. Implement the algorithm with floating-point data types in M.
2. Convert to fixed-point data types in M and run with default settings; observe scaling issues!
3. Log the full numerical range of variables (data logging and data type override)
4. Use the logged minimum and maximum values to set the fixed-point scaling.
5. Validate the fixed-point solution interactively.
For more information

- Fixed-point signal processing webinars
  - Fixed-Point Programming in MATLAB
  - Fixed-Point Signal Processing with MATLAB and Simulink

- About MATLAB and Simulink signal processing products
  - Relevant product demos
  - User-contributed examples in MATLAB Central
    - [http://www.mathworks.com/matlabcentral](http://www.mathworks.com/matlabcentral)
Summary

Model-based design
- Single-truth, integrated design environment for development of a design from idea all the way to realizable implementation

Benefits
- Integrated modeling, simulation and prototyping for signal processing systems
- Easy conversion to fixed-point data types and trade-off analyses
- Automatic generation of C-code for DSPs
- Construct test harnesses for real-time hardware verification