Design and Verification of FPGA Applications

Giuseppe Ridinò  
Paola Vallauri  
MathWorks  
giuseppe.ridino@mathworks.it  
paola.vallauri@mathworks.it  

Torino, 19 Maggio 2016, INAF
Agenda

- Model-Based Design for FPGA

- Generating HDL Code from MATLAB and Simulink
  - For prototyping and production
  - Optimizing code for efficiency

- Verifying HDL Designs with MATLAB and Simulink
  - Co-simulation with HDL simulators
  - FPGA-in-the-Loop verification

- Verifying HDL Designs outside MATLAB and Simulink
  - Generating code for integration with SystemC/TLM and SystemVerilog/DPI-C
Model-Based Design for FPGA

- Model multi-domain systems
- Explore and optimize system behavior
- Collaborate across multi-disciplinary teams

- Generate bit-accurate models
- Explore and optimize implementation tradeoffs
- Generate efficient code

- Verify designs to detect errors earlier in development
- Reuse testbenches
- Automate regression testing
It’s about Collaboration

- Usually, many engineers get involved in different parts of the design flow:
  - Each brings valuable expertise from their discipline
  - Model-Based Design aids collaboration across the project
    - integrating the workflow
    - providing the backbone of a common modelling environment
A Typical Model Structure

- **Algorithm** interacts with outside **environment** through other **components**.
- **Algorithm** is stimulated with **data**
- **Algorithm** performance is **analysed**.
Algorithm Development
Generation of HDL Source Code

- **HDL Coder**
  - Generation of synthesizable RTL HDL (VHDL or Verilog)

- **Support for**
  - MATLAB
  - Simulink
  - Stateflow

- **Workflow Advisor**
  - Guides through process
  - Preparing model for generation of HDL
  - Configuring HDL Generation options
  - Integrated with FPGA synthesis tools for timing annotation on model
  - Configurations for turnkey FPGA targets and IP Core generation
Simulink Library Support for HDL Generation

HDL Supported Blocks

- ~180 blocks supported

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

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

- Communications
  - Pseudo-random Sequence Generators, Modulators / Demodulators, Interleavers / Deinterleavers, Viterbi Decoders, Reed Solomon Encoders / Decoders, CRC Generator / Detector
MATLAB & Stateflow for HDL Generation

HDL Supported Blocks

- **MATLAB**
  - Relevant subset of the MATLAB language for modeling and generating HDL implementations
  - Useful MATLAB Function Block Design Patterns for HDL

- **Stateflow**
  - Modeling FSMs (Mealy, Moore)
  - Different modeling paradigms (Graphical Methods, State Transition Tables, Truth Tables)
  - Integrate MATLAB code
HDL code generation
Critical Path Highlighting and Design Review

- Feedback in Simulink
- Review results in synthesis tools
Algorithm Verification
Data-driven Verification of HDL Source Code

- Stand-alone HDL testbench
  - Stand-alone
    - Executable in any 3rd-party HDL simulator
  - Self-contained
    - Instantiated algorithmic RTL HDL (DUT)
    - Input stimuli stream at DUT top-level interface
    - Expected output stream at DUT top-level interface
  - Self-testing
    - Checks on bit and cycle accuracy

- Handwritten or generated code
  - With HDL Coder, RTL HDL and standalone testbenches are created automatically
Algorithm Verification
Co-simulation for Verification of HDL Source Code

- Co-simulation with 3rd-party HDL simulator
  - Reuse of existing testbench in MATLAB/Simulink
  - HDL code execution in 3rd-party HDL simulator
  - Flexible HDL sources
    - Handwritten or generated code
  - Automated generation of co-simulation infrastructure
  - Automatic handshaking
    - Combined analysis and debugging in both simulators
Co-simulation
Prototype your algorithm in hardware connected to the system-level test environment

- FIL simulation with FPGA development board
  - Reuse existing testbench
  - HDL code execution on FPGA
  - Handwritten or generated HDL code
  - Automated generation of co-simulation infrastructure
    - Encapsulation of algorithm within GBit Ethernet MAC, or JTAG
  - Automatic handshaking
FPGA-in-the-Loop Target Device
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

FPGA Development Board
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

Key Takeaway
Integrating with other Verification Activities

- Verification is the single biggest cost in hardware design
  - Investment in developing simulations for verification
    - SystemVerilog and UVM test frameworks
    - SystemC/TLM virtual platforms
  - Shift towards ‘model-based’ verification
    - Enabling techniques like Constrained Random testing

- Rather than recreate a behavioural model, we can reuse the assets developed in the system models in MATLAB & Simulink
  - Maintains connection with earlier part of the flow
    - Removes risk of manual error in test framework
    - Avoids duplicating effort
System Verification
Reuse of models in SystemVerilog Testbench

- Code generation translates models to other languages (e.g. C, HDL)
  - Implementation code
  - Verification models

- For verification, C code generation is convenient
  - analog and digital models
  - Wider block and language support for C generation

- HDL Verifier extends code generation tools to provide wrappers for
  - SystemVerilog DPI-C
  - SystemC TLM
Integrating DPI-C/SV into Existing Testbench

- Using a public SystemVerilog Testbench example*, adapted to execute the DPI-C as a golden reference:

Integrated Verification
Model-Based Design and SystemVerilog/SystemC
Zynq HW/SW Co-design Workflow Summary

1. **HW Design**
   - SW Interface Model
   - Generate SW Interface Model
   - SW Build

2. **IP Core Generation**
   - FPGA IP Core
   - AXI Lite Accessible registers
   - Algorithm from MATLAB and Simulink
   - External Ports

3. **Simulink Model**
   - Embedded System Project
   - AXI4-Lite Bus
   - FPGA Bitstream

4. **Algorithm from MATLAB and Simulink**
   - AXI Lite Accessible registers
   - FPGA IP Core
   - External Ports

5. **External Ports**
   - Processor
   - SW I/O Driver Blocks
   - SW Interface Model
Summary

- Model-Based Design for FPGA

- Generating HDL Code from MATLAB and Simulink
  - For prototyping and production
  - Optimizing code for efficiency

- Verifying HDL Designs with MATLAB and Simulink
  - Co-simulation with HDL simulators
  - FPGA-in-the-Loop verification

- Verifying HDL Designs outside MATLAB and Simulink
  - Generating code for integration with SystemC/TLM and SystemVerilog/DPI-C