Co-Simulating Matlab/Simulink Models in a UVM Environment

Neal Okumura – Raytheon Company
Paul Yue – Raytheon Company
Glenn Richards – Mentor Graphics Corporation
Agenda

- Introduction
- Environment
- How
  - Matlab Engine
  - SystemVerilog Direct Programming Interface (DPI)
  - Simulink
- Coverage/Assertion Based Verification
- Conclusion
- Benefits
Introduction

• System design and RTL design processes are largely treated as 2 independent activities
  – Decreased productivity
    • Forcing/enforcing serialized process between design activities
    • Limited reuse opportunities
  – Prone to error: hand-off is a manual process
  – Need hard-to-find commonality between the 2 disciplines to ensure design success
Introduction

• Combining the simulations together in one environment
  – Improves the process efficiency
  • Increase re-use opportunities
  • Increase interactions between the 2 groups
  – Reduces the chance of errors, automated hand-off
  – Reduce the dependency on the system architect for test vector generation
Environment

• Why co-simulate with Simulink
  – Matlab is a common language understood by System Engineering
  – Temporal constructs easily correlate to RTL simulation
    • Timebased modeling
  – Well established modeling environment
    • Long support life and is able to easily resurrect
    • Programs must last 30 years at Raytheon as required by our customers
  – Graphical environment, easier to visualize system
Overview

• Matlab has a library of C functions that can be used to interface to it
• SystemVerilog can import C functions through the Direct Programming Interface (DPI)
• When Matlab functions are imported to SystemVerilog, the SystemVerilog environment can invoke a Matlab environment
  – Control Simulink from Matlab
  – Create UVM environment in SystemVerilog
Matlab Engine

• Matlab has a built in library of C functions called the Matlab engine
• Functions in the library include:
  – Open and close the Matlab software
  – Put and get variables into the Matlab workspace
  – Send strings to Matlab console
• Contained in the file “engine.h” included with Matlab
C Source File

- Create a C source file containing user functions that call the Matlab engine functions in the engine.h file
- Functions can also format the data and perform data checking like NULL pointers
- User example shown
  - Matlab also has an included example in the file engdemo.c

```c
#include "engine.h"
void MYengOpen () {
  engOpen();
}
```
SystemVerilog DPI

- SystemVerilog Direct Programming Interface (DPI) allows SystemVerilog to interface to external languages like C
- Use DPI to import the user functions containing the Matlab engine
- Allows SystemVerilog to call Matlab engine functions

```import "DPI-C" function void MYengOpen();```
Engine Object

• Put all Matlab Engine functions in an object
  – Array handles disappear after a Matlab function completes
  – Keeping the functions within an object keeps the handle alive as long as the Matlab engine is still open
  – Helps with code readability
  – Easier to hand off to another user
Engine Object

• Create an engine class and use the imported Matlab engine functions within the class
• Engine object will then execute the Matlab functions

class engine_example;
    function void void engOpen();
        MYengOpen();
    endfunction
endclass : engine example
Package

• All Matlab engine and DPI functions should be contained together in a package
  – Matlab engine functions imported by DPI
  – Engine class
  – Other supporting data types
Using the Engine Object

• Create the engine object in the main testbench module
• Run the Matlab functions from the created object
• Matlab can now be executed from SystemVerilog

```systemverilog
import example_engine_pkg::*;

engine_example eng;

initial begin
  eng = new();
  eng.engOpen();
end
```
Simulink

• Matlab engine cannot directly command Simulink, but by using commands in the Matlab console Simulink can be commanded

• The Matlab engine library contains a function called engEvalString() that sends a string to the Matlab console
Simulink

SystemVerilog source:

```systemverilog
eng = new();
eng.engOpen();
eng.engEvalString("open_system('simulink_model');");
eng.engEvalString("sim('simulink_model',[0,1]);");
```

Matlab engine sends strings to the Matlab console:

```matlab
>> open_system('simulink_model');
>> sim('simulink_model',[0,1]);
```
Simulink Data

• Matlab engine cannot directly access Simulink data
• However, the Matlab engine can access data in the Matlab workspace
• Simulink data needs to be placed in Matlab workspace
• Modify Simulink model
Simulink Monitors

• Monitors need to be added to the Simulink model to the points that will be compared to the DUT
• These monitors capture the simulation data at a given sample rate and saves it to an array
Simulink Data Transfer to UVM

- Simulink data captured by monitors will be placed in the Matlab workspace upon simulation pause or completion.
- Matlab engine library contains a function called `engGetVariable()` that gets a variable from the Matlab workspace.
- This data can be placed in a variable in SystemVerilog.
UVM Environment

• A single iteration of Simulink can be run
  – SystemVerilog opens Matlab as a subprocess, and
    runs the Simulink model using the Matlab console
  – Once completed the Simulink monitors transfer the
data to the Matlab workspace
  – From SystemVerilog, get the data from the Matlab
    workspace
  – Using the data from Simulink run the UVM test,
    compare, and scoreboard the DUT
Simulation begins

Matlab Engine opens, Simulink opens, UVM pauses

Get data from Matlab, simulation resumes

Simulation ends

Time

Simulation begins

Simulation ends

Close Matlab Engine

UVM

Matlab

UVM

Matlab Engine

Simulink

UVM
Stimulus

• Stimulus Generation
  – Directed tests (conventional)
    • Generated from Simulink to UVM
    • Reuse existing Simulink stimulus generation
    • Less difficult requiring monitor functions only
  – Random/Constrained Random testing (unique)
    • Generated from UVM to Simulink
    • More difficult requiring input drivers to Simulink model
    • Creates new and unique test patterns
Random/Constrained Random Variables

• Generate random/constrained random variable in UVM
• Use Matlab engine function engPutVariable() to put the data in the Matlab workspace
• Create a driver in Simulink to take data from workspace and drive to component
Random/Constrained Random Variables

- Use Simulink driver to either
  - Replace existing stimulus in Simulink
  - Augment stimulus generated
- Allows for more coverage
Coverage Based Verification

- Environment compares the behavior of the RTL code to the Simulink model.
- Coverage goals can be applied to evaluate the amount of behavior compared.
- Use coverage goals to rank coverage in test cases.
- Use ranked cases to apply to verification plan.
Coverage Goal Completion

- To run a test case to coverage goal completion, the UVM environment runs indefinitely until all coverage goals have been met
- The Simulink model will need to run for multiple iterations
- Allows for churn type testing
Multiple Iterations of Simulink

• Simulink simulation needs to pause to return back to UVM environment
  – Create a timing counter in Simulink
    • Use a predetermined time frame
    • Will use same time frame in UVM
    • The 2 simulations will each step forward by this time step
    • Time frame dependent on amount of data being collected
    • 50K vectors
Multiple Iterations of Simulink

- Once counter reaches the predetermined time frame, trigger an assertion to pause the simulation
- The process will return back to the SystemVerilog environment
Multiple Iterations of Simulink

• Once Simulink is paused, use Matlab engine to get the data from the Matlab workspace
• Run UVM test to the same time frame with imported data
• Use Matlab console command to resume Simulink simulation
• Repeat and continue until coverage goals are completed

>> set_param('simulink_model', 'SimulationCommand', 'continue');
Multiple Iterations of Simulink

- Simulation begins
- Matlab Engine opens, Simulink opens, UVM pauses
- Get data from Matlab, After run time resume Simulink
- Simulation pauses
- Simulation resumes
- UVM simulation resumes
- Time

...
Assertion Based Verification

• Simulink model is used as a golden model to compare the DUT against
  – Coverage based environment tests that the DUT behaves like the golden model
• Assertions can also be created in the UVM environment to further verify the design
  – Assertion based environment will test the DUT against the requirements
• Take advantage of automated requirement traceability
  – Functional coverage
Conclusion

• Improved productivity
  – Model reuse with limited modification (monitors/drivers)
  – Parallel development realized, Performance measurement captured and real time validate using other MatLab toolkits.
• Efficient hand off (no hand off) and automated
• Simulation testcases are run solely by verification engineer and reviewed by system engineering.
Additional Benefits Realized

- Direct automated comparison between Simulink model and DUT
- Constrained random variables in UVM can be used for coverage completeness while still using Simulink as a golden model for comparison
  - Provided more complete coverage for verification
  - Churn testing with constrained random vectors
- Automated requirement verification traceability
Questions?
Enhancement - Parallel Process Simulation

- UVM starts Matlab Engine
- Simulink runs for a period of time and pauses
- When UVM resumes, fork the process to continue running UVM with the last Simulink data and continue running Simulink to get the next set of data
  - Instead of Simulink waiting while UVM is running with resulting data it can run and generate the next set of data
Enhancement - Parallel Process Simulation

UVM simulation starts

UVM environment starts the Matlab Engine and pauses

Matlab runs, starts Simulink simulation

Simulink simulation pauses

Simulink simulation resumes

Simulink simulation pauses

UVM simulation resumes

UVM simulation pauses

Matlab fork

Time
Enhancement - Emulation

- UVM lends itself well to emulation
- DUT can be placed in an emulator while the UVM environment and Matlab/Simulink run in software
- Accelerates the environment
Enhancement - Emulation

SystemVerilog

UVM

DPI

C

Matlab Engine

Matlab

Simulink

Emulator

DUT