Generating, Optimizing and Verifying HDL Code with MATLAB and Simulink

Puneet Kumar
Application Engineering Team
Agenda

- Integrated Workflow for FPGA/ASIC Development

- Automatic HDL Code Generation & Optimization
  - Refining DUT from floating to fixed point
  - Automatic HDL Code Generation from MATLAB, Simulink, and Stateflow

- Integrated HDL Verification
  - Automatically Measuring Test Coverage
  - Generating Co-Simulation Test benches
  - FPGA-in-Loop Verification with Xilinx ML 605 FPGA Board

- Q&A
Algorithm Development Process

Requirements

Research & Design

- Explore and discover
- Gain insight into problem
- Evaluate options, trade-offs

Implementation

Desktop
- .dll
- .exe
- .c, .cpp

Embedded
- C
- VHDL / Verilog
- Structured Text

Test & Verification

Design → Test → Elaborate
The Algorithm Design Challenge

- How can we:
  - Implement designs faster?
  - Reuse designs on a variety of hardware?
Solution: C and HDL Code Generation

- Design, execute, and verify algorithms in MATLAB
- Automatically generate C or HDL code
- Deploy generated code on hardware
Code Generation Products for VHDL/Verilog

- **HDL Coder™**
  Automatically generate VHDL or Verilog from MATLAB code and Simulink Model

- **MATLAB® Coder™**
  Automatically generate C and C++ from MATLAB code

- **Fixed-Point Designer™**
  Provides fixed-point data types and arithmetic
Best Practice 1:  
Algorithm and System Design with Fixed-Point Quantization Analysis

Use **modeling** and **simulation** to **optimize** at the **system level** using **Simulink** and **Simulink Fixed Point**
Best Practice 1: Algorithm and System Design with Fixed-Point Quantization Analysis

Best Practice 2: Automatic HDL Code Generation

Automatically generate readable, traceable HDL code for FPGA and ASIC designs using HDL Coder
Best Practice 1: Algorithm and System Design with Fixed-Point Quantization Analysis

Best Practice 2: Automatic HDL Code Generation

Best Practice 3: HDL Cosimulation

Reuse system-level test benches with cosimulation for HDL verification using HDL Verifier
Best Practice 1:
Algorithm and System Design with Fixed-Point Quantization Analysis

Best Practice 2:
Automatic HDL Code Generation

Best Practice 3:
HDL Cosimulation

Implement Design

Best Practice 4:
FPGA Hardware-in-the-Loop

Enable regression testing with FPGA-in-the-loop simulation using HDL Verifier
Audio Equalizer
(A Brief Example)
Challenge
Accelerate the development of optimized digital receiver chains for wireless RF devices

Solution
Use MathWorks tools for Model-Based Design to generate production VHDL code for rapid FPGA and ASIC implementation

Results
- Prototypes created 50% faster
- Verification time reduced from weeks to days
- Optimized, better-performing design delivered

"Writing VHDL is tedious, and the handwritten code still needs to be verified. With Simulink and Simulink HDL Coder, once we have simulated the model we can generate VHDL directly and prototype an FPGA. It saves a lot of time, and the generated code contains some optimizations we hadn’t thought of.”

Frantz Prianon
Semtech
HDL Coder

Generate VHDL and Verilog Code for FPGA and ASIC designs

- Automatic floating-point to fixed-point conversion
- HDL resource optimizations and reports
- Algorithm-to-HDL traceability
- Integration with simulation & synthesis tools

New: MATLAB to HDL
Model-Based Design flow using MATLAB/Simulink
from Algorithm to FPGA Implementation
Algorithm to HDL Workflows

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

VHDL & Verilog
Using HDL Coder: 5-Step Workflow

**Prepare** your MATLAB algorithm for code generation
- Use supported language features
- Make implementation choices

**Fixed-Point** MATLAB code generation from your floating-point design using your MATLAB TestBench
- Accelerate TestBench for fast simulation
- Automatically propose Fixed-Point type
- Iterate data-type customization to optimize
- Verify Fixed-Point code against original Floating-Point code.

**Generate** synthesizable RTL & TestBench code from Fixed-Point MATLAB code for final use
- Iterate your MATLAB code to optimize
- Implement as source, executable or library

**Simulate** the generated HDL code with test vectors from the test bench using the specified simulation tool

**Synthesis, Place and Route** the generated RTL code by creating project with ISE/Quartus II
- Check timing analysis report to optimize
Simulink Library Support for HDL

HDL Supported Blocks

- 180 blocks supported

- Core Simulink Blocks
  - Basic and Array Arithmetic, Look-Up Tables, Signal Routing (Mux/Demux, Delays, Selectors), Logic & Bit Operations, Dual and single port RAMs, FIFOs, CORDICs, Busses

- Signal Processing Blocks
  - NCOs, FFTs, Digital Filters (FIR, IIR, Multi-rate, Adaptive), Rate Changes (Up & Down Sample), Statistics (Min/Max)

- Communications Blocks
  - Psuedo-random Sequence Generators, Modulators / Demodulators, Interleavers / Deinterleavers, Viterbi Decoders
Agenda

- Integrated Workflow for FPGA/ASIC Development

- **Automatic HDL Code Generation & Optimization**
  - Refining DUT from floating to fixed point
  - Automatic HDL Code Generation from MATLAB, Simulink, and Stateflow

- **Integrated HDL Verification**
  - Automatically Measuring Test Coverage
  - Generating Co-Simulation Test benches
  - FPGA-in-Loop Verification with Xilinx ML 605 FPGA Board

- Q&A
From Algorithm to Synthesizable RTL
Best Practice 1
Use modeling and simulation to optimize at the system level

- Convert floating point to **optimized** fixed-point models
  - Automatic tracking of signal range (also intermediate quantities)
  - Word / Fraction lengths recommendation
- Bit-true models in the same environment

Optimize on fixed-point word-length to reduce area and power
Best Practice 2
Automatically generate readable, traceable HDL code for FPGA and ASIC designs

Automatically generate bit true, cycle accurate HDL code from Simulink, MATLAB and Stateflow

Full bi-directional traceability!!

Requirements
Speed optimization
Use pipelining to improve speed

Critical Path highlighting makes it easier to identify the true bottlenecks of the system

Advanced Pipelining options for pipeline distribution and automatic delay compensation
Area optimization

Use sharing and streaming to reduce area

<table>
<thead>
<tr>
<th>Resource</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multipliers</td>
<td>10</td>
</tr>
<tr>
<td>Adders/Subtractors</td>
<td>36</td>
</tr>
<tr>
<td>Registers</td>
<td>292</td>
</tr>
<tr>
<td>RAMs</td>
<td>2</td>
</tr>
<tr>
<td>Multiplexers</td>
<td>116</td>
</tr>
</tbody>
</table>

Automatically generated validation models

Resource utilization reports provide early feedback on resource utilization
Agenda

- Integrated Workflow for FPGA/ASIC Development

- Automatic HDL Code Generation & Optimization
  - Refining DUT from floating to fixed point
  - Automatic HDL Code Generation from MATLAB, Simulink, and Stateflow

- Integrated HDL Verification
  - Automatically Measuring Test Coverage
  - Generating Co-Simulation Test benches
  - FPGA-in-Loop Verification with Xilinx ML 605 FPGA Board

- Q&A
Which tests do you perform today?
Verification Landscape:

Model
- Requirements
- Functional
- Equivalence
- Coverage
- Property Proving
- Virtual Platforms

VHDL / Verilog
- Requirements
- Equivalence
- Coverage
- Assertions

FPGA
- Equivalence
- Regression
- Timing Analysis

```vhdl
-- <S1>/Unit Delay
Unit_Delay_process : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    Unit_Delay_out1 <= to_signed(0, 16);
  ELSIF clk'EVENT AND clk = '1' THEN
    IF enb = '1' THEN
      Unit_Delay_out1 <= filter_subsystem_out1_signed;
    END IF;
  END IF;
END PROCESS Unit_Delay_process;

leftOut <= std_logic_vector(Unit_Delay_out1);
```
Verification Challenges:
Stimuli-Driven Test Bench in HDL Simulators

- Digital waveforms are difficult to analyze
  - Application specific analysis methods are needed
- How to get test vectors to achieve 100% test coverage?
  - Formal methods to derive required test cases
Verification Landscape Solution: Re-use System Level Test Bench

- Requirements
- Functional
- Equivalence
- Coverage
- Property Proving
- Virtual Platforms

- Requirements
- Equivalence
- Coverage
- Assertions

- Equivalence
- Regression
- Timing Analysis
Audio Equalizer

- Bank of 10 filters
  - Controllable by up to +/-6dB
- 5 pre-programmed user settings for
  - Rock, Pop, Jazz, Classical, Vocal
- Fits into available FPGA space
- No dead-locks or unreachable states
- Sounds good
Automatically Measuring Test Coverage

Audio Equalizer

Summary

<table>
<thead>
<tr>
<th>Model Hierarchy/Complexity:</th>
<th>Test 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. EqualizerAlgorithm</td>
<td>47 69%</td>
</tr>
<tr>
<td>2. EQ_Parameters</td>
<td>44 69%</td>
</tr>
<tr>
<td>3. effect_selection</td>
<td>36 64%</td>
</tr>
<tr>
<td>4. SF_effect_selection</td>
<td>35 64%</td>
</tr>
<tr>
<td>5. parameter_lookup</td>
<td>7 100%</td>
</tr>
</tbody>
</table>

Automatically collect and report test coverage
Test Generation for 100% Coverage

Audio Equalizer

Not the same as HDL code coverage

Automatically generate tests to reach coverage objectives
Integrate with HDL Code Coverage

Audio Equalizer

Re-use test benches for equivalence checking and code coverage analysis

Integrate with leading HDL Simulators
Best Practice 3
Re-use System Level Test Bench for HDL Verification

Model
- Requirements
- Functional
- Equivalence
- Coverage
- Property Proving
- Virtual Platforms

VHDL / Verilog
- Requirements
- Equivalence
- Coverage
- Assertions

FPGA
- Equivalence
- Regression
- Timing Analysis
- On target prototyping
HDL cosimulation to verify HDL
Re-use System Level Test Bench for HDL Verification

Re-use test benches for equivalence checking
Integrate with HDL code coverage analysis

Flexible test bench creation: closed loop, multi domain
Also works with handwritten code
Integrate with Modelsim/Questa and Incisive
Best Practice 4
Enable regression testing with FPGA-in-the-loop simulation

Model
- Requirements
- Functional
- Equivalence
- Coverage
- Property Proving
- Virtual Platforms

VHDL / Verilog
- Requirements
- Equivalence
- Coverage
- Assertions

FPGA
- Equivalence
- Regression
- Timing Analysis
- On target prototyping
FPGA-in-the-loop
Enable regression testing with FPGA-in-the-loop simulation

- Re-use test benches for regression testing
- Integrate with Altera / Xilinx FPGA Development Boards
- Flexible test bench creation: closed loop, multi domain
- Also works with handwritten code
Automation FPGA-in-the-loop Verification

Supported FPGA boards

Integration with FPGA development boards

Automatic creation of FPGA-in-the-loop verification models
Additional Methods for Verification

HDL Verification Techniques

- Generate stimuli-based test benches for standalone verification
- MATLAB based verification
Stimuli-based test benches for standalone verification

MATLAB or Simulink Test bench

MATLAB or Simulink Design
Targeted to Hardware

Stimulus

Reference Results

Automatically Generated HDL Test Bench

Stimulus

HDL Design

Actual Results

Can be used in any HDL Simulator

Automatically generate self-checking test benches
MATLAB Based Verification

- **Re-use** the MATLAB test bench
- **Accelerate Verification** with FPGA Hardware
FPGA turnkey workflow
FPGA on target prototyping

Music in ➔ Music out

Integrate with Altera / Xilinx FPGA Development Boards

Stand alone testing of algorithms on FPGA hardware

Automated workflow from model to FPGA prototype
Summary

Best Practice 1: Algorithm and System Design with Fixed-Point Quantization Analysis

Best Practice 2: Automatic HDL Code Generation

Implement Design

Best Practice 3: HDL Cosimulation

Best Practice 4: FPGA Hardware-in-the-Loop
Challenge
Accelerate the implementation of advanced thermal imaging filters and algorithms on FPGA hardware

Solution
Use MATLAB to develop, simulate, and evaluate algorithms, and use HDL Coder to implement the best algorithms on FPGAs

Results
- Time from concept to field-testable prototype reduced by 60%
- Enhancements completed in hours, not weeks
- Code reuse increased from zero to 30%

“With MATLAB and HDL Coder we are much more responsive to marketplace needs. We now embrace change, because we can take a new idea to a real-time-capable hardware prototype in just a few weeks. There is more joy in engineering, so we’ve increased job satisfaction as well as customer satisfaction.”

—Nicholas Hogasten, FLIR Systems
### Public Trainings in the next Few Months

<table>
<thead>
<tr>
<th>Course</th>
<th>Dates</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistical Methods in MATLAB</td>
<td>2 Sep – 3 Sep</td>
<td>Bangalore</td>
</tr>
<tr>
<td>MATLAB based Optimization Techniques</td>
<td>4 Sep</td>
<td>Bangalore</td>
</tr>
<tr>
<td>MATLAB Fundamentals</td>
<td>23 Sep – 25 Sep</td>
<td>Delhi</td>
</tr>
<tr>
<td>Simulink for System and Algorithm Modeling</td>
<td>26 Sep – 27 Sep</td>
<td>Delhi</td>
</tr>
<tr>
<td>MATLAB Fundamentals</td>
<td>07 Oct – 09 Oct</td>
<td>Pune</td>
</tr>
<tr>
<td>Simulink for System and Algorithm Modeling</td>
<td>10 Oct – 11 Oct</td>
<td>Pune</td>
</tr>
<tr>
<td>Generating HDL Code from Simulink</td>
<td>28 Nov – 29 Nov</td>
<td>Bangalore</td>
</tr>
</tbody>
</table>

Email: training@mathworks.in  
URL: [http://www.mathworks.in/services/training](http://www.mathworks.in/services/training)  
Phone: 080-6632-6000
Agenda

- Integrated Workflow for FPGA/ASIC Development

- Automatic HDL Code Generation & Optimization
  - Refining DUT from floating to fixed point
  - Automatic HDL Code Generation from MATLAB, Simulink, and Stateflow

- Integrated HDL Verification
  - Automatically Measuring Test Coverage
  - Generating Co-Simulation Test benches
  - FPGA-in-Loop Verification with Xilinx ML 605 FPGA Board

- Q&A
Thank You!