Accelerating FPGA/ASIC Design and Verification

MATLAB EXPO 2017

Tabrez Khan – Senior Application Engineer
Vidya Viswanathan – Application Engineer
Agenda

- Challenges with Traditional Implementation workflow
- Model-Based Design for Implementation
- Generate VHDL® and Verilog® code from MATLAB, Simulink, and Stateflow®
- Optimize the generated RTL design for area and/or speed
- Develop system-level test benches in MATLAB and Simulink for RTL verification with EDA tools
- Automate verification with FPGA-in-the-Loop
- Summary & next steps
Traditional Implementation Workflow

- Long development cycles
- Prevents short iteration cycles
- Difficult to optimize the algorithm at a system level

- **Design**
  - Algorithm Development
  - MATLAB Simulink Stateflow

- **Manual Steps**
  - Fixed Point Conversion
  - HDL Code Creation
  - HDL Verification
  - HDL Refinement
  - FPGA Verification
Separate Views of DSP Implementation

Algorithm Designer

Software Designer

Hardware Designer
Separate Views of DSP Implementation

**System Designer**
- **Algorithm Design**
  - Fixed-Point
  - Timing / 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
- **Verification**
  - Behavioral Simulation
  - Functional Simulation
  - Static Timing Analysis
  - Timing Simulation
  - Back Annotation
- **Implement Design**
  - Synthesis
  - Map
  - Place & Route
- **FPGA Hardware**
Model-Based Design for Implementation

MATLAB® and Simulink®
Algorithm and System Design

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

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

FPGA Requirements
- Hardware Specification
- Test Stimulus

RTL Design
- IP Interfaces
- Hardware Architecture

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

Implement Design
- Synthesis
- Map
- Place & Route

FPGA Hardware
Model-Based Design for Implementation

MATLAB® and Simulink®
Algorithm and System Design
Model Refinement for Hardware

Automatic HDL Code Generation

RTL Design
- IP Interfaces
- Hardware Architecture

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

Implement Design
- Synthesis
- Map
- Place & Route

FPGA Hardware

Synthesis
Map
Place & Route

Behavioral Simulation
Functional Simulation
Static Timing Analysis
Timing Simulation
Back Annotation

FPGA Hardware
Model-Based Design for Implementation

MATLAB® and Simulink®
Algorithm and System Design
Model Refinement for Hardware

Automatic HDL Code Generation ➔ HDL Co-Simulation ➔ Behavioral Simulation

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

Implement Design
- Synthesis
- Map
- Place & Route

FPGA Hardware
Model-Based Design for Implementation

MATLAB® and Simulink®
Algorithm and System Design
Model Refinement for Hardware

Automatic HDL Code Generation
HDL Co-Simulation
Behavioral Simulation
Back Annotation

Implement Design
Verification
Synthesis
Map
Place & Route
Functional Simulation
Static Timing Analysis
Timing Simulation

FPGA Hardware
Verification
Functional Simulation
Static Timing Analysis
Timing Simulation
Back Annotation

Algorithm and System Design
Model Refinement for Hardware
Automatic HDL Code Generation
HDL Co-Simulation
Behavioral Simulation
Back Annotation
Implement Design
Verification
Synthesis
Map
Place & Route
Functional Simulation
Static Timing Analysis
Timing Simulation
FPGA Hardware
Model-Based Design for Implementation

MATLAB® and Simulink®
Algorithm and System Design
Model Refinement for Hardware

Automatic HDL Code Generation
HDL Co-Simulation
Behavioral Simulation
Back Annotation

Implement Design
- Synthesis
- Map
- Place & Route

Verification
- Functional Simulation
- Static Timing Analysis
- Timing Simulation

FPGA Hardware
FPGA-in-the-Loop

FPGA Hardware
Model-Based Design for Implementation
Integrated Workflow

MATLAB® and Simulink®
Algorithm and System Design
Model Refinement for Hardware

Automatic HDL
Code Generation

HDL Co-Simulation

Behavioral Simulation
Back Annotation

Implement Design
Synthesis
Map
Place & Route

Verification
Functional Simulation
Static Timing Analysis
Timing Simulation

FPGA Hardware
FPGA-in-the-Loop

All steps from
1 single GUI
Why Model-Based Design: Achieving the Shift-Left

Reduce overall development time

- Reduced FPGA prototype development schedule
- Shorter design iteration cycle by 80%
- Improved product quality
Automatic HDL Code Generation
HDL Coder

Full bi-directional traceability!!

Automatically generate bit-true, cycle-accurate HDL code from Simulink, MATLAB and Stateflow
HDL Code Generation Example
Generate Verilog or VHDL code
Code Generation Report

- Traceability Report
- Resource Utilization Report
- Critical Path Estimation Report
What’s new?
Native Floating-Point

Generate target-independent synthesizable RTL from single-precision floating-point models

- Good for:
  - Designs with high dynamic range calculations
  - Getting started prototyping FPGAs without having to perform fixed-point conversion
- Mix integer, fixed-point, and floating point operations to balance numerical accuracy versus hardware resource usage
- Over 130 Simulink blocks supported
- Demo video

» edit hdlcoderFocCurrentFloatScript
HDL Optimizations: What, How and Why?

The three golden questions:
1. Speed: Does it meet timing?
2. Area: Does it fit on my FPGA?
3. Validation: Does it do the right thing?

HDL optimizations assists the engineer in meeting these constraints
Critical Timing Path

- Critical path highlighting
- Helps you identify speed bottlenecks
Speed Optimization

Summary Section

Critical Path Delay: 6.910 ns
Critical Path Begin: `ud8`
Critical Path End: `y_out_pre`
Highlight Critical Path: `hdl_pr\hdlsrc\symmetric_fir_fixed\criticalPathEstimated.m`

Maximum rate = 145 MHz

Critical Path Details

<table>
<thead>
<tr>
<th>Id</th>
<th>Propagation (ns)</th>
<th>Delay (ns)</th>
<th>Block Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.2980</td>
<td>0.2980</td>
<td><code>ud8</code></td>
</tr>
<tr>
<td>2</td>
<td>1.4960</td>
<td>1.1980</td>
<td><code>a1</code></td>
</tr>
<tr>
<td>3</td>
<td>5.5000</td>
<td>4.0040</td>
<td><code>m1</code></td>
</tr>
<tr>
<td>4</td>
<td>5.5000</td>
<td>0.0000</td>
<td><code>a5</code></td>
</tr>
<tr>
<td>5</td>
<td>6.9100</td>
<td>1.4100</td>
<td><code>y_out_pre</code></td>
</tr>
</tbody>
</table>

Is this the best rate that is achievable??

- Automatic pipelining
- Helps you meet speed objectives
Speed Optimization
Output Pipelining
Speed Optimization
Output Pipelining

Where do I place the pipeline registers??
Speed Optimization
Distributed Pipelining
Speed Optimization
Distributed Pipelining

Maximum rate = 235 MHz
Area Optimization

‘N’ (say 20) multipliers, each running at 1 clock cycle

1 multiplier running at ‘N’ (20) clock cycles
Area Optimization
Resource Sharing
Area Optimization

Resource Sharing

### Generic Resource Report for symmetric_fir_fixed

<table>
<thead>
<tr>
<th>Summary</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Multipliers</td>
<td>4</td>
</tr>
<tr>
<td>Adders/Subtractors</td>
<td>7</td>
</tr>
<tr>
<td>Registers</td>
<td>27</td>
</tr>
<tr>
<td>Total 1-Bit Registers</td>
<td>559</td>
</tr>
<tr>
<td>RAMs</td>
<td>0</td>
</tr>
<tr>
<td>Multiplexers</td>
<td>0</td>
</tr>
<tr>
<td>I/O Bits</td>
<td>135</td>
</tr>
<tr>
<td>Static Shift operators</td>
<td>0</td>
</tr>
<tr>
<td>Dynamic Shift operators</td>
<td>0</td>
</tr>
</tbody>
</table>

### Generic Resource Report for symmetric_fir_fixed

<table>
<thead>
<tr>
<th>Summary</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Multipliers</td>
<td>1</td>
</tr>
<tr>
<td>Adders/Subtractors</td>
<td>9</td>
</tr>
<tr>
<td>Registers</td>
<td>38</td>
</tr>
<tr>
<td>Total 1-Bit Registers</td>
<td>814</td>
</tr>
<tr>
<td>RAMs</td>
<td>0</td>
</tr>
<tr>
<td>Multiplexers</td>
<td>6</td>
</tr>
<tr>
<td>I/O Bits</td>
<td>135</td>
</tr>
<tr>
<td>Static Shift operators</td>
<td>0</td>
</tr>
<tr>
<td>Dynamic Shift operators</td>
<td>0</td>
</tr>
</tbody>
</table>
Area Optimization

Resource Sharing

Block Parameters: m4

Product
Multiply or divide inputs. Choose element-wise or matrix product and specify one of the following:
a) * or / for each input port. For example, ***/ performs the operation 'u1*u2/u3*u4'.
b) scalar specifies the number of input ports to be multiplied.
If there is only one input port and the Multiplication parameter is set to Element-wise(·*), a single * or / collapses the input signal using the specified operation. However, if the Multiplication parameter is set to Matrix(*), a single * causes the block to output the matrix unchanged, and a single / causes the block to output the matrix inverse.

Main
Signal Attributes

Number of inputs:
**

Multiplication: Element-wise(·*)

Sample time: 0.25
Not recommended for this block. Set to -1 to remove. Why?
What’s new?
Adaptive Pipelining

Specify synthesis tool and target clock frequency for automatic pipeline insertion and balancing

- Automatically inserts pipeline registers to meet target frequency
  - On by default
  - Adds pipeline registers on parallel paths to balance number of stages
- Good for:
  - Getting started prototyping FPGAs without worrying about manually inserting Delay blocks

Target Frequency = 500
Integrated HDL Verification

MATLAB® and Simulink®
Algorithm and System Design
Model Refinement for Hardware

Automatic HDL Code Generation

Implement Design
- Synthesis
- Map
- Place & Route

HDL Co-Simulation
- Behavioral Simulation
- Back Annotation

Verification
- Functional Simulation
- Static Timing Analysis
- Timing Simulation

FPGA Hardware
- FPGA-in-the-Loop
Co-Simulation with HDL Simulator

- Proof your HDL matches the MATLAB/Simulink specification
- Re-using MATLAB/Simulink testbench
FPGA-in-the-Loop (FIL) for any HDL code

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

Supported Xilinx boards
- KC 705
- ML605
- ML505
- ML506
- ML507
- XUP Atlys
- XUP-V5

Supported Altera boards
- Arria II
- Cyclone III
- DE2-115
- Cyclone IV
Automation FPGA-in-the-Loop Verification

Integration with FPGA development boards

Automatic creation of FPGA-in-the-Loop verification models

Add your own FPGA board (needs Ethernet)
New FPGA Families and Boards Supported by FIL

- FPGA Family
  - Virtex Ultrascale

- FPGA board
  - Artix-7 Arty (JTAG)
  - Virtex-7 VC709 (JTAG, PCIe)
  - Virtex Ultrascale VCU110 (JTAG)
SystemVerilog DPI Test Bench

- Previously only available via command-line interface
- Now it’s available in Config Param as well as HDL Workflow Advisor
HDL Verifier: HDL Code Coverage

Activate HDL simulator code coverage in generated test benches

- Works for cosimulation, SystemVerilog DPI, or vector-based testbenches
- Supports Mentor Graphics Questa Sim and Cadence Incisive

```makehdltb('sfir_fixed/symmetric_fir', ...
'GenerateSVDPITestBench','ModelSim', ...
'HDLCodeCoverage', 'on', )```
HDL Verifier: FPGA Data Capture

Probe internal FPGA signals to analyze in MATLAB or Simulink

- Debug signals in a free-running FPGA directly in MATLAB or Simulink
- Generates a block to add into the VHDL/Verilog design going onto the FPGA
- Collects and visualizes the data in MATLAB or Simulink
- Demo video

» generateFPGADataCaptureIP

Available as part of HDL Verifier Xilinx/Intel hardware support packages
Harris Accelerates Verification of Signal Processing FPGAs

Challenge
Streamline a time-consuming manual process for testing signal processing FPGA implementation

Solution
Use HDL Verifier to verify the HDL design from within MATLAB

Results
- Functional verification time cut by more than 85%
- 100% of planned test cases completed
- Design implemented defect-free

“HDL Verifier enabled us to greatly reduce functional verification development time by providing a direct cosimulation interface between our MATLAB model and our logic simulator. As a result, we verified our design earlier, identified problems faster, completed more tests, and compressed our entire development cycle.”

Jason Plew
Harris Corporation

Link to user story
Lockheed Martin Develops Configurable, Space-Qualified Digital Channelizer Using MathWorks Tools

Challenge
Design and implement a reconfigurable, space-qualified digital channelizer

Solution
Use Simulink to model and simulate the system, and HDL Verifier with Mentor Graphics ModelSim to verify the VHDL implementation

Results
- Verification time reduced by 90%
- Overall development time shortened by eight months
- Key algorithms reused, saving 50% of design effort on subsequent projects

"With Simulink and HDL Verifier, simulation and verification are performed in one environment. As a result, we can test the design from end to end, improving quality and ensuring design accuracy and validity."

Bradford Watson
Lockheed Martin Space Systems
Summary

- **Respect project timeline**
  - Discover issues early through simulation
  - Fast code turnarounds allow better design trade-offs

- **Collaborate in multidisciplinary teams**
  - Use one Model for Design and Implementation
  - Seamlessly integrate version management
  - Graphically compare models

- **Create working code**
  - Analyze fixed-point impact before going to implementation
  - Auto-generate bug free code
  - Verify early through co-simulation with FPGA’s

- **Achieve required efficiency**
  - Optimize through advisors and automatic optimizations
Call To Action

Learn more with recorded webinars & videos

- **Accelerate Design Space Exploration Using HDL Coder Optimizations**
- **Using HDL Coder and HDL Verifier for FPGA and ASIC Designs**
- **HDL Implementation and Verification of a High-Performance FFT**
- **Using Custom Boards for FPGA-in-the-Loop Verification**
- **A Guided Workflow for Zynq Using MATLAB and Simulink**
- **HDL Verifier: FPGA Data Capture**
Generating HDL Code from Simulink

two-day course shows how to generate and verify HDL code from a Simulink® model using HDL Coder™ and HDL Verifier™

Topics include:

- Preparing Simulink models for HDL code generation
- Generating HDL code and testbench for a compatible Simulink model
- Performing speed and area optimizations
- Integrating handwritten code and existing IP
- Verifying generated HDL code using testbench and cosimulation
Programming Xilinx Zynq SoCs with MATLAB and Simulink

two-day course focuses on developing and configuring models in Simulink® and deploying on Xilinx® Zynq®-7000 All Programmable SoCs. For Simulink users who intend to generate, validate, and deploy embedded code and HDL code for software/hardware codesign using Embedded Coder® and HDL Coder™. A ZedBoard™ is provided to each attendee for use throughout the course. The board is programmed during the class and is yours to keep after the training.

Topics include:

- Zynq platform overview and environment setup, introduction to Embedded Coder and HDL Coder
- IP core generation and deployment, Using AXI4 interface
- Processor-in-the-loop verification, data interface with real-time application
- Integrating device drivers, custom reference design
DSP for FPGAs
This three-day course will review DSP fundamentals from the perspective of implementation within the FPGA fabric. Particular emphasis will be given to highlighting the cost, with respect to both resources and performance, associated with the implementation of various DSP techniques and algorithms.

Topics include:
- Introduction to FPGA hardware and technology for DSP applications
- DSP fixed-point arithmetic
- Signal flow graph techniques
- HDL code generation for FPGAs
- Fast Fourier Transform (FFT) Implementation
- Design and implementation of FIR, IIR and CIC filters
- CORDIC algorithm
- Design and implementation of adaptive algorithms such as LMS and QR algorithm
- Techniques for synchronisation and digital communications timing recovery
Your feedback is valued.
Please complete the feedback form provided to you.