Ingredients for Successful System Level Automation & Design Methodology

Hiren Patel
hiren@vt.edu

Formal Engineering Research with Models, Abstractions, and Transformations (FERMAT) Research Lab.
http://fermat.ece.vt.edu/

Mentor: Sandeep Shukla
shukla@vt.edu

Presentation for Chess seminar
Setting the Scene

- **Embedded system design**
  - Increasingly complex and heterogeneous

- **Multi-functioned**
  - Voice/data communication, music players, video players, and cameras

- **Using a variety of components with different functionalities**
  - Microcontrollers, memories, and DSP cores

Setting the Scene

- Designers struggle with increasing requirements
  - Tools and methodologies do not scale
- Possible approach for mitigation:
  - Raise the abstraction layer to Electronic System Level (ESL)
- Electronic system level (ESL)
  - “a level above RTL (register transfer level) including hardware and software design” [ITRS 2004]
- But what is the next appropriate abstraction layer?
  - Lack of consensus in the EDA community
  - Proliferation of methodologies and tools
ESL Design Flow

- Exciting opportunities for research
  - Specification, analysis, IP Composition, validation, verification, behavioral synthesis, equivalence checking
- Required: recipe for ESL
  - Prescription [Martin et al., “ESL Design and Verification”]
Possible Recipe: By leveraging other works

Abstract State Machines and Verification
• SpecExplorer [Microsoft Research]
• Habibi and Tahar [Concordia]
• Patel and Shukla [Virginia Tech.]
• Chen and Sztipanovits [Vanderbilt]

Heterogeneous Modeling and Simulation
• Patel and Shukla [Virginia Tech.]
• Herrera and Villar [U. Cantabria]

Directed test case generation
• Patel and Shukla [Virginia Tech.]

Analysis and Synthesis
• Bergamaschi et al. [IBM Research]
• Herrera and Villar [U. Cantabria]
Focus

1. Managing the prevalent need for heterogeneity in today’s designs
   - Heterogeneous models of computation (MoCs)

2. Generating directed test cases for heterogeneous system level designs
   - Model-driven approach for validation

3. Integrating multiple tools and methodologies
   - Service-oriented architecture
Today's Outline

- Overview of MoC Extensions for SystemC
- Directed test case generation with Discrete-Event MoC
Heterogeneity in SystemC
Why SystemC?

- Strong industrial backing
  - Large users community

- Language based on a library of C++ classes

- Recent IEEE standard

- Free modeling and simulation framework
  - Allows for experimentation

- Compiled with open-source compilers such as GCC
Heterogeneity in SystemC

- Introduce heterogeneous Models of Computation (MoCs)
  - Synchronous Data Flow (SDF)
  - Finite State Machine (FSM)
  - Communicating Sequential Processes (CSP)
  - Bluespec’s Rule-based Paradigm
- Extend SystemC simulation framework to support simulation semantics of these MoCs
- Interoperable with Discrete-Event reference implementation of SystemC
Related work

- Ptolemy Project [UC Berkeley]
  - Heterogeneity for embedded software synthesis

- Metropolis Project [UC Berkeley]
  - Heterogeneity and platform-based methodology

- ForSyDE [KTH] & SML-Sys [Virginia Tech.]
  - Heterogeneous modeling and simulation

- HetSC [U. Cantabria]
  - Heterogeneous SystemC
Heterogeneity in SystemC


Some Examples

- **DE with SDF**

  - Encrypted Image Downloads (.mtr)
  - Decryption
  - SDF
  - Image Format Converter (.sob)
  - CleanFigure
  - Channel
  - Sobre

- **CSP with FSM**

Example: Rule-based MoC with SystemC

Example from:
Why Behavioral Hierarchy?

- Natural to decompose behaviors into small behaviors and compose/reuse small behaviors to realize larger behavior
- Traditional HDLs
  - Structural Hierarchy: components within other components connected via ports/signals
    - Encapsulation benefits during modeling and reuse
    - Hierarchy information flattened during simulation
- Behaviors realized by MoCs
- Simulation kernel responsible for simulating one level of hierarchy at a time
  - Refinements invoke another instance of the simulation kernel
  - Example for hierarchical FSM: Starchart semantics [Ptolemy II Group]
Example of Power State Model


Hiren D. Patel
Virginia Tech.
Heterogeneous Behavioral Hierarchy

- Heterogeneous extensions of behavioral hierarchy
- Various behaviors realized by different MoCs are embedded in a variety of ways
  - Examples:
    - SDF embedded in an FSM state
    - FSM embedded in an SDF function block
- MoC-specific simulation kernel simulates each level of hierarchy according to MoC
Polygon Infill Processor Example

Example from:
Modeling HFSM in SystemC

```
SCH_FSM_STATE(init) {
    SCH_FSM_STATECTOR(init)
    { // ... }; 
}

SCH_FSM_ENTRY_ACTIONS() 
{ // ... }; 

SCH_FSM_EXIT_ACTIONS() 
{ // ... }; 
```

```
master.FSM 

hrtr  init  inpt

d  e  f  a  b  c
```
Additional member functions for HFSM

- Adding more than one FSM / SDF refinements
  - add_fsm_refinement(), add_sdf_refinement()
- Three types of transitions
  - Run-to-complete: refinement must traverse from initial to final state before returning control
    - set_run_complete()
  - Preemptive: refinement is not executed
    - set_preemptive()
  - Reset: before changing state, resets the refinement of the destination state
    - set_reset()
Modeling SDF

```
SCH_SDF_MODEL ( hrtr.SDF ) {
    src * s; hrtr * h; sdf_edge * sh;
    SCH_SDF_MODELCTOR( hrtr.SDF ) {
        s = new src("source");
        h = new hrtr("trace");
        sh = new sdf_edge("s_to_trace");
        sh->set_production_rate(1);
        sh->set_consumption_rate(1);
        insert_sdf_block(s);
        insert_sdf_block(h);
        insert_sdf_edge(s,h,sh);
        /// ...
    }
}
```
Integrating Extensions

- Extensions compiled as a library
- Encapsulate toplevel of extended model into a wrapper process
  - Process can be sensitive to other SystemC signals/channels

Wrapper process
- Communicate with master MoC (DE)
- Synchronize via tunnels and interfaces with heterogeneous MoCs
- Data conversion

trigger()
Simulation Experiments

- Pure SDF models show ~65% speed up
- Pure CSP models show insignificant improvement
- Pure HFSM models showed insignificant improvement
- Heterogeneous behavioral hierarchy models showed up to 40% speed up
Model-driven Validation for Heterogeneous SystemC Designs
Design flow using SystemC

- Focus only on concept and design phase
- Functional validation
  - Test scenarios
  - Test suite implementation

Inputs for system to desired state(s)
The possible paths to the desired state(s)
Implementation of input paths
New design flow SystemC

- Semantic level
  - AsmL specification of design under investigation
  - Simulation
Related work

• Bruschi et al [Politechnico Di Milano]
• Kroening & Sharygina [Carnegie Mellon]
• Habibi & Tahar [Concordia University]
• Koo & Mishra [University of Florida]
Microsoft’s SpecExplorer

- Specification, exploration and test suite generation
- Specifications in AsmL
  - Based on ASM formalism
    - Sequential and parallel constructs
  - Serve proof obligations
- Exploration
  - States: grouping and filters
  - Actions: observable and controllable
- Test suite generation
  - Generate test paths from exploration
Model-driven validation for SystemC using SpecExplorer

- We need to provide:
  - Formal description of intended design similar to designing in SystemC
    - Precise formal semantics for the Discrete-event MoC
  - Method for SpecExplorer to drive implementation
Methodology flow

- Discrete event simulation semantics provided
- Test case generation for SystemC
- Design under investigation
- Simulate using SpecExplorer's model simulator
- Controllable and observable actions

Hiren D. Patel
Virginia Tech.
Discrete-event Semantics

\[
\text{update()} \equiv \forall c \in \text{updateChannels} \ c.\text{update}()
\]

\[
\text{initializeDiscreteEvent()} \equiv \\
\begin{align*}
\text{forall b \in \text{behaviorSet}} & \ b.\text{trigger}() \\
\text{let newEvents} \subseteq \text{EVENTSET} = \\
\{ ev \mid ev = b.\text{trigger}() \}
\end{align*}
\]

\[
\text{evaluate()} \equiv \\
\begin{align*}
\text{seq} \{ \\
\text{foreach } ev \in \text{newEvents} & \text{ eventSet := eventSet} \cup \{ ev \} \\
\text{triggerBehaviors(newEvents)}
\}
\end{align*}
\]

\[
\text{processTime()} \equiv \\
\begin{align*}
\text{let deltaEvents} \subseteq \text{EVENTSET} = \\
\{ x \mid x \in \text{eventSet} \land x.\text{ev} = \Delta \}
\end{align*}
\]

\[
\text{triggerDiscreteEvent()} \equiv \\
\begin{align*}
\text{initializeDiscreteEvent()} \seq \text{evaluate()} \seq \text{update()} \seq \text{processTime()}
\end{align*}
\]
Modeling & simulation of simple FIFO example

- For a fixed size $n$
- WR is valid
  - DI in pushed
- RD is valid
  - DO has popped value
  - DV is valid
- FULL is high
  - Size $n$ reached
- EMPTY is high
  - Size 0 is reached
- RESET empties FIFO

Phase A: Semantic model

- FIFO FSM
  - INIT, REQUESTS, WAIT
- REQUESTS
  - Update FULL/EMPTY
  - Write
    - push(DI)
    - Reset WR
  - Read
    - pop and store DO
    - DV is valid
    - Reset RD

```
FSM()
★ require (reqmode = PROCESS )
★ reqmode := READWRITE
  if (mode = INIT)
    mode := REQUESTS
  if (mode = REQUESTS)
    step
      Full() // Update FULL status
      Empty() // Update EMPTY status
    step
    if ( WR.read() = true and FULL.read() = false )
      push(Di.read()) // Throws excep. overflow
      WR.write( false )
    else
      if ( RD.read() = true and EMPTY.read() = false )
        DO := pop() // Throws excep. overflow
        DV.write( true )
        RD.write( false )
        mode := WAIT
      if ( mode = WAIT)
        DV.write( false )
        mode := REQUESTS
```
Phase B: Implementation model

- Translation is intuitive
- Possible errors
  - Overflow
  - Underflow
- Validation in SystemC
Phase B: Exporting SystemC and design actions

- **SystemC wrapper**
  - Must be done once only
  - `sc_start()` to drive simulation
  - Additional members may be exported for debugging purposes

- **Implementation model wrapper**
  - Stimulus that drives model
    - FIFO example
      - `WriteRequest`
      - `ReadRequest`
  - Setup and cleanup
Phase C: Interfacing SystemC, C# and SpecExplorer

- Import actions from SystemC and implementation wrappers
- Abstract C# class
  - Invokes imported actions
- Build C# class as reference library
- SpecExplorer refers to C# library
  - Actions bound to abstract class actions
Phase D: Exploration and test suite generation

- Exploration techniques
  - State groups
  - State filters
  - Accepting states
  - Controllable, observable and scenario actions

- Test suite generation
  - Bind exploration actions to C# interface actions

- Generate test suite
  - All possible paths to accepting states
Testing for overflow

- Size n=3
- Accepting state allowing for n+1 writes
  - Only valid write requests are when in REQUESTS
- Throws an exception
Testing for underflow

- Size $n=3$
- Accepting state allowing for read request when FIFO is empty
- Throws an exception
Summary and Future work

- MoC-driven design to SystemC
  - Work on synthesis from MoC descriptions

- Model-driven methodology for validation of heterogeneous designs in SystemC
  - Extend framework for automatic translation of MoCs
  - Add other MoCs
  - Examples: network routers, FIR filters, …

- Integrate related researches to realize recipe
Sorry – no more slides!