Using MATLAB and Simulink in a SystemC Verification Environment

Jean-François Boland\textsuperscript{1}
Mathieu Hemon\textsuperscript{2}, Claude Thibeault\textsuperscript{2}, Zeljko Zilic\textsuperscript{1}

\textsuperscript{1}McGill University
Department of Electrical and Computer Engineering
Montréal, Québec, Canada

\textsuperscript{2}École de Technologie Supérieure
Département de génie Électrique
Montréal, Québec, Canada

September 29\textsuperscript{th}, 2004
Outline

- Goals
- Context of work
- Using MATLAB and Simulink for verification
- Related work
- Interfacing SystemC and MATLAB
  - MATLAB Engine
  - Simulink S-Functions
  - Simulators synchronization
  - Data types
- Multi Equalizer examples
Goals

- Functional verification takes approximately 70% of the total effort on a project
  - Need new techniques to cut verification time
- Large number of DSP designs
  - MATLAB and Simulink for algorithmic modeling
- Need to efficiently model complex environment during the verification phase
- Add robustness to stimulus generator and response checking
- Early testbench development and reuse
Context of Work

- Project MAME: Methodology and Architecture for Multi Equalization
- New design flow:
  - Design and Verify
  - Software Defined Radio
  - Multi equalizer architecture
- SystemC Verification Environment (with SCV)

MAME’s Design Flow
Functional Verification Environment

- Key contributions
  - Verification methodology
  - Interface between SystemC and Simulink
Why MATLAB and Simulink?

For Data Generation:
- **Data Generator**
  - MATLAB
  - Simulink
- **Verification Environment**
- **DUV**

For Data Analysis:
- **Data Analysis**
  - MATLAB
  - Simulink
- **Verification Environment**
- **DUV**

For “golden” Reference Model:
- **Golden Reference Model**
  - MATLAB
  - Simulink
- **Testbench Reuse**

2nd NASCUG
Related Work

- **SystemC calls MATLAB demo**
  - Source: MATLAB Central File Exchange, March 2003
  - Uses a SystemC module as a wrapper to call MATLAB via the Engine
  - No Simulink support

- **Modeling Cycle-Accurate Hardware with MATLAB and Simulink using SystemC**
  - Source: 6th European SystemC Users Group Meeting (ESCUG), October 2002
  - SystemC model included in Simulink as an S-Function
  - Simulink centric approach
  - Requires extensions to the SystemC kernel
How to connect SystemC and MATLAB/Simulink?

- SystemC communicates with MATLAB through MATLAB engine and the workspace
- Simulink communicates with MATLAB through S-Functions and the workspace
1// MATLAB/Simulink Interface
2
3#include "systemc.h"
4#include <engine.h> // Matlab engine header
5
6// SystemC Module that create the interface to MATLAB
7SC_MODULE(matlab){
8    ...
9
10    Engine *ep; // Pointer to MATLAB engine
11
12    // Module Constructor
13    SC_CTOR(matlab)
14    {
15        SC_THREAD(mat_sync); // Thread function to sync
16        sensitive_pos << CLK; // Call at each rising clock
17
18        // Start MATLAB engine
19        if(!(ep = engOpen(""))){
20            cout << "Can't start MATLAB engine" << endl;
21            exit(-1); // Exit on error
22        }
23        engSetVisible(ep, 1); // Makes Matlab session visible
24        wd_path = get_wdpath(); // Get the working directory
25        engEvalString(ep, wd_path); // Change MATLAB working directory
26    }
27
28    // Module Destructor
29    SC_END()
Role of S-Functions

- Read SystemC data from the MATLAB base workspace
- Create I/O ports
- Create an input persistent class object in memory
- Synchronize the burst mode
- Data generator
- Data analysis
- Golden model
- Write Simulink data in the base workspace
- Pause the simulation
- Create an output persistent class object in memory
- Synchronize Simulink with SystemC
Simulator Synchronization

- SystemC is master of the simulation
- Simulink is controlled from SystemC through MATLAB command prompt with these commands:
  - 'set_param'
    - start, stop, pause, continue, update
  - 'get_param'
    - stopped, initializing, running, paused, updating, terminating, external

```c
// Check if simulation is running
engEvalString(ep, "get_param('firl6b','SimulationStatus')");
mxB = engGetVariable(ep, "ans");
string = mxArrayToString(mxB);
cout << "\nMatlab: simulation is " << string << endl;
```
Data Type Conversion

- MATLAB single object type: MATLAB Array
- MATLAB Engine library uses the ‘mx’ prefixed API routines to manipulate MATLAB arrays in SystemC
  - In SystemC, MATLAB array is declared to be of type ‘mxArray’
  - The API includes over 60 routines to create, access, manipulate, and destroy mxArrays

```c
1 // Declare a C++ MATLAB array
2 mxArray *mxB = NULL;
3 // Read an Array from Matlab
4 mxB = engGetVariable(ep, "data");
```
Example 1

- Verification of a SystemC multi-equalizer with a MATLAB/Simulink data generator

![Diagram of a SystemC multi-equalizer with a MATLAB/Simulink data generator](image)

- Reduce testbench development using a Simulink model
Example 2

- Verification of a SystemC linear equalizer with a Simulink ‘golden’ model reference

- Reuse testbench

- Add robustness to verification with a Golden Reference
Conclusion and Future Work

- Presented an interface from SystemC to MATLAB and Simulink for verification
- SystemC centric approach
- Reduce total verification time through faster testbench development
- Future Work
  - Use dynamic memory allocation for S-Function persistent objects
  - Add flexibility to S-Function block parametrically
  - Improve modularity within SystemC code
  - Other application areas
    - video processing, voice recognition, etc.
Questions

Thank you for your attention.

Contact:

jfboland@macs.ece.mcgill.ca