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**

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Set-up simulation flow</td>
</tr>
<tr>
<td>2.</td>
<td>Express your floating-point algorithm</td>
</tr>
<tr>
<td></td>
<td>- Focus on algorithmic integrity, proof of concept</td>
</tr>
<tr>
<td>3.</td>
<td>Simulate (floating-point)</td>
</tr>
<tr>
<td></td>
<td>- Iterate on algorithm trade-offs</td>
</tr>
<tr>
<td></td>
<td>- Validate against requirements</td>
</tr>
<tr>
<td>4.</td>
<td>Convert design to fixed-point</td>
</tr>
<tr>
<td></td>
<td>- Focus of design viability based on implementation constraints</td>
</tr>
<tr>
<td>5.</td>
<td>Simulate (fixed-point)</td>
</tr>
<tr>
<td></td>
<td>- Iterate on implementation trade-offs</td>
</tr>
<tr>
<td></td>
<td>- Validate against original requirements</td>
</tr>
<tr>
<td>6.</td>
<td>Generate code for implementation</td>
</tr>
<tr>
<td>7.</td>
<td>Validate and verify design after deployment</td>
</tr>
</tbody>
</table>
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

The MathWorks
MATLAB® & SIMULINK®
Aerospace and Defense Conference ’07
Examine a fixed-point algorithm design

Model-based Design workflow

1. Set-up simulation flow
   - MATLAB & Simulink
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 makes 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

```
>> fspref('loggingmode','on');
>> 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 - double(logical(is_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

Embedded MATLAB Function

Integration of Embedded MATLAB Functions in Simulink
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

// Video Processing Blockset 2D Resize (svipresize) - 
/* Perform on-the-fly interpolation */
int32_T holdInt0;
int32_T accum0=0, accum2=0;
int32_T accum3;
int32_T prec;
int32_T row, col, index = 0, indexSet = 0;
/* First resize along x direction. */
for (col = 0; col < 62; col++) {
  int_T floorPos, intPart, nextIndex;
  holdInt0 = col;
  acc accum0 = svip_int32_652_x32_prs_sat_nearest(holdInt0, 0);
  // Use Simulink fixed-point support
  holdInt0 =ceil(18,accum0);
  floorPos = holdInt0;
  intPart = floorPos*18;
  nextIndex = (floorPos < 23) ? intPart+18 : intPart;
  ACCUM_TRUNC_32_32_SAT(accum0, F32FIX_32_32_S16_S10_3A
  // Now we calculate gFracFact = 1 - fracFact */
  /* Linear Interpolation. */
  holdInt0 = 1;
  accum0 = F32FIX_32_32_S16_S10_SAT(holdInt0);
  ACCUM_TRUNC_32_32_SAT(accum0, accum0);
  for (row = 0; row < 18; row++) {
    int_T index = intPart;
    int_T index = index+row;
    accum3 = mul_e32_x32_x32_prs_sat_nearest((int32_T)tdb,
    accum2);
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