TKT-1527 Digital System Design Issues
Tero Arpinen

Introduction to SoC modeling and Models of Computation
Reference material


Outline

Fundamentals of SoC modeling
- What is a model?
- What is a good model?
- What are the typical goals of SoC modeling?

Developing modeling languages, and tools
- Steps in development of modeling languages and tools

Introduction to Model of Computation (MoC)
- General building blocks of MoC
- Classification of MoCs
  - Notion of time
  - Communication semantics

Example
- Performance evaluation of embedded streaming applications with system-level simulation
Fundamentals of SoC modeling
Motivation

- Current System-on-Chips are heterogeneous and complex
  - Several microcontrollers, memories, custom HW, FPGAs, etc.
- Complexity grows much faster than the system size (number of components) due to component interaction
- In order to manage the complexity, the system should be modeled at a high level of abstraction prior to implementation
- Validation of the system using high level of abstraction saves enormous amount of time and effort
What is a model?

- Model is a *simplification of reality*
- Model highlights the aspects of the system that are relevant for the problem to be solved
- Model can be built for:
  - Specification (documentation)
  - Static analysis (mathematical calculation)
  - Dynamic analysis (simulation)
  - Model transformation (producing other models)
  - Implementation (detailed, architecture dependent)
Typical goals of modeling in SoC design

- Validating and verifying the functional correctness of design
- Design-space exploration
  - Mapping the application onto platform
- Performance evaluation
  - Throughput, response times of tasks
  - Platform computational capacity
- Power consumption estimation
- Resource consumption estimation
  - Number of gates, memory bits, etc.
Weapons for modeling

- Abstraction and encapsulation
  - Hide irrelevant details
- Focusing on the correct aspects
  - Hide irrelevant aspects
  - Leads to abstraction
- Form hierarchies
  - Decomposition
  - Divide and conquer
  - Leads to abstraction
Decomposition in SoC HW design

Hemani’s law: size of reused component grows 10x per every decade
The power of decomposition

- Decomposing a complex system into smaller hierarchical parts is mandatory during the design.
- This is due to limitations of human mind to cope with chunks of information.
- Human can comprehend only 5-9 new chunks of information simultaneously.
- Also, it takes about 5 seconds to accept a new chunk of information.
Types of hierarchy

- **Composition**
  - ”Part of” hierarchy
  - Abstraction is based on identifying larger objects composed of smaller ones
  - Used e.g. modular design approach

- **Inheritance**
  - ”Is a” hierarchy
  - Abstraction is based on identifying common properties shared between objects
  - Used e.g. by scientific classification in biology and object-oriented programming languages
Five attributes of complex system

- **Hierarchic structure**
  - Complex system is composed of interrelated subsystems that have in turn their own subsystems, and so on.
  - Complex system is a function of its components and hierarchic structure of its components

- **Relative primitives**
  - The choice of what components of the system are primitive is dependent on the observer.
  - E.g. transistor vs. logic gate or algorithm vs. CPU instruction
Five attributes of complex system

- **Separation of concerns**
  - Intra-component relationships are stronger than inter-component relationships
  - Separating high-frequency dynamics of components (its internal behavior) from low-frequency dynamics (interactions between components)

- **Common patterns**
  - Reuse of small identical components (e.g. cells in human body)

- **Stable intermediate forms**
  - Complex systems are evolved from simple systems much more rapidly if there are stable intermediate forms than if there are not
Metrics for good model (1)

- Abstract
  - Emphasize important aspects while removing irrelevant ones

- Understandable
  - Expressed in a form that is readily understood by observers
  - Uses vocabulary of the domain experts

- Accurate
  - Faithfully represents the modeled system
  - Analysis produces relevant results
Metrics for good model (2)

- Predictive
  - Can be used to answer questions about the modeled system

- Inexpensive
  - Much cheaper and easier to construct and study than the modeled system

- To be useful, engineering models must satisfy all of these characteristics!
Different notations of a model description

**Textual**
- Mathematical expressions, programming languages, XML
- Well-suited for describing sequential behavior, machine readable, good portability
- Requires only emacs to describe the model

**Graphical**
- Block diagrams, process graphs, state machine diagrams
- Suffers from portability issues
- The capturing always needs a specific tool, there is no ‘graphical emacs’ (at least not yet)

**Hybrid**
- E.g. a process graph annotated with textual constraints
System model processing and analysis

- First the system is modeled with a modeling language
- Then the system is analysed using dedicated analysis methods and tools
- System analysis methods can be divided into
  - Static analysis
  - Dynamic analysis
Dynamic analysis methods

- Dynamic methods are based on executing the system model with simulations.

- Simulations can be divided into:
  - Cycle-accurate (e.g. RTL VHDL simulation)
  - System-level (e.g. TLM SystemC simulation)

- Takes into account sporadic effects in the system behavior:
  - Aperiodic interrupts and events
Static analysis methods

- Static methods are based on symbolical processing of the model.
- Typically used in early design-space exploration to find different corner cases.
- Static methods are faster than dynamic methods.
- Static methods provide larger coverage of the design-space than dynamic.
- But are less accurate as they cannot consider dynamic aspects of the system.
Summary: benefits of modeling

- Problems are better understood
- Problems are easier to analyse and solve
- Problems are spotted earlier in the design
- Problems are easier to communicate between designers
- Altogether, time and effort is saved
Conclusion

The purpose of modeling is to make things simpler to understand than they appear to be in the hard and cold reality.
Developing modeling languages and tools
Steps for developing modeling languages and analysis tools

1. Formulation of meta-model
   - Defining semantics for the modeling language

2. Developing methods for model presentation
   - Defining notations for the modeling language

3. Development of analysis tools according to defined modeling methods
Formulation of meta-model

- Meta-model is a model behind a model
- It determines the rules how a model is constructed
- It defines model elements that can be used for modeling and their semantics
- The formulation is done by considering the purpose of the model
Formulation of meta-model

- Meta-model determines
  - Abstraction level of the model
  - Accuracy of the model
  - Effort required to perform modeling and analysis
  - Whether the model can be executed, simulated, or statically analysed

- Methods for meta-modeling
  - Natural language
  - Mathematical expressions
  - Meta-Object Facility (Subset of UML)
Developing methods for model presentation

- Defining how the model is captured by a designer
- In practice done by selecting an existing language or creating a new modeling language
- Requires transformation rules to be defined between the elements of the meta-model and the elements of the modeling language
  - 1-to-1 mapping is desired
On creating modeling language

- Meta-modeling should be performed
  - With full concentration on the primary objectives of the modeling (or analysis)
  - Independently from the resulting modeling language

- Model semantics (the meta-model) truly determine whether the model is usable

- Presentational features contribute to the feasibility of the model for a human designer
Development of model analysis tools

- The implementation of the tools should follow the created meta-model and its original objectives.
- The modeling language and tools are linked together with model transformations.
- Transformations are used to convert the notations of the modeling language to the format understood by the tools, while the semantics of the model is maintained.
Introduction to Model of Computation
Model of Computation

- Model of Computation (MoC) defines how computation of a system takes place in a structure of concurrent processes
- MoC semantics can be used to formulate an abstract machine capable of executing a model
- MoC is typically described with process (or task) networks
- MoC describes the notion of time in the process network
- MoC determines the semantics how processes communicate with each other
Model of Computation

Choosing a suitable MoC for the analysis is very important since each MoC have their own properties.

This is often done by selecting an existing design/modeling language.

Design languages are not MoCs themselves but they have underlying computational models.

For instance VHDL, Verilog, and SystemC share the same discrete-time, event-driven computational model.
General building blocks of MoC

- Event
- Signal
- Process
- Evaluation cycle
Event

- Also sometimes referred as *token*
- Elementary unit of information exchange between processes
- Basic element on signals
- Events either are or contain values

\[ S_1 = \langle e_1, e_2, e_3.. \rangle \]

Diagram:

- **P1**
- **P2**
- **S1**
- **S2**
Signal

- Also referred as *channel*
- Ordered sequence of events
- Medium through which events are communicated from one process to another
- One and only one process can emit into a signal, but one or several can receive events from a signal
  - Signal has only one ”driver”
- Signal may be either finite or infinite in length

\[ S_1 = <e_1, e_2, e_3..> \]
Process

- Processes are defined as functions on signals
- $p : S \to S$
- Process consumes events and emits events
- Processes may or may not have an internal state
- The notation based on finite state machines (FSM) with initial state ($S_0$), next state function ($g$) and output encoding function ($f$) are used as principal components with processes
Evaluation cycle

- The activity of processes is divided into evaluation cycles.
- In each evaluation cycle a process consumes inputs, computes its internal state and emits outputs.
- A process partitions its input and output signals into subsequences corresponding its evaluation cycle.
- During each evaluation cycle a process consumes exactly one subsequence of each of its input signals and emits exactly one subsequence of each of its output signals.
Example

Process P chooses the minimum and maximum of three incoming values and emits them to output

\(<e_0, e_1, e_2>\)
1st evaluation cycle (i = 0)
2nd evaluation cycle (i = 1)
Notion of time

Salvador Dali, *The Persistence of Memory*, 1931
MoC categories based on notion of time

- Timed
  - Continuous-time (e.g. Simulink)
  - Discrete-time (VHDL)
    - Synchronous time (RTL FSM)

- Untimed
  - Kahn process network (~course exercise)
  - Data flow process network
  - Synchronous data flow
Continuous-time models

- Time represented as a continuous set as *real* number values.
- Behaviour represented as equations over real numbers.
- Simulators for continuation time are typically **very slow**
  - Due to need for solving differential equations.
- Thus, only small parts of system are modeled with continuous time models.
  - Typically analogue and mixed-signal parts.
Continuous-time models

- Examples of continuous time MoC languages
  - Simulink
  - VHDL-AMS
  - Modelica

- *Mixed-signal languages* allow modeling of pure digital parts in discrete-time and analog parts in continuous-time MoC
  - This allows simulations with acceptable performance
  - E.g. VHDL-AMS
Discrete-time models

- All events are associated with a time instant and time is represented with a discrete set
  - For example with integer or natural numbers

- Both VHDL and Verilog use discrete-time model for their simulation semantics

- A simulator for discrete-time MoCs is usually implemented with a global event queue that automatically sorts occurring events
Discrete-time vs. discrete-event

- Discrete-time **is not** the same as discrete-event
- Discrete-event means that values of events are taken from a discrete set
- So there are four different combinations
  - Cont.-time / Cont.-event
  - Cont.-time / Discr.-event
  - Discr.-time / Cont.-event
  - Discr.-time / Discr.-event
Synchronous models

- Special case of discrete time model
- Elementary unit is not a physical unit but more abstract unit (clock cycle, evaluation cycle)
- Timing activities and events is not precisely defined but constrained by beginning and end of an elementary time slot
- Timing of intermediate events that are not visible at the end of an elementary time slot is ignored
Synchronous models

- Synchronous approach considers ideal reactive systems that produce their outputs synchronously with their inputs.
- Global clock triggers computations that are simultaneous and instantaneous.
- This frees the designer from modeling complex communication mechanisms.
Synchronous models

- Synchronous design technique has been used in HW design for clocked synchronous circuits
- A circuit behavior can be described by separating
  - Combinational blocks
  - Clocked registers
- Implementation will have the same behavior as the abstract circuit as long as combinational blocks are fast enough
  - Critical path
Untimed models

- Adopts the simplest timing model
- Processes are combined together with signals carrying events which do not carry timing information
- However, signals still preserve the order of the emitted events; that is, events that are emitted first by the sender are received first
- Theoretically this means that the time interval between two consecutive events on a signal can vary from the positive limit of zero to infinity
Kahn process networks

- Processes communicate via unbounded FIFO channels
  - Non-blocking write; writing always succeeds and do not stall the process
  - Blocking read; reading from empty channel stalls the process until the channel contains proper tokens

- Monotonic processes: process does not need the whole input signal to start the computation of output events
  - This allows representation of parallelism

- Processes cannot test an input channel for a token without consuming the token

- Partially-ordered events
  - Events in signals are ordered
  - No order relation between events in different signals
Data flow process network

- Variant of Kahn process network
- Directed graph of nodes and arcs
  - Nodes represent actors
  - Arcs represent ordered sequences of events
- The execution is sequence of firings or evaluations
- On each firing, tokens are consumed and produced
- Nodes have firing rules that determine the number of consumed and produced tokens on each firing
- Used for modeling DSP applications
Synchronous data flow (SDF)

- Data flow model that requires that a process consumes and produces a fixed number of tokens on each firing
Communication abstraction and semantics
## Communication abstraction levels

<table>
<thead>
<tr>
<th></th>
<th>Example description method</th>
<th>Paradigm</th>
<th>Media</th>
<th>Addressing</th>
<th>Protocol</th>
<th>Data</th>
<th>Real-time behavior</th>
<th>Simulation /analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Service</strong></td>
<td>CORBA, UML, Java Messaging</td>
<td>Un-timed functional</td>
<td>Abstract network</td>
<td>By object discovery</td>
<td>Service request: do something</td>
<td>Abstract, object itself</td>
<td>request-response causality (Unpredictable)</td>
<td>1k-10Mx speed, computer network</td>
</tr>
<tr>
<td><strong>Message</strong></td>
<td>UML, SysML, SDL, Matlab</td>
<td>Un/timed functional</td>
<td>Channels</td>
<td>Abstract, explicit</td>
<td>High-level primitives, send/receive</td>
<td>Abstract, signal/message</td>
<td>Ordering of messages</td>
<td>1k-1Mx speed, computer simulations</td>
</tr>
<tr>
<td><strong>Transaction</strong></td>
<td>SystemC, Simulink</td>
<td>Bus cycle accurate</td>
<td>Logical links</td>
<td>Logical, explicit</td>
<td>Read/write, wait for new event</td>
<td>Varying size units</td>
<td>Synchronized to bus cycle</td>
<td>100x speed, TLM simulator, e.g. TUT transaction generator</td>
</tr>
<tr>
<td><strong>Transfer</strong></td>
<td>VHDL, Verilog</td>
<td>Cycle accurate</td>
<td>Logical links</td>
<td>Logical, explicit</td>
<td>Read/write, wait for new clock cycle</td>
<td>Fixed size units</td>
<td>Synchronized to clock cycle</td>
<td>1x speed, Modelsim, Instruction set simulator, emulator</td>
</tr>
<tr>
<td><strong>RTL</strong></td>
<td>VHDL, Verilog</td>
<td>Pin accurate</td>
<td>Physical links</td>
<td>Physical, explicit</td>
<td>Set/reset, wait for new clock cycle</td>
<td>Bits on links</td>
<td>Synchronized to clock cycle</td>
<td>1x speed, Modelsim, Instruction set simulator, emulator</td>
</tr>
</tbody>
</table>
Semantics for asynchronous communication

- Used to categorize untimed MoCs
- Blocking vs. non-blocking
  - Blocking read
    - Process reading from empty signal/channel will stall
    - Process must wait for proper input events to arrive before continuing its operation
  - Blocking write
    - Process must wait for successful write before continue its operation

Source: Alberto Sangiovanni-Vincentelli
Semantics for asynchronous communication

- Buffers used to adapt when sender and receiver have different rate
- Lossless vs. lossy
  - Events/tokens may be lost
  - Bounded memory: overflow or overwriting
  - Need to block the sender
- Single vs. multiple read
  - Result of each write can be read at most once or several times
Typical asynchronous communication models

- **Rendezvous**
  - No space is allocated for the data, processes need to synchronize in some specific points to exchange data
  - Read and write occur simultaneously

- **FIFO**
  - Bounded
  - Unbounded

- **Shared memory**
  - Multiple non-destructive reads are possible
  - Writes delete previously stored data

Source: Alberto Sangiovanni-Vincentelli
Example: Performance evaluation of embedded streaming applications with system-level simulation
Problem statement

What happens to performance

System engineer

New functionality

Web browser

Existing system

GUI, email, calendar, tetris, MP3 player, etc.

Speech codec, Channel equalization, Modulation

Application model library

Platform model library

SW platform
Custom HAL

SW platform
Symbian

SW platform
Custom HAL

Communication network

TI 55X

ARM11

ASIP

SW platform
Custom HAL

Application model library

Platform model library

Existing system

GUI, email, calendar, tetris, MP3 player, etc.

Speech codec, Channel equalization, Modulation

What happens to performance

System engineer

New functionality

Web browser

Existing system
Results of the simulation are early estimations on PE, memory, and on-chip network utilization, task response times among other information that is used for design-space exploration.
Model of Computation

- Application MoC resembles Kahn process network
- Application tasks are mapped onto platform resources
- Platform resources schedule and execute tasks
- No detailed behavior of task is required, only their workload characteristics (operation counts)
Segment of example application workload model

<<WorkloadEvent>>
Videoinput
{eventKind = periodic,
 sendAmount = "1",
 sendProbability = "1.0",
 time = "1.0/fr"}

<<ExecutionWorkload>>
PreProcessing
{intOps = 56764,
 sendAmount = "MBPixelSize*BPP/8",
(Encoder:::)}

<<ExecutionWorkload>>
MotionEstimation
{intOps = 29231,
 sendAmount = "MBPixelSize*BPP/8",
(Encoder:::)}

<<ExecutionWorkload>>
MBtoFrame
{intOps = 5440,
 sendAmount = "MBPixelSize*BPP/8",
(Decoder:::)}

<<ExecutionWorkload>>
MotionCompensation
{intOps = 4222,
 sendAmount = "MBPixelSize*BPP/8",
(Decoder:::)}
Example platform performance model

<<hwProcessor>>
<<PePerformance>>
<<ep_allocated>>
cpu1 : ARM9
{opFreq = "150 MHz"}

<<hwBus>>
bus : Hibi_segment

<<hwProcessor>>
<<PePerformance>>
<<ep_allocated>>
cpu2 : ARM9
{opFreq = "120 MHz"}

<<hwProcessor>>
<<PePerformance>>
<<ep_allocated>>
cpu3 : ARM9
{opFreq = "120 MHz"}
Mapping application onto platform
Tool framework for evaluation

UML2 performance model

Model parser

Back-annotator

XML system model

SystemC simulation with Transaction Generator

Performance results

Design-space exploration tool

Execution monitor
Execution monitor

- Visualizes the simulation
- Useful for spotting trends in execution
Conclusions
Conclusions

- Modeling allows validation of system using high level of abstraction prior to implementation
  - Time and effort is saved
- Good model is abstract, understandable, accurate, predictable, inexpensive
- MoCs can be categorized according to their
  - Notion of time
  - Communication semantics