Making the Most of your MATLAB Models to Improve Verification

Verification Futures 2016

Graham Reith
Industry Manager: Communications, Electronics & Semiconductors
Graham.Reith@mathworks.co.uk
Agenda

- Motivation: Role of MATLAB Models in Semiconductor Design
  - Desire to ‘shift-left’ verification
  - Faster verification testbench development, directly linked to design reference

- Technical Topics:
  - Building better models in MATLAB and Simulink
    - Fixed-point, architectural
  - Improving verification at the design stage
    - Model coverage, test case generation
  - Re-using models in verification environments
    - Deployment to SystemVerilog/UVM with DPI-C export
    - Co-simulation and FPGA-in-the-Loop

- Next Steps
Shifting-Left

Where Errors are Introduced… and Detected

Shift-left Verification

Increased Modelling & Simulation

Traditional Verification

Clive Maxfield and Kuhoo Goyal
”EDA: Where Electronics Begins”
Model-Based Design

- RESEARCH
- REQUIREMENTS
- ALGORITHM DESIGN
  - Environment Models
  - Digital Models
  - Analog Models
  - RF Models
  - Timing and Control Logic
  - Algorithms
- ALGORITHM IMPLEMENTATION
  - C/C++
  - HDL
  - MCU
  - DSP
  - FPGA
  - ASIC
  - RF & Analog
  - Transistor
- INTEGRATION
Model-Based Design

WHAT am I making?

Am I making the right thing?

HOW am I making it?

Is it going to work?

MAKE IT!

Have I made it right?

Is it going to work?

Have I made it right?

WHAT am I making?

Am I making the right thing?

HOW am I making it?

Is it going to work?

MAKE IT!

Have I made it right?
Model-Based Design

Modelling Algorithms
Ideal, Floating Point

Detailed design:
Fixed-Point
Architectural
System context

Implementation:
C and RTL
generation

Validation of
Requirements

Verification
against
Requirements

Verification
against
Detailed Design
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
Survey of Existing Customers: Model-Based Design Benefits

- **Requirements Defect Reduction**
  - Medium - Very High: 70%
  - Low - Very Low: 30%

- **Design Defect Reduction**
  - Medium - Very High: 91%
  - Low - Very Low: 9%

- **Coding Defect Reduction**
  - Medium - Very High: 93%
  - Low - Very Low: 7%
Survey of Existing Customers: Split by Process Maturity

<table>
<thead>
<tr>
<th></th>
<th>Initial</th>
<th>Optimizing</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQUIREMENTS DEFECT REDUCTION</td>
<td>40% (Medium) - 60% (Low)</td>
<td>75% (Medium) - 25% (Low)</td>
</tr>
<tr>
<td>DESIGN DEFECT REDUCTION</td>
<td>55% (Medium) - 45% (Low)</td>
<td>92% (Medium) - 8% (Low)</td>
</tr>
<tr>
<td>CODING DEFECT REDUCTION</td>
<td>57% (Medium) - 43% (Low)</td>
<td>100% (Medium) - 0% (Low)</td>
</tr>
</tbody>
</table>
How to use MATLAB as a Golden Reference

- Different model fidelities:
  - Floating point
  - Bit accurate, no latency
  - Bit accurate, with latency
  - Bit accurate, latency and controls

- Tools to improve quality of test framework

- Export to C/C++, DPI-C or SystemC

- Co-simulation with HDL, FPGA, UVM, etc.
Verification at the Model Level

- **Simulink Verification & Validation**
  - Coverage results for tests that have been simulated
  - Interfaces to Requirements Management systems

- **Simulink Design Verifier**
  - Generation of tests to deliver coverage
  - Identification of unreachable code
  - Formal proof methods against objectives
Example Summary
Example Summary - Coverage

Summary

- Model Hierarchy/Complexity
- Test 1
- C1
- MCDC
- Execution

Details

1. Model "Determine_Cell_ID_and_Offset"

Child Systems:

- Unit Delay Enabled
- Unit Delay Enabled1
- Unit Delay Enabled2
- Unit Delay Enabled3

Metric | Coverage (this object) | Coverage (inc. descendants)
---|---|---
Cyclomatic Complexity | 1 | 14
Condition (C1) | NA | 99% (18/20) condition outcomes
Decision (D1) | NA | 83% (20/24) decision outcomes
MCDC (C1) | NA | 40% (2/5) conditions reversed the outcome
Execution | NA | 100% (5/5) objective outcomes

Coverage: lteofdm_modDetect

- Logic block "Logical Operator"

Conditions port2 and port3 were never true. Conditions input port 2 and input port 3 have not demonstrated MCDC. Full Execution coverage.
Example Summary – Test Case Generation
Verification Environment Creation

- System and algorithm verified in MATLAB or Simulink
  - Algorithmic models
  - Full system environment
  - Realistic stimulus

- Specification written, passed to verification

- Verification interprets spec to recreate:
  - Checker model
  - Models external to DUT
  - Stimulus

Why recreate when we can reuse?
HDL Verifier
Automatically generate SystemVerilog DPI-C models

- Reuse MATLAB/Simulink models in verification
  - Available immediately
  - Already verified
  - Easy to update

- Everything is automated
  - Generates code and SystemVerilog interface
  - Generates and runs makefile to build shared library

- Anywhere C code can be generated
  - Digital and analog
  - Broad block and language support
Demo Overview

Checker:
- Calculate floating point FFT
- Calculate difference vs. DUT
  \[ \text{normalized rms error} = \frac{\text{rms}(error)}{\text{rms}(result)} \]
- Compare vs. theoretical upper bound

Waveform:
What would it take to write these in SystemVerilog?
Demo Summary

```matlab
function result = fft_checker(fftin_re, fftin_im, fftout_re, fftout_im)
    %#codegen
    % First convert FFT input to double
    fftin_c = complex(double(fftin_re), double(fftin_im));
    % Calculate FFT
    fft_exp = fft(fftin_c)/64;
    % Calculate error
    fftoutc = complex(double(fftout_re), double(fftout_im));
    error = fft_exp - fftoutc;
    % Calculate normalized rms of error
    result = real((rms(error)/rms(fftoutc)));
end
```

Build the checker, specifying input data types

dpigen -args \{int16(ones(1,64)),int16(ones(1,64)),int16(ones(1,64)),int16(ones(1,64))\} fft_checker -launchreport

### Generating DPI-C Wrapper **fft_checker_dpi.c**
### Generating DPI-C Wrapper header file **fft_checker_dpi.h**
### Generating SystemVerilog module **fft_checker_dpi.sv**
### Generating makefiles for: fft_checker_dpi
### Compiling the DPI Component
Code generation successful: View report
Demo Summary – Generated Files

```
/* File: fft_checker.c */
* MATLAB Coder version
* C/C++ source code generated
*/

#include "rt_nonfinite.h"
#include "fft_checker.h"

/* Function Declarations */
static void fft(const real_T x);
static real_T rms(const real_T *x);
static double rt_hypotd_end;

/* Function Definitions */

static void fft(const real_T *x)
{
    real_T y[64];
    int i;
    int j;
    int k;
    int l;

    for (i = 0; i < 64; i++)
    {
        y[i] = 0;
    }

    for (i = 0; i < 64; i++)
    {
        for (j = 0; j < 64; j++)
        {
            y[j] += x[i] * x[j];
        }
    }

    for (i = 0; i < 64; i++)
    {
        y[i] /= 64;
    }

    /* Further processing...
    */
```

Generated by MATLAB 8.6 and HDL Verifier 4.7
// Declare imported C functions generated by MATLAB and Simlink

import "DPI-C" function cchandle DPI_fft_checker_initialize();
    cchandle existchandle;
import "DPI-C" function void DPI_fft_checker();
    input cchandle objchandle,
        input shortint ffitin_re [64],
        input shortint ffitin_im [64],
        input shortint fftout_re [64],
        input shortint fftout_im [64],
        output real result);

ufw_analysis_imp_dolp (ffitin)
ufw_analysis_imp_dolp (ffftot)
class fft_scoreboard extends uvm_scoreboard;
    `uvm_component_utils (fft_scoreboard)
uvm_analysis_imp_fftin #(fft_transaction, fft_scoreboard) pfft_transaction qmem fftin[$];
chandle dplc_h;
function new(string name, uvm_component parent);
    super.new(name, parent);
    port ffttin = new("port_fft tin", this);

endfunction: write_fft tin

function void write_fft tinout (fft_transaction t);
    fft_transaction t;
    real rms;
    real threshold = 0.002;
    string msg;
    
    // Pop out an input transaction
    ffitin = queue fftin.pop_front();
    
    // Calculate normalized rms of fixed point design
    DPI_fft_checker [dplc_h, ffitin.data_re, ffitin.data_im, t.data_re, t.data_im, rms];

$format (msg, "Normalized rms value of fixed-point FFT is \%", rms);
    'uvm_info ("SCORE", msg, UVM_NONE);
    if (rms > threshold) begin
        uvm_error ("FAIL", "The normalized rms value is higher than specified threshold.");
    end
    else begin
        'uvm_info ("PASS", "The normalized rms value is lower than threshold. Test passed.", UVM_NONE);
    end
    if (queue fftin.size() == 0) begin
        uvm_test_done.drop_objective(this);
    end
endfunction: write_fft tinout
Demo Results

SystemVerilog UVM environment

**Checker:**
- 5 lines of MATLAB
- Generated with a single command
- Easily adjusted

**Waveform:**
- 10-line MATLAB function
- Generated with a single command
- Easily adjusted or replicated

How much work does this save?
HDL Verifier DPI-C Component Generation
Reuse MATLAB and Simulink models for verification

- Models available earlier
- Accurately capture algorithm behavior
- Establish connection to original source
- Easy to update
- Applicable to broad class of models
Reusing MATLAB as a Verification Environment

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
Reusing MATLAB as a Verification Environment
FPGA-in-the-Loop Verification of HDL Source Code

- FIL simulation with FPGA development board
  - Reuse of existing testbench in MATLAB/Simulink
  - HDL code execution on FPGA
  - Flexible HDL sources
    - Handwritten or generated code
  - Automated generation of co-simulation infrastructure
    - Encapsulation of algorithm within GBit Ethernet MAC, or via JTAG
  - Automatic handshaking
Summary

- **Motivation: Role of MATLAB Models in Semiconductor Design**
  - Desire to ‘shift-left’ verification
  - Faster verification testbench development, directly linked to design reference

- **Technical Topics:**
  - Building better models in MATLAB and Simulink
    - Fixed-point, architectural
  - Improving verification at the design stage
    - Model coverage, test case generation
  - Re-using models in verification environments
    - Deployment to SystemVerilog/UVM with DPI-C export
    - Co-simulation and FPGA-in-the-Loop

- **Next Steps**
Next Steps

- **Verification Focus:**
  - Establish whether you can re-use existing MATLAB models in your verification testbench
    - Save effort
    - Close the loop on the development process

- **Overall Design Flow Focus:**
  - Consider how you can achieve a shift-left in verification
  - How can modelling fidelity be improved ahead of implementation
    - Improved Models
    - Improved Testbenches

- For more information, references, and discussion on this topic
  - Contact [Graham.Reith@mathworks.co.uk](mailto:Graham.Reith@mathworks.co.uk)