Rhapsody in SystemC (RiSC)

Allen Goldberg
Sodius
Agenda

- Defining System-Level Design
- Relevance of SysML
- Role of SystemC
- RiSC: The Rhapsody SystemC code generator for SysML
- RiSC Case Studies
What is System Level Design?

- System Level Design refers to building devices that include:
  - Custom and off-the-shelf hardware components
  - And that often include processors cores running complex software

- Examples

  - Smartphones
  - Automotive and Aero systems
  - Medical devices
  - Defense/weapon systems using network communication
# System Level Design: Challenges and Needs

<table>
<thead>
<tr>
<th>Challenge</th>
<th>Need</th>
</tr>
</thead>
</table>
| System Performance  | Model system at multiple levels of abstraction in order to study architectural tradeoffs  
|                     | • HW/SW partitioning  
|                     | • Power, time, space  
|                     | • Micro-architecture |
| Time to Market      | • Enable early SW development against HW models  
|                     | • Evolve existing IP |
| Quality             | • Employ a verification process  
|                     | • Requirements tracking |
| Process             | • Need better collaboration between HW, SW, and firmware  
|                     | • Tracing of design artifacts  
|                     |   • Models (SW and HW)  
|                     |   • Requirements  
|                     |   • Test cases |
Addressing the Challenges

- HW/SW integration
  - Enable SW development and testing against HW models early in the design process

- Rich Design Trade-offs
  - Model-based simulation to explore architectural trade-offs
    - models estimate power usage, chip area, speed
  - Models defined at multiple levels of abstraction
    - untimed, transaction level, cycle accurate

- Verification
  - Rigorous verification process supported at each modeling level
  - Requirements traceability

- Process
  - Centralized representation of design artifacts:
    - Unified models (SW and HW)
    - Requirements
    - Test cases
Why is SysML a Good Design Framework?

- Applicable throughout the design process
  - Enables System Engineers to model systems early
  - Build & Use a repeatable Development process
  - Provides a common vocabulary across teams
  - Code generators extend modeling beyond static diagrams

- Enables modeling at different levels of abstraction
  - System’s Engineer View
  - Untimed (Programmer’s View)
  - Programmer’s View With Timing (TLM LT)
  - TLM AT
  - Cycle-Accurate
  - Implementation (Synthesizeable SystemC model)

- Integrates all artifacts in a single framework
  - Requirements
  - Models / Code
  - Test Cases
SysML System On a Chip (SoC)
SysML Simulation/Execution
Role of SystemC
Trends of Electronic Design

- Register Transfer Level (RTL) synthesis mature, but...
- Hardware complexity continues to increase
  - RTL-based flows are becoming inadequate
- Industry pushing up to a higher level of abstraction
  - High-level synthesis
  - HW/SW co-simulation (i.e., Virtual platforms)
- Trending toward standardization on the use of SystemC and Transaction Level Modeling (TLM)
C++ / SystemC / TLM Summary

- Decouples communication from functionality
- Standardized and customizable communication protocols

SystemC

- C++ class library
- Event-driven simulation kernel
- Modules, ports, signals, interfaces

C++

- High-level general purpose language
- Object-oriented
SystemC Use Cases

- **Architectural Exploration and Analysis**
  - Partitioning
  - Design trade-offs (i.e. fifo depth, latency, …)
  - Provides loosely-timed HW model for SW dev
  - *TLM* mode – transaction level view

- **Implementation**
  - Generate SystemC synthesizable subset
  - Generated SystemC flows into HLS tool chain for HW implementation
  - *Synthesis* mode

- **Verification**
  - Cycle-level timing (*Cycle Accurate* mode)
  - Increased concurrency in targeted framework
  - Cycle-accurate SystemC reference model used to validate RTL
System Modeling with SystemC: TLM

- Provides model for SW dev and high-level architectural exploration.

- TLM 2.0 standard calls out two coding styles
  - Loosely Timed: Component interaction encapsulated as tlm transactions.
    - Temporal Time aka Time Warping.
  - Approximately Timed: Partial refinement of transactions to protocols. Limited use of time warping.

- Temporal decoupling techniques minimize context switches during simulation to speed execution
  - Goal is fast simulation (100 to 1000x over RTL)
SystemC HLS Synthesis

- OSCI defines a SystemC synthesizable subset that can be fed directly into HLS tools
- Lower-level specification, but need not provide hardware details
- SysML model specifies communication and protocols
- Timing primarily determined by results of HLS
- Some restrictions on SysML input based on SystemC synthesizable subset requirements:
  - HLS tool determines structure from constructor
  - Limits on types of communication channels
  - No dynamic allocation or unbounded loops
Cycle Accurate Modeling

- Timing Accurate. Agreement between the simulated and actual time for communication and computation steps.

- Default Timing Model:
  - One step of a statechart takes one execution cycle
  - Communication takes one cycle

- Default timing model can be altered by user annotations (primarily wait statements).

- Concurrency using “and” states in statechart modeled with fine-grained (cycle accurate) concurrency.
HW/SW System Level Design Flow

- System Model
- Analysis
- Software
- Hardware Model
- Integration
- Virtual Platform
- Validate and refine

- TLM: Loosely Timed
- TLM: Approximately Timed
- Cycle-Accurate
HW/SW System Level Design Flow – Using SysML

- System Model maintains integrity across HW and SW
- Models emphasize
  - Functionality
  - Composition
  - Interfaces
- Models are executable
- Models remain consistent
- Ensures smooth integration
RiSC: The Rhapsody SystemC code generator for SysML
HW/SW System Workflow

DOORS: Requirements

import

generate

Rhapsody: UML/SysML

refine

co-execution & validation

Software: C/C++

Hardware: SystemC, HDL, ...

Automation GAP!

Electronic Design Level Tool Chain

SystemC

Transaction Level

Cycle-Accurate

Verification

SystemC

Register Transfer Level

HDL

Netlist

GDSII

Placement

Routing
Rhapsody in SystemC (RiSC)

- Enables SystemC generation from SysML models
  - Simulation (TLM)
  - High Level Synthesis

- Fully integrated with Rhapsody
  - Generate/Build/Make directly from menus
  - Intelligent generation
  - Code round-tripping (code/model synchronization)
  - Animation (simulation) support
  - Code navigation from browser
  - Compilation/Build error navigation

- Standards based solution
  - UML / SysML
  - SystemC / TLM 2.0

- Compatible with Collaboration Processes / Tools
  - Harmony SE
  - DOORS
  - RTC
May go at end
mbakal, 3/12/2013
Rhapsody in SystemC

SysML Annotations
- Timing spec
- Clock/Reset options
- TLM 2.0 options
- other options

RiSC

//Constructor
NCU_User::NCU_User(sc_module_name instanceName) : sc_module(instanceName)
  , pMobile_PlaceCall_out("pMobile_PlaceCall_out")
  , pMobile_Disconnect_out("pMobile_Disconnect_out")
  , pMobile_evOn_Off_out("pMobile_evOn_Off_out")
{
  SC_THREAD(executeFsm_0);
  SC_THREAD(executeFsm_1);
  SC_THREAD(executeFsm_2);
}
Compatibility with Different Abstraction Levels

**RiSC provides TLM interface capabilities**

**RiSC targets SystemC synthesize-able subset**

**RiSC supports cycle-accurate coding style and wrapper generation to ease integration**
Untimed $\rightarrow$ Timed Model
Rhapsody in SystemC Advantages

- Traceability across entire design in a single unified model
- Effective reuse since architecture and design elements trace to requirements
- Collaboration between systems, hardware, and software engineering can be supported in a single environment
- Provides platform for design reviews
- Graphical modeling allows working at higher levels of abstraction
  - auto-generation of SystemC is more efficient and less error-prone
  - Visualize state-driven behavior of system and hardware designs
  - SysML model execution validates high-level system designs
    - e.g. – inter-block and intra-block communications
- Componentize HW + SW to optimize and make trade-off decisions
RiSC: Case Studies
TLM Case Study: Global Semiconductor Company (cont.)

```
CPU + TLM Interface
   CPU
     ReadTrans()
   TlmAdapter
     <SysC Interface>
   Bus
     Mem
     UART
     ...
TLM Case Study: Global Semiconductor Company

- **Objective:** Achieve significant reduction in design cycle time to increase competitive advantage
- Constructed SysML and TLM models for SoC architecture exploration
- Synthesized SystemC for virtual hardware prototypes
  - eliminate software engineering waiting for availability of hardware (or new versions of hardware)
- Reducing design cycle times from multiple months to weeks
- Traceability provides other benefits
  - confidence in building correct and complete product
  - reuse of features and architecture in product variants
Synthesis Case Study: Major Aerospace & Defense Technology Company (cont.)

Diagram:

- **SysML**
  - Stimulus Generator
  - Function
  - Checker
  - RiSC
  - SystemC
  - High-Level Synthesis

- **FPGA HDL**
- **RTL Synthesis**
- **RTL HDL**
Synthesis Case Study: Major Aerospace & Defense Technology Company

- Communication Block in Hardware
- Data Flow and Signal Processing Functions:
  - Routing
  - QPSK Encoding/Decoding
  - STAP Filtering
- SystemC automatically synthesized from SysML
  - Targeting SystemC synthesizable subset
- Utilized high-level synthesis tool to produce RTL
- Achieved and confirmed accurate predication of hardware performance estimates
Cycle-Accurate Case Study: Market-Leading Medical Device Manufacturer

- Modeled moderately complex blocks taken from medical device
- Produced cycle-accurate golden reference model in SysML.
- Generated SystemC from the model using RiSC
- Integrated SystemC model into verification environment
- Validated RTL against SystemC golden reference for thousands of test cases.
Thank You

www.ibm.com/software/rational
Backup
SysML Structural View

- Structure represented in a way familiar to HW designers
- Hierarchical Components
- Ports with interfaces
- FIFO channels
SysML Behavioral View - Statechart

State-based Behavior of the FingerprintScanner Block
Key requirements being satisfied - Traceability

<table>
<thead>
<tr>
<th>ID</th>
<th>System Requirement</th>
<th>&lt;&lt;satisfy&gt;&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS11-3</td>
<td>Employee ID Card Identification - Exit</td>
<td>readSecurityCard(), validateSecurityCard()</td>
</tr>
<tr>
<td>SS11-4</td>
<td>Biometric Scan</td>
<td>scanBiometricData()</td>
</tr>
<tr>
<td>SS11-5</td>
<td>Time Between Two Independent Checks</td>
<td>t_Bs</td>
</tr>
<tr>
<td>SS11-7</td>
<td>Three Attempts On Employee ID Entry</td>
<td>ScFailCount, flagSecurityCardFailure()</td>
</tr>
<tr>
<td>SS11-8</td>
<td>Three Attempts On Biometric Data Entry</td>
<td>BsFailCount, flagBiometricDataCheckFailure()</td>
</tr>
<tr>
<td>SS11-9</td>
<td>Three Attempts On Employee ID Exit</td>
<td>ScFailCount, flagSecurityCardFailure()</td>
</tr>
<tr>
<td>SS11-10</td>
<td>Denied Entry Notification</td>
<td>logEntryData(), logAccountData(), reqProcessAlert()</td>
</tr>
<tr>
<td>SS11-11</td>
<td>Denied Exit Notification</td>
<td>logExitData(), logAccountData(), reqProcessAlert()</td>
</tr>
<tr>
<td>SS11-12</td>
<td>Alarm - Entry</td>
<td>alarm()</td>
</tr>
<tr>
<td>SS11-13</td>
<td>Alarm - Exit</td>
<td>alarm()</td>
</tr>
<tr>
<td>SS11-18</td>
<td>Alarm Reset</td>
<td>resetAlarm()</td>
</tr>
<tr>
<td>SS11-14</td>
<td>Disabling User Account</td>
<td>disableUserAccount()</td>
</tr>
<tr>
<td>SS11-15</td>
<td>Visualization of Security Card Check Status - Entry</td>
<td>displayCardStatus()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>System Requirement</th>
<th>&lt;&lt;satisfy&gt;&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS11-16</td>
<td>Visualization of Security Card Check Status - Exit</td>
<td>displayCardStatus()</td>
</tr>
<tr>
<td>SS11-17</td>
<td>Visualization of Biometric Data Check Status</td>
<td>displayAuthenticationStatus()</td>
</tr>
<tr>
<td>SS11-2</td>
<td>Access Precondition</td>
<td>validateSecurityCard()</td>
</tr>
<tr>
<td>SS11-4</td>
<td>Out of Date Cards - Entry</td>
<td>validateSecurityCard()</td>
</tr>
<tr>
<td>SS11-5</td>
<td>Out of Date Cards - Exit</td>
<td>validateSecurityCard()</td>
</tr>
<tr>
<td>SS11-12</td>
<td>Approval of Biometric Data</td>
<td>authenticateBiometricData()</td>
</tr>
<tr>
<td>SS12-2</td>
<td>Entry Time</td>
<td>t_Unlocked</td>
</tr>
<tr>
<td>SS12-3</td>
<td>Exit Time</td>
<td>t_Unlocked</td>
</tr>
<tr>
<td>SS12-5</td>
<td>Automatic Securing the Secure Area - Entry</td>
<td>evAccessPointLocked()</td>
</tr>
<tr>
<td>SS12-6</td>
<td>Automatic Securing the Secure Area - Exit</td>
<td>evAccessPointLocked()</td>
</tr>
<tr>
<td>MA1-1</td>
<td>Image Capture</td>
<td>reqTakeSnapshot()</td>
</tr>
<tr>
<td>MA2-1</td>
<td>Time Recording</td>
<td>logExitData()</td>
</tr>
<tr>
<td>MA2-2</td>
<td>Alarm Conditions</td>
<td>checkForTimeLimitViolations()</td>
</tr>
</tbody>
</table>
Definition of the use case functional flow ("storyboard")
Why is SysML a Good Design Framework?

- Untimed Programmer’s View: Kahn Process Network
  - actors
  - events or transactions trigger processing
  - causal dependency of events, but no explicit timing
  - event queues mitigate timing
    - precise timing synchronization not required

- Traditional use of pure C/C++ obscures view of model architecture and communication, which are explicit in SysML

- Product line development
  - Architecture evolves slowly
  - Lower level features evolve faster
Use Case Diagram: System Use Cases

Requirements Diagram: Visualization of Links between Stakeholder Requirements and System Requirements

Functional Decomposition
Allocation

Internal Block Diagram IBD_Security System

White-Box Activity Diagram Uc1ControlEntry

White-Box Activity Diagram Uc2ControlExit
Tools and Technologies

- Requirements Analysis
- Architectural Analysis
- Functional Analysis/Allocation
- Implementation
- Integration and Test

Tools:
- Rhapsody – UML/SysML
- SystemC/HDL
- Doors
Systems Engineering Process

1. Requirements Analysis
2. Architectural Analysis
3. Functional Analysis/Allocation
4. Implementation
5. Integration and Test

Hardware and Software
Rhapsody in SystemC (RiSC)

- Critical to successful model-based design methodology
- Leverage SysML executable models and validation
- Expedite availability of domain-appropriate models for software and hardware development
- Ensure consistency with Systems Engineering perspective
- Reduce errors introduced through manual SystemC creation
- Easily propagate top-level changes to lower levels
Overview of Translation from SysML to SystemC

- **Blocks**
  - SystemC and SysML view an architecture as hierarchical collection of blocks
  - Primarily static
    - No new SystemC modules may be allocated after elaboration completes
  - RiSC maps blocks to SystemC `sc_module`
    - RiSC determines when a block is just an auxiliary class and is not mapped to a `sc_module`
  - RiSC generates code for statecharts
    - Statechart behavior initiated by asynchronous communication of events, similar to SystemC signals
  - Blocks optionally contain `sc_thread` process that run statechart behavioral code or depend on a parent module
    - Active/Sequential option
  - Generate a .h and .cpp file for each block
    - Dependency analysis determines includes
Asynchronous Communications

- Models signals and interrupts
- One way message communication
- In SysML messages are events.
- Events can be parameterized
  - ev(interruptCode)
- SysML model
  - Events are queued in a thread
  - Dispatched to a statechart in FIFO order
  - Statechart behavior may generate new events
Synchronous Communications

- Method call
  - Runs in the thread of the caller
- Ideally calls between blocks are made via ports
- Called remote operations in SysML
- Transaction level modeling in SystemC
An architecture is a (generally) static hierarchy of components

In SysML you specify the parts of a block, i.e. its components

These are instances (i.e. objects) of other blocks (i.e. classes)

Specify one or more top-level instances, and this creates the architecture
In both SysML and SystemC good practice requires:
- Blocks of the architecture communicate through ports
  - Synchronously or asynchronously
  - No global state

Ports define interfaces that specify the events and methods that can be communicated over the port.

While the concept of port exists in SysML and SystemC there are differences in the details that are managed by the RiSC generator.

SysML contracts defines a collection of interfaces the port allows and allows bi-directional communication.

SystemC communication is one-directional.

Asynchronous communication is mitigated by channels:
- Sc_fifo
- Sc_signal
Links

- Ports are connected to each other via links
- Links are between sibling components or parent/child in a well-structured design and this is required by RiSC
- To summarize an architecture is the static hierarchy of block instances, together with the communication structure implied by ports and links
Generating SystemC Ports and Links

- RiSC deals with these differences
  - Splitting ports with multiple contracts
  - Generates channels when required
    - RiSC uses custom channels
  - Generates sockets for TLM 2.0 communication

- RiSC insures classes defining port interfaces inherit from `sc_interface`

- RiSC supports
  - Delegation ports: ports that forward operations/events to parents or children allowing well structured communication
  - Multi-ports
    - Fan in
    - Fan out
    - Broadcast
  - Port policy checking
Types, Relations, and Global Types

- SysML allows
  - definition of types e.g. enumerations, unions
  - Relations: which generate additional attributes and methods
  - Global functions

- Rhapsody in SystemC supports these constructs, although these are more important for software-centric development.
Packages

- Using SysML blocks, events, types, parts are defined within packages
- Packages are hierarchical: packages may contain sub-packages
- RiSC generates files for a package when it contains events, parts, or types
  - Recall blocks are generated into separate files
- RiSC can generate a C++ namespace corresponding to a package
SystemC is a timed simulation kernel.
- A SystemC thread may invoke `wait(sc_time(10, sc_ns))`
  - The thread is suspended, and a notification to resume the process is generated and places on a time-ordered queue
  - When the notification is on the front of the queue the system simulated time is advanced.
  - Suspending a thread is sometimes referred to as yielding. It requires a context switch which is computationally expensive.

- `wait()`’s are used to model computation or communication delay.
- User code may invoke `wait()`. (User code runs in sc_threads)
  - States and transitions may be annotated with tags that specify timing delays or `wait()` may be used directly.

- When a thread has no further events to process it issues a wait on an event notification generated when some other thread sends it an event.
- RiSC generation modes adjust timing behavior and some other code generation strategies depending on use cases.
TLM 2.0 defines transport interfaces.

When ports use the transport interfaces then specialized ports called sockets are generated.

Automatic generation of temporal time keepers for each thread:
- `tlm_global_quantum`
- Time keepers operate differently for AT or LT styles.

Threads yield when event queue empties or user code invokes `wait()`.

Event queues order events based on local time stamps.
Rhapsody in System C: Synthesis Mode

- Insure tools can determine static structure of the design. Simple and restricted behavior in constructors.
- Generation of clock and reset ports and a clock generator.
- Reset behavior in sc_clock_thread.
- Handshaking protocols generated used to communicate between testbench and design.
- Specialized memory ports.
- Efficient synthesizable statechart implementation.
Synthesis Case Study: Major Aerospace & Defense Technology Company

- SysML + synthesizable SystemC as core Model-based Systems Engineering Method (MSEM)

- Objectives
  - realize cost savings, schedule reduction and risk reduction
  - enable communications and collaboration between systems, software, and hardware engineers”

- Synthesizable SystemC model describes “Executable Specification” which becomes the design baseline

- Workflow: Capture design in SysML → Synthesize SystemC → Synthesize VHDL → Target hardware
SysML Structure and Communication

- CPU Part
- Synchronous interface
- Asynchronous interface

- Uart Part
- Link
- Port

- Term Part
Specifying Timing in SysML

- RiSC provides various mechanisms for the user to specify timing
- Method used depends on context and user preference

- **Action Language Macro**
  - Timing macros embedded in action code specify a delay duration
  - Semantics of the timing depends on the timing mode
  - Example:
    ```
    ...  
    TLM_INC(10ns);  
    ...  
    ```

- **Delay Tag**
  - SysML tags allow delays to be associated with statechart elements
    - State entry/exit action
    - Transition action
Specifying Timing in SysML: Delay Tags
Specifying Timing in SysML: Timeout Transition

- Timeout Transition

Diagram showing a state transition from idle to state_1 with a timeout event labeled tm(20ns).
Customer Discovery Questions

Are you using UML/SysML?
  - For software, hardware, or both?
  - Are you using executable models?

Are you using SystemC?
  - What level of abstraction?
    - Transaction Level Modeling (TLM 2.0)
    - Cycle Accurate
    - Synthesizable Subset
      - Which High Level Synthesis tool?

Are you using a Virtual Platform?
  - Which tool/vendor?