MATLAB EXPO 2019

Adopting Model-Based Design for FPGA, ASIC, and SoC Development

Robert Anderson
Principal Application Engineer - MathWorks
Agenda

- Why Model-Based Design for FPGA, ASIC, or SoC?
- How to get started
  - General approach – collaborate to refine with implementation detail
  - Re-use work to help RTL verification
  - Hardware architecture
  - Fixed-point quantization
  - HDL code generation
  - Chip-level architecture
- Customer results
FPGA, ASIC, and SoC Development Projects

67% of ASIC/FPGA projects are behind schedule

Over 50% of project time is spent on verification

75% of ASIC projects require a silicon re-spin

84% of FPGA projects have non-trivial bugs escape into production

Statistics from 2018 Mentor Graphics / Wilson Research survey, averaged over FPGA/ASIC
Many Different Skill Sets Need to Collaborate

- Poor communication across teams
- Key decisions made in silos
- System-level issues found in late stages
- Hard to adapt to changing requirements

“Rapid innovation under a rapid timeline – that’s when this flow falls apart.”

Jamie Haas
Allegro Microsystems
Abstraction vs Design Space Exploration

Effort required to move across design space

Abstraction Level

- High Level Modeling
- Model Elaboration/ Fixed-Point Conversion
- RTL Design and HDL Verification
- Place and Route and Floor-planning
Cost of Finding a Bug vs Location in Design Cycle

Location in Design Cycle

- Requirements
- High Level Modeling/Verification
- Model Elaboration/Fixed-Point Conversion
- RTL Design and HDL Verification
- Place and Route/Floor-planning
- Integration and Validation Test
- Post – Production/Product Launch
SoC Collaboration with Model-Based Design

What am I making?

How am I making it?

Make it!

Research

Requirements

Design

System Architecture

Algorithms

Implementation Architectures

Implementation Knowledge

Generate Code

Export Models

Validation & Verification

Am I making the right thing?

Is it going to work?

Have I made it right?
Agenda

- Why Model-Based Design for FPGA, ASIC, or SoC?
- How to get started
  - General approach – collaborate to refine with implementation detail
  - Re-use work to help RTL verification
  - Hardware architecture
  - Fixed-point quantization
  - HDL code generation
  - Chip-level architecture
- Customer results
General Approach: Use the Strengths of MATLAB and Simulink

MATLAB

✓ Large data sets
✓ Explore mathematics
✓ Control logic
✓ Data visualization

Bit Accurate

Simulink

✓ Parallel architectures
✓ Timing
✓ Data type propagation
✓ Mixed-signal modeling

Cycle Accurate
Partition Hardware-Targeted Design, System Context, Testbench

Algorithm Stimulus

Create input stimulus

```matlab
function [ CorrFilter, RxSignal, RxFxPt ] = pulse_detector_stim 

% Create pulse to detect
rng('default');
PulseLen = 64;
theta = rand(PulseLen,1);
pulse = exp(1i*2*pi*theta);

% Insert pulse to Tx signal
rng('shuffle');
TxLen = 5000;
PulseLoc = randi(TxLen-PulseLen*2);

TxSignal = complex(ones(TxLen,1));
TxSignal(PulseLoc:PulseLoc+PulseLen-1) = pulse;

% Create Rx signal by adding noise
Noise = complex(randn(TxLen,1),randn(TxLen,1));
RxSignal = TxSignal + Noise;

% Scale Rx signal to +/- one
scale = max(abs(real(RxSignal)), abs(imag(RxSignal)));
RxSignal = RxSignal / scale;
end
```

MATLAB golden reference

```matlab
% Create matched filter coefficients
CorrFilter = conj(flip(pulse))/PulseLen;

% Correlate Rx signal against matched filter
FilterOut = filter(CorrFilter,1,RxSignal);

% Find peak magnitude & location
[peak, location] = max(abs(FilterOut));
```

Hardware Algorithm

Software Algorithm

Analysis

MATLAB EXPO 2019
Streaming Algorithms: MATLAB or Simulink...or Both

Hardware friendly implementation of peak finder
Instead of calculating the maximum value of the entire frame, we look for a local peak within a sliding window of the last 11 samples using the following criteria:

- The middle sample is the largest
- The middle sample is greater than a pre-defined threshold

```matlab
WindowLen = 11;
MidIdx = ceil(WindowLen/2);
threshold = 0.03;

% Compute magnitude squared to avoid sqrt operation
MagSqOut = abs(FilterOut).^2;

% Sliding window operation
for n = 1:length(FilterOut)-WindowLen
    % Compare each value in the window to the middle sample via a
    % DataBuffer = MagSqOut(n:n+WindowLen-1);
    MidSample = DataBuffer(MidIdx);
    CompareOut = DataBuffer - MidSample; % this is a vector
    % if all values in the result are negative and the middle sample
    % greater than a threshold, it is a local max
    if all(CompareOut < 0) && (MidSample > threshold)
        peak_idx = MidSample;
        location = n + (MidIdx-1);
    end
end
```

% Simulate model
`sim('pulse_detector_v1')`

% Correlation filter output
`FilterOutStart = squeeze(logout.get...
compareData(real(FilterOut),real(F)`

Stream input data using the "Signal From Workspace" block
Refine Algorithm and Verify Against Golden Reference

Algorithm
Stimulus

Reference
Algorithm
Verification
“Scoreboard”

Design Under Test

Streaming Algorithms
Streaming Hardware Architectures
Fixed-Point Hardware Architectures

MATLAB EXPO 2019
Generate SystemVerilog DPI Components for RTL Verification

- Reuse MATLAB/Simulink models in verification
  - Scoreboard, stimulus, or models external to the RTL
    - Generate from frame-based or streaming algorithm
    - Floating-point or fixed-point
    - Individual components or entire testbench
  - Runs natively in SystemVerilog simulator
  - Eliminate re-work and miscommunication
  - Save testbench development time
  - Easy to update when requirements change

MATLAB EXPO 2019
What if there’s a mismatch?

- Co-simulate with 3rd-party HDL simulator
  - Reuse MATLAB/Simulink test environment
  - Run HDL design in a supported simulator*
  - Generate co-simulation infrastructure and handshaking
  - Analyze both the design and test environment

MATLAB EXPO 2019

* Mentor Graphics® ModelSim® or Questa®
Cadence® Incisive® or Xcelium™
Collaborate to Add Hardware Architecture

Optimize architecture design for hardware goals

Specify HDL implementation options
Fixed-Point Streaming Algorithms: Manual Approach
Fixed-Point Streaming Algorithms: Automated Approach

- Simulate with representative data to collect required ranges
- Fixed-Point Designer proposes data types
- Choose to apply proposed types or set your own
- Simulate and compare results
Generating Native Floating Point Hardware

HDL Coder Native Floating Point
- Extensive math and trigonometric operator support
- Optimal implementations without sacrificing numerical accuracy
- Mix floating- and fixed-point operations
- Generate target-independent HDL

<table>
<thead>
<tr>
<th>Floating point target</th>
<th>Fixed point</th>
<th>Floating point</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUTs</td>
<td>10k</td>
<td>25k</td>
</tr>
<tr>
<td>DSP slices</td>
<td>50</td>
<td>100</td>
</tr>
<tr>
<td>Development time</td>
<td>~1 week</td>
<td>~1 day</td>
</tr>
</tbody>
</table>

~2x more resources
~5x less development effort
Automatically Generate Production RTL

- Choose from over 250 supported blocks
  - Including MATLAB functions and Stateflow charts
- Quickly explore implementation options
  - Micro-architectures
  - Pipelining
  - Resource sharing
  - Fixed-point or native floating point
- Generate readable, traceable Verilog/VHDL
  - Optionally generate AXI interfaces with IP core
- Quickly adapt to changes and re-generate
- Production-proven across a variety of applications and FPGA, ASIC, and SoC targets
Agenda

- Why Model-Based Design for FPGA, ASIC, or SoC?
- How to get started
  - General approach – collaborate to refine with implementation detail
  - Re-use work to help RTL verification
  - Hardware architecture
  - Fixed-point quantization
  - HDL code generation
  - Chip-level architecture
- Customer results
Results at Allegro Microsystems

The Enlightenment: Model Based Design

- Research
- Requirements

Algorithm and Architecture Design
- Continuous Time
- Discrete Time

Implementation
- Custom Analog Transistor Level
- Verilog and Software

Physical Implementation and Integration

- Prototyping, Testing, Verification
- Architecture and Algorithm Design Evolve into Executable Specifications
- Front load testing and verification
- Development is “parallelized”
- Continuous Equivalency Testing is utilized
- And of course auto-generated production code

Link to MATLAB Expo video
Getting Started Collaborating with Model-Based Design

- **RESEARCH**
  - System Architecture
  - Algorithms
  - Implementation Architectures

- **REQUIREMENTS**

- **DESIGN**

- **Implementation Knowledge**
  - Embedded Software
  - Digital Hardware
  - Analog Hardware
  - System Integration

- **Generate Code**

- **Export Models**

- **Validation & Verification**

- • Refine algorithm toward implementation
- • Verify refinements versus previous versions
- • Generate verification models
- • Add hardware implementation detail and generate optimized RTL
- • Simulate System-on-Chip architecture

- ➢ Eliminate communication gaps
- ➢ Key decisions made via cross-skill collaboration
- ➢ Identify and address system-level issues before implementing subsystems
- ➢ Adapt to changing requirements with agility

MATLAB EXPO 2019
Learn More

- **Next steps to get started:**
  - Verification: [Improve RTL Verification by Connecting to MATLAB webinar](https://www.mathworks.com/solutions/hdl-verification.html)
  - Fixed-point quantization: [Fixed-Point Made Easy webinar](https://www.mathworks.com/solutions/fixed-point-quantization.html)
  - Incremental refinement, HDL code generation: [HDL self-guided tutorial](https://www.mathworks.com/solutions/hdl-code-generation.html)

- **Technology showcase here at MATLAB EXPO**

- **MathWorks Advisory Board (MAB)**

- **Pilots and Consulting services to help you get on-board**

- **Contact your local sales representative for hands-on workshops**