MATLAB EXPO 2019
Adopting Model-Based Design for FPGA, ASIC, and SoC Development
Fahd Morchid
Agenda

▪ Why Model-Based Design for FPGA, ASIC, or SoC?
▪ Case Study – Pulse Detector
▪ HW/SW Co-Design
▪ Customer results

Just an example, the workflow is the same for...
Agenda

- Why Model-Based Design for FPGA, ASIC, or SoC?
  - Case Study – Pulse Detector
  - HW/SW Co-Design
  - 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

SPECIFICATIONS
- Algorithms
- System Architecture

RESEARCH

SPECIFICATIONS
- Embedded Software
- Digital Hardware
- Analog Hardware

SYSTEM INTEGRATION

SPECIFICATIONS
- Verification

- 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
SoC Collaboration with Model-Based Design

WHAT am I making?

HOW am I making it?

MAKE IT!

Am I making the right thing?

Is it going to work?

Have I made it right?

Design Elaboration

SIMULATION

DESIGN

RESEARCH

REQUIREMENTS

Implementation Knowledge

Generate Code

Export Models

Embedded Software

Digital Hardware

Analog Hardware

System Integration

Validation & Verification

Analog Hardware

Hardware

Implementation Architectures

Implementation Knowledge

Generate Code

Export Models

MathWorks

MATLAB EXPO 2019
General Approach: Use the Strengths of MATLAB and Simulink

MATLAB

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

Simulink

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

- Why Model-Based Design for FPGA, ASIC, or SoC?
- Case Study – Pulse Detector
- HW/SW Co-Design
- Customer results
Case Study | Pulse Detector

1. Example Overview
2. Reference Pulse Detector
3. Pulse Detector Design
4. Prepare for Hardware Implementation
5. Fixed-point Conversion
6. HDL code generation, synthesis and verification
Case Study | Pulse Detector

1. Example Overview
2. Reference Pulse Detector
3. Pulse Detector Design
4. Prepare for Hardware Implementation
5. Fixed-point Conversion
6. HDL code generation, synthesis and verification
**Pulse Detector | Overview**

**Send**

**Receive**

**Detect**

**Reference Design (MATLAB)**

**Detector Design (Simulink)**

**Hardware Implementation (HDL)**

MATLAB EXPO 2019
Case Study | *Pulse Detector*

1. Example Overview
2. Reference Pulse Detector
3. Pulse Detector Design
4. Prepare for Hardware Implementation
5. Fixed-point Conversion
6. HDL code generation, synthesis and verification
Pulse Detector | *Reference Design (MATLAB)*

**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( expj(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)));
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));
```

**Analysis**

![Graph showing Tx and Rx signals with peak found at 2749 with a value of 1.915e-01]
Pulse Detector | Reference Design (MATLAB)

Algorithm
Stimulus

Design Under Test
Streaming Algorithms
Streaming Hardware Architectures
Fixed-Point Hardware Architectures

Reference Algorithm
Verification “Scoreboard”

Streaming Algorithms
Streaming Hardware Architectures
Fixed-Point Hardware Architectures

MATLAB EXPO 2019
Pulse Detector | Reference Design (MATLAB)

- Reuse MATLAB/Simulink models in verification
  - Scoreboard, stimulus, or models external to the RTL
  - Runs natively in SystemVerilog simulator
  - Eliminate re-work and miscommunication
  - Save testbench development time
  - Easy to update when requirements change
Pulse Detector | Reference Design (MATLAB)

- 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

* Mentor Graphics® ModelSim® or Questa® Cadence® Incisive® or Xcelium™
Case Study | *Pulse Detector*

1. Example Overview
2. Reference Pulse Detector
3. Pulse Detector Design
4. Prepare for Hardware Implementation
5. Fixed-point Conversion
6. HDL code generation, synthesis and verification
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 if
    DataBuff = MagSqOut(n:n+WindowLen-1);
    MidSample = DataBuff(MidIdx);
    CompareOut = DataBuff - 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_2 = MidSample;
        location_2 = n + (MidIdx-1);
    end
end
```

MATLAB EXPO 2019
Case Study | *Pulse Detector*

1. Example Overview
2. Reference Pulse Detector
3. Pulse Detector Design
4. Prepare for Hardware Implementation
5. Fixed-point Conversion
6. HDL code generation, synthesis and verification
In this step, we:

- prepare the model for HDL code generation
- pipeline the data path using various techniques
- add data valid control signal
- verify against MATLAB golden reference
Case Study | *Pulse Detector*

1. Example Overview
2. Reference Pulse Detector
3. Pulse Detector Design
4. Prepare for Hardware Implementation
5. Fixed-point Conversion
6. HDL code generation, synthesis and verification
Pulse Detector | Fixed-Point Conversion

In this step, we:

- convert the model to fixed-point
- compare the Simulink fixed-point model to the MATLAB golden reference
Some words about Fixed-Point conversion…
Fixed-Point Conversion | Automated Approach

Fixed-Point Designer™ provides data types and algorithms to optimize performance.

- **Simulate with representative data**
- **Fixed-Point Designer proposes data types**
- **Choose to apply proposed types or set your own**
- **Simulate and compare results**

Fixed-Point Designer helps you convert double-precision to single-precision or fixed-point. You can create and optimize numerical accuracy requirements and target hardware to determine range requirements of your design. It guides you through the data conversion process and enables you to compare results with floating-point baselines.

Fixed-Point Designer supports C, HDL, and PLC code generation.
Fixed-Point Conversion | Native Floating-Point

**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:**
<table>
<thead>
<tr>
<th></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
Case Study | *Pulse Detector*

1. Example Overview
2. Reference Pulse Detector
3. Pulse Detector Design
4. Prepare for Hardware Implementation
5. Fixed-point Conversion
6. HDL code generation, synthesis and verification
Pulse Detector | HDL Code Generation and Verification

In this step, we:

- generate HDL code and reports
- synthesize the design using Xilinx Vivado
- verify the design
3.1.5. Set Testbench Options

Test Bench Generation Options
- HDL test bench
- Cosimulation model
- SystemVerilog DPI test bench
Simulation tool: Mentor Graphics ModelSim

HDL code coverage

Configuration
- Test bench name postfix: _tb
- Force clock
  - Clock high time (ns): 5
  - Clock low time (ns): 5
- Hold time (ns): 2
- Setup time (ns): 8
- Force clock enable
  - Clock enable delay (in clock cycles): 1
- Force reset
  - Reset length (in clock cycles): 2
- Hold input data between samples
- Initialize test bench inputs
- Multi-file test bench

Test bench data file name postfix: _data
Test bench reference postfix: _ref
Is there more?
Case Study | Pulse Detector

1. Example Overview
2. Reference Pulse Detector
3. Pulse Detector Design
4. Prepare for Hardware Implementation
5. Fixed-point Conversion
6. HDL code generation, synthesis and verification
Case Study | Workflow Summary

Golden Reference

Hardware Architecture

Fixed-point Implementation

HDL Code Generation and Optimization

HDL Verification and Targeting

MATLAB

Simulink

Fixed Point Designer

HDL Coder

Integrated Verification

MATLAB EXPO 2019
A few more words about code generation ...
Automatically Generate Production RTL

- Choose from over 300 supported blocks
  - Including MATLAB functions and Stateflow charts
- Quickly explore implementation options
- Generate readable, traceable Verilog/VHDL
  - Optionally generate AXI interfaces with IP core
- Production-proven across a variety of applications and FPGA, ASIC, and SoC targets
Agenda

- Why Model-Based Design for FPGA, ASIC, or SoC?
- Case Study – Pulse Detector
- HW/SW Co-Design
- Customer results
HW/SW Design
Model Based Design Workflow for SoC

*Deploy to Hardware with Coders and HW Support Package*

![Diagram of Model Based Design Workflow for SoC](image)

- **Algorithmic Model**
- **Algorithmic Code**
- **HW Support Package (Reference Design)**
- **Hardware Platform**
Actual Data Exchange Between FPGA and Processor

- FPGA
  - Ts (ns)
  - Burst
  - Sample
  - FIFO size

- FIFO

- Memory
  - Buffer1
  - Buffer2
  - Buffer3
  - Buffer4
  - Tb (us)
  - Tf (ms)

- ARM
  - Memory Reader
  - Alg2

- Other Memory Readers and Writers

- Other Threads and Processes

Questions:
- FIFO size?
- Data rate?
- Burst size?
- Number of buffers?
- How to synchronize incoming data with task execution?
- Data rate?
- FIFO size?
- Burst size?
- Number of buffers?
SoC Blockset / Model and Simulate SoC Architecture
• Simulate algorithms as well as hardware/software architecture
  ➢ Memory
  ➢ Internal/external connectivity
  ➢ I/O
  ➢ Task scheduling

• Deploy on support hardware

• Profile performance using external mode
SoC Blockset / Example

Streaming Data from Hardware to Software

Latency Requirements

<table>
<thead>
<tr>
<th>#</th>
<th>Frame Size</th>
<th>Frame period (ms)</th>
<th>Number of buffers</th>
<th>Mean Task Duration (ms)</th>
<th>Avg Samples dropped per 10000</th>
<th>Meets or Violates requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>0.05</td>
<td>1999</td>
<td>0.059</td>
<td></td>
<td>Violates throughput</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>1</td>
<td>99</td>
<td>1.06</td>
<td></td>
<td>Violates throughput</td>
</tr>
<tr>
<td>3</td>
<td>8000</td>
<td>8</td>
<td>1</td>
<td>7.858</td>
<td>172.6</td>
<td>Violates drop samples</td>
</tr>
<tr>
<td>4</td>
<td>1000</td>
<td>10</td>
<td>9</td>
<td>9.61</td>
<td>0</td>
<td>Meets all requirements</td>
</tr>
<tr>
<td>5</td>
<td>1600</td>
<td>16</td>
<td>5</td>
<td>15.3</td>
<td>1</td>
<td>Meets all requirements</td>
</tr>
<tr>
<td>6</td>
<td>2000</td>
<td>20</td>
<td>4</td>
<td>19.067</td>
<td>2.25</td>
<td>Violates drop samples</td>
</tr>
<tr>
<td>7</td>
<td>2400</td>
<td>24</td>
<td>3</td>
<td>22.812</td>
<td>3.9</td>
<td>Violates drop samples</td>
</tr>
<tr>
<td>8</td>
<td>8000</td>
<td>80</td>
<td>&lt;1</td>
<td>76.56</td>
<td></td>
<td>Violates min buffers req</td>
</tr>
<tr>
<td>9</td>
<td>18000</td>
<td>180</td>
<td>&lt;1</td>
<td>175.23</td>
<td></td>
<td>Violates min buffers req</td>
</tr>
<tr>
<td>10</td>
<td>30000</td>
<td>300</td>
<td>&lt;1</td>
<td>289.52</td>
<td></td>
<td>Violates min buffers req</td>
</tr>
</tbody>
</table>
Simulate SoC Architectures

Develop and combine software algorithms, hardware logic, memory systems, and I/O devices into your SoC application. Evaluate architecture alternatives before deploying to hardware.

Analyze System Performance

Evaluate memory performance and task execution through simulation and perform on-device profiling.

Deploy to SoC and FPGA Devices

Generate reference designs and RTL code for programmable logic. Generate C/C++ code for processor tasks.
Agenda

- Why Model-Based Design for FPGA, ASIC, or SoC?
- Case Study – Pulse Detector
- HW/SW Co-Design
- 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
Getting Started Collaborating with Model-Based Design

- **Requirements**
  - System Architecture
  - Algorithms
  - Implementation Architectures

- **Research**
  - Analog Hardware
  - Embedded Software
  - Digital Hardware
  - System Integration

- **Design**
  - Implementation Knowledge
  - Generate Code

- **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
Learn More

- Visit FPGA & SoC booth!

- Next steps to get started with:
  - Verification: Improve RTL Verification by Connecting to MATLAB webinar
  - Fixed-point quantization: Fixed-Point Made Easy webinar
  - Incremental refinement, HDL code generation: HDL self-guided tutorial
  - SoC Blockset: Getting Started with SoC Blockset