Modeling a 4G LTE System in MATLAB

Part 3: Path to implementation (C and HDL)

Houman Zarrinkoub PhD.
Signal Processing Product Manager
MathWorks
houmanz@mathworks.com
LTE Downlink processing

Transport block CRC attachment
\rightarrow
Code block segmentation
\rightarrow
Channel coding
\downarrow
Rate matching
\downarrow
Code block attachment
\rightarrow
Advanced channel coding

MIMO
\downarrow
OFDM
\rightarrow
antenna ports
\rightarrow
OFDM signal generation

Scrambling \rightarrow Modulation mapper \rightarrow Layer Mapper \rightarrow Precoding
\rightarrow Resource element mapper \rightarrow OFDM signal generation

Scrambling \rightarrow Modulation mapper

(a) Transport channel processing for DL-SCH

(b) Overview of downlink physical channel processing

Adapt Everything

codewords

layers
Why Engineers translate MATLAB to C today?

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
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
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.

<table>
<thead>
<tr>
<th>Matrices and Arrays</th>
<th>Data Types</th>
<th>Programming Constructs</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Matrix operations</td>
<td>• Complex numbers</td>
<td>• Arithmetic, relational, and logical operators</td>
<td>• MATLAB functions and sub-functions</td>
</tr>
<tr>
<td>• N-dimensional arrays</td>
<td>• Integer math</td>
<td>• Program control (if, for, while, switch )</td>
<td>• Variable length argument lists</td>
</tr>
<tr>
<td>• Subscripting</td>
<td>• Double/single-precision</td>
<td></td>
<td>• Function handles</td>
</tr>
<tr>
<td>• Frames</td>
<td>• Fixed-point arithmetic</td>
<td></td>
<td>Supported algorithms</td>
</tr>
<tr>
<td>• Persistent variables</td>
<td>• Characters</td>
<td></td>
<td>• &gt; 400 MATLAB operators and functions</td>
</tr>
<tr>
<td>• Global variables</td>
<td>• Structures</td>
<td></td>
<td>• Signal processing</td>
</tr>
<tr>
<td></td>
<td>• Numeric classes</td>
<td></td>
<td>• Communications</td>
</tr>
<tr>
<td></td>
<td>• Variable-sized data</td>
<td></td>
<td>• Computer vision</td>
</tr>
<tr>
<td></td>
<td>• System objects</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Fixed-Point Design: Motivation

- ASIC/FPGA or fixed-point DSP implementation
- Saves power and/or cost
- Introduces quantization errors

![Diagram showing fixed-point design with overflow, integer + sign, fractional, and quantization]

- Word length and Fraction Length must be specified
  - For every variable
- Degradation must be analyzed
Doing Fixed-Point in C is Hard

- 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 Specification in MATLAB

- Quantizing data:
  \[ x_{\text{fxp}} = \text{fi}(x, 1, 16, 15) \ percent\ 16\ bits, \ 15\ fractional\ bits \]

- Architecture specification:
  - 16,32,40 bits (DSP) vs flexible number of bits (FPGA)
  - Arithmetic rules: full precision, keep MSB
MATLAB Accelerates Fixed-Point Design

- Streamline conversion of floating-point algorithms to fixed-point
  - *NumericTypeScope* tool

- Simulate fixed-point algorithms
  - Directly in MATLAB
  - Handle large data sets
  - Simulate with compiled-C-code speed

- Generate efficient and portable C code with MATLAB Coder
  - *codegen* command
Path to implementation

- Bring it all to Simulink

- Elaborate your design
  - Model System-level inaccuracies
  - Compensate
  - Add fixed-point
  - Generate HDL
MATLAB to Hardware
Why do we use FPGAs?

- Customized interfaces to peripherals
- High-speed communication interfaces to other processors
- Finite state machines, digital logic, timing and memory control
- High speed, highly parallel DSP Algorithms

We are going to focus on this use case today.
Separate Views of DSP Implementation

System Designer

- Algorithm Design
  - Fixed-Point
  - Timing and Control Logic
  - Architecture Exploration
  - Algorithms / IP

- System Test Bench
  - Environment Models
  - Analog Models
  - Digital Models
  - Algorithms / IP

- FPGA Requirements
  - Hardware Specification
  - Test Stimulus

FPGA Designer

- RTL Design
  - IP Interfaces
  - Hardware Architecture

- RTL Verification
  - Behavioral Simulation
  - Functional Simulation
  - Static Timing Analysis
  - Timing Simulation
  - Back Annotation

- Implement Design
  - Synthesis
  - Map
  - Place & Route

- Hardware
What we will learn today

MATLAB to Hardware

- Convert design to fixed-point
  - Use Fixed-point Tool
  - Use `NumericTypeScope` in MATLAB
  - Verify against floating-point design

- Serialize design

- Implementation using HDL Coder
  - Verify through software and/or hardware co-simulation
OFDM Transmitter

```
45  % IFFT processing
46  x = ifft(tmp, N, 1);
47  x = x.*sqrt(N/sqrt(N/len));
48
49  % Add cyclic prefix per OFDM symbol per antenna port
50  % and serialize over the subframe (equal to 2 slots)
51
52  % For a subframe of data
53  y = complex(zeros(subframeLen, numLayers));
54  for j = 1:2 % Over the two slots
55      % First OFDM symbol
56      y((j-1)*slotLen+(1:cpLen0), :) = x((N-cpLen0+1):N, :);  
57      y((j-1)*slotLen+cpLen0+(1:N), :) = x(1:N, (j-1)*7+1, :);
58
59  % Next 6 OFDM symbols
60  for k = 1:6
61      y((j-1)*slotLen+cpLen0+k*N+(k-1)*cpLenR+(1:cpLenR), :) =
62      y((j-1)*slotLen+cpLen0+k*N+k*cpLenR+(1:N), :)
63  end
64  end
```
MATLAB to Hardware

```matlab
ifft(x, 2048, 1)
```

- **Issue #1**
  - ‘x’ is 2048x4 matrix
  - MATLAB does 2048-pt FFT along first dimension
  - Output is also 2048x4
  - Cannot process samples this way in hardware!
  - Serialize design

- **Issue #2**
  - MATLAB does double-precision floating-point arithmetic
  - Floating point is expensive in hardware (power and area)
  - Convert to fixed-point
HDL Workflow

- **Floating Point Model**
  - Satisfies System Requirements
    - Executable Specification
  - MATLAB and/or Simulink Model

- **Model Elaboration**
  - Develop Hardware Friendly Architecture in Simulink
  - Convert to Fixed-Point
    - Determine Word Length
    - Determine Binary Point Location

- **Implement Design**
  - Generate HDL code using Simulink HDL Coder
  - Import Custom and Vendor IP

- **Verification**
  - Software co-simulation with HDL simulator
  - Hardware co-simulation
Divide and conquer

Save simulation data to use in development
MATLAB-based FFT

- Can be done
- Resources
  - 120 LUTs (1%)
  - 32 Slices (1%)
  - 0 DSP48s
- Need 2048-point FFT
- 32-point FFT chokes synthesis tool!

```matlab
18 - u = complex(bu_r, bu_i);
19
20 - y = fft4(N, u);
21
22 - end
23
24 - function x = fft4(N, u)
25 - nt = numerictype(u);
26 - fm = fimath(u);
27 - x = complex(fi(zeros(4,1), nt, fm));
28 - tu1 = complex(fi(zeros(2,1), nt, fm));
29 - tu2 = complex(fi(zeros(2,1), nt, fm));
30 - [tu1(1), tu1(2)] = bfly2(u(1), u(2));
31 - [tu2(1), tu2(2)] = bfly2(u(3), u(4));
32
33 - % typical butterfly structure in an FFT
34 - %
35 - % u0 =--------------------------------> tu0
36 - % |                                    |
37 - % |                                    |    
38 - % |                                    |     |
39 - % |                                    |       |
40 - % |                                    |         |
41 - % |                                    |            |
42 - % |                                    |              |
43 - % |                                    |                |
44 - % |                                    |                  |
45 - % |                                    |                    |
46 - % u1 =--------------------------------> tu1
```
Re-implement using Simulink blocks
compare against original code
Convert to fixed-point
compare against original code
What you just saw

- Simulink Fixed-Point to model fixed-point data types
  - Word lengths
  - Fraction lengths

- Fixed-Point Tool
  - Monitoring signal min/max, overflow
  - Optimization of data types
Use original testbench to optimize settings
Analyze BER to determine word length

- Anything beyond 8 bits is “good enough”
Recall

\texttt{ifft(x, 2048, 1)}

- **Issue #1**
  - ‘x’ is 2048x4 matrix
  - MATLAB does 2048-pt FFT along first dimension
  - Output is also 2048x4
  - Cannot process samples this way in hardware!
  - Serialize design

- **Issue #2**
  - MATLAB does double-precision floating-point arithmetic
  - Floating point is expensive in hardware (power and area)
  - Convert to fixed-point
Serial & Fixed-point

“HDL ready”
Automatically Generate HDL Code
Simulink HDL Coder
What You Just Saw - Workflow Advisor

1. Select ASIC, FPGA, Or FPGA Board Target
2. Prepare Model For HDL Code Generation
3. Generate HDL Code
4. Physical Design and Critical Path Highlighting
5. Program FPGA
What You Just Saw – Generated HDL Code

Generated Source Files
- Algorithm_pkg.vhd
- EQ_Parameters.vhd
- Filter.vhd
- filter_bank_left.vhd
- filter_subsystem.vhd
- Algorithm.vhd

```vhdl
51   -- Component Configuration Statements
52   FOR ALL : filter_bank_left
53       USE ENTITY work.filter_bank_left(rtl);
54
55   -- Signals
56   SIGNAL filter_bank_left_out1 : std_logic_vector(15 DOWNTO 0); -- ufix16
57   SIGNAL filter_bank_right_out1 : std_logic_vector(15 DOWNTO 0); -- ufix16
58
59   BEGIN
60       -- <S6>/filter_bank_left
61       u_filter_bank_left : filter_bank_left
62           PORT MAP( clk => clk,
63           reset => reset,
64           enb => enb,
65           input => LeftIn, -- sfix16_En15
66           parameters => Parameters, -- uint8 [10]
67           gain => Gain, -- sfix16_En11
68           data_out => filter_bank_left_out1 -- sfix16_En8
69       );
70
71       -- <S6>/filter_bank_right
72       u_filter_bank_right : filter_bank_left
73           PORT MAP( clk => clk,
74           reset => reset,
75           enb => enb,
76           input => RightIn, -- sfix16_En15
77           parameters => Parameters, -- uint8 [10]
78           gain => Gain, -- sfix16_En11
79           data_out => filter_bank_right_out1 -- sfix16_En8
80       );
81
82       LeftOut <= filter_bank_left_out1;
83
84       RightOut <= filter_bank_right_out1;
```
LTE Frame Structure (FDD)

- 1 Frame = 10ms
- 10 sub-frames per frame
- 2 slots per sub-frame (1 slot = .5ms)
- 7 OFDM symbols per slot
  - 2048 subcarriers in our simulation
  - IFFT output sample time

\[
\frac{0.5ms/7}{2048} = 3.4877 \times 10^{-8} \text{s OR } 28.672MHz
\]

(30.72MHz after cyclic prefix)
Frame and Slot Structure

30.72 MHz = 15000 * 2048

25 * cdma2000 = 8 * W-CDMA = 30.72 MHz

- If we do 2048-length FFT, we can have 15 of them per millisecond
- But we need a cyclic prefix

=> Choose to have 14 symbols/ms (1 sub-frame)
  - Split into two slots of 7 symbols
Review: Automatic HDL Code Generation

- Readable, portable HDL code
- Target ASIC and FPGA
- Standard Simulink libraries
- Push-button programming of Xilinx and Altera FPGA
- Optimize for area and speed
- Code traceability between model and code
FPGA Prototyping

- **Shorter iteration cycles**
  - Automatic HDL code generation
  - Integrated HDL verification

- **Flexible automatic HDL Code generation**
  - Speed Optimization
  - Area Optimization
  - Power saving options
  - Resource utilization
  - Validation models
HDL Coder
Generate VHDL and Verilog Code for FPGA and ASIC designs

MATLAB  Simulink

HDL Coder

Verilog and VHDL

New: MATLAB to HDL

- Automatic floating-point to fixed-point conversion
- HDL resource optimizations and reports
- Algorithm-to-HDL traceability
- Integration with simulation & synthesis tools
HDL Verifier
Verify VHDL and Verilog code using cosimulation and FPGAs

- Support for 15 Altera and Xilinx FPGA boards
- Use with:
  - HDL Coder
  - Hand-written HDL code

New:
FPGA Hardware-in-the-Loop Verification

MATLAB  Simulink

HDL Verifier

ModelSim® and Incisive®
Xilinx® and Altera® boards
HDL Workflow

- **Floating Point Model**
  - Satisfies System Requirements
    - Executable Specification
  - MATLAB and/or Simulink Model

- **Model Elaboration**
  - Develop Hardware Friendly Architecture
  - Convert to Fixed-Point
    - Determine Word Length
    - Determine Binary Point Location

- **Implement Design**
  - Generate HDL code using HDL Coder
  - Import Custom and Vendor IP

- **Verification**
  - Software co-simulation with HDL simulator
  - Hardware co-simulation
Communications Prototyping with USRP2

- MATLAB and Simulink for communications system design and verification
  - Behavioral modeling, fixed-point, C/HDL code-gen

- Reuse simulation model as verification environment with interface to USRP2 radio

- Experiment and verify with real-world signals and systems
Summary

- MATLAB is an ideal language for LTE modeling and simulation
- Communications System Toolbox extend breadth of MATLAB modeling tools
- You can accelerate simulation with a variety of options in MATLAB
  - Parallel computing, GPU processing, MATLAB to C
- Address implementation workflow gaps with
  - Automatic MATLAB to C/C++ and HDL code generation
  - Hardware-in-the-loop verification