SoC Modeling

Ming-Hwa Wang, Ph.D.
COEN 207 SoC (System-on-Chip) Verification
Department of Computer Engineering
Santa Clara University

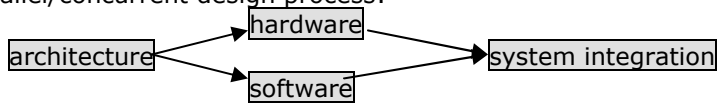
Topics

- advantages of modeling
- what is modeling
- cost of modeling
- languages for modeling productivity
- model creation and verification
- simulation technology
- SoC modeling execution

Advantages Of Modeling

Why Modeling?

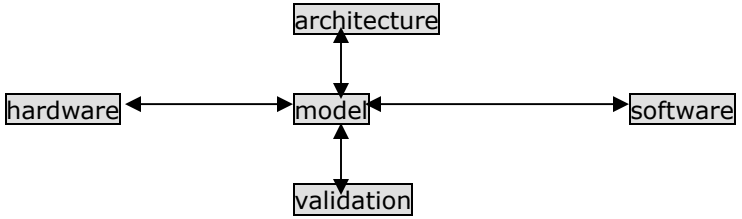
- time to market
  - conventional iterative serial design process:
    - architecture → hardware → software → system integration
  - parallel/concurrent design process:



  - a golden model for both hardware and software designs
  - software development can start much earlier in the design cycle, reduce time to market
  - fast turn around time for changes than RTL
- performance is the key as complexity grows exponentially

model	20-1000 mips
FPGA	20-200M cps
hardware accelerator	1M cps
C executables translated from RTL	1-10K cps
RTL	20-100 cps
Gate	1 cps

- very high reusability across projects
  - infrastructure
  - methodologies
  - models
  - tools
- use simulation, debugging, verification, co-simulation, etc. to help ensure the SoC works with 1st time success (no re-spin or reduce number of re-spins to very minimum)
- served as bridges among architecture, HW design/verification, SW development, and validation



- provides a consistent/uniform environment for modeling/simulation efforts
- TLM modeling can ease verification
- do pre-silicon validation before chip is back

What Is Modeling

Who Are the Customers?

- internal customers
  - hardware design - served as a golden model
  - hardware verification
  - software driver testing/integration
  - software applications
  - validation
    - L0 validation for JTAG (model not needed)
    - L1 tests on U-boot like environment without socket
    - L2/L3 flow testing on Linux like environment with socket (need drivers)
- external customers
  - functional model to run applications
  - cycle accurate model to debug applications

Models for Different Levels

- architecture
  - performance modeling
  - architecture exploration and trade-off
    - throughput, delay, congestion, buffer size, etc.
  - hardware/software partitioning
- software
  - algorithmic model - functional/behavior
  - transaction level modeling (TLM)
    - programmer's view (PV) model - register-based, bus generic, untimed
    - programmer's view model with timing (PV/T) - bus architecture with protocol, timing approximate
    - cycle-accurate model
- hardware
  - RTL: cycle-, bit-, and pin-accurate model

Transaction Level Modeling (TLM)

- bus function model (BFM)

    - traditional verification uses testbench to generate test vector and then test against the golden model
    - using a BFM provides an efficient means of including bus transactions in simulation instead of test vectors or stimuli
- TLM
    - a transaction is a quantum of activity that occurs in a system
    - TLM shifts upward in modeling abstraction w/o accompanied by a corresponding, automated path back down to the lower level
    - models communication mechanism (buses, FIFOs) are channels, by calling interface functions of these channel models, which encapsulate low-level details (pins) of the information exchange
    - transactions typically have a specific starting time and ending time
    - use of multiple inheritance to provide the flexibility, reuse, and protection
- TLM vs. RTL summary
    - faster to write, faster to simulate, less code, pure software - because a higher level of abstraction is used to describe the system
    - however - TLM means "giving up timing detail/accuracy" and is generally non-synthesizable
    - solution: integrated TLM + RTL flow

RTL-Dependent Modeling

- emulation and h/w accelerator
    - easy to attach validation devices by using speed bridges
    - run time ok
    - need modify RTL – synthesizable RTL only
    - one user at a time, need both hw and sw engineers work together
    - not intended for interactive debugging
    - single point of failure, expensive (cost vs. performance)
    - h/w accelerator is parallel processing, good for small design and speedup limited by the testbench (Amdahl's law)
    - emulation is FPGA based, and good for big design
- C executable translated from RTL – Carbon VSP or Tenison (now ARC)
    - easier to do, minimum model verification required
    - cycle accurate
    - need modify RTL – synthesizable RTL only
    - run time is slow
- co-simulation
    - very accurate model
    - need process support package (PSP), need modify RTL
    - difficult to debug
    - run time is slow even with back door memory access

RTL-Independent Modeling

- Virtutech
    - post-silicon model, functional model, no transactor needed
    - run-time speed is very fast
    - not yet productive/mature for model development by users
    - extensive verification is required (match specs? match RTL?)
    - difficult to do co-simulation and difficult to attach validation devices
- CoWare
    - automatic SystemC wrapper generation for RTL blocks or RTL wrapper generation for SystemC blocks
    - drag and drop blocks and connected by adding wires using GUI
    - hierarchical composition for the design
    - automatic makefile generation to build the whole design
    - run-time speed is fast
    - support both functional models and cycle accurate models
    - support co-simulation
    - not good in convert existing design into the flow
- Synopsys Innovator
- VaST
- ARM
    - maybe the best environment, but ARM processor only

Example Modeling Comparisons

	PV model	PVT model	emulation	C executable	co-simulation
RTL-dependent	no	no	yes	yes	yes
customers want	yes	yes	no	no	no
speed	very fast	fast	ok	slow	slow
model verification	need but difficult	need	no need	no need	no need
attach device	no and difficult	no	yes	no	no
interactive debug	easy	easy	waste	easy	difficult
model accuracy	functional	both	cycle accurate	cycle accurate	cycle accurate
transactors	no need	need	no need	need	need
resource requirement	high	high	low	low	low
model type	post-silicon	Post-silicon	pre-silicon	pre-silicon	pre-silicon
sequential or concurrent	concurrent	concurrent	sequential	sequential	sequential
hw/sw coop debug	no	no	yes	no	no
tool maturity	low	depends	high	high	high
single point of failure	no	no	yes	no	no
cost	expensive	expensive	expensive	reasonable	reasonable

Cost Of Modeling

Modeling Is Expensive

- cost/benefit trade-off
  - benefits of modeling is directly proportional to the availability and quality of the model - simulation is only as good as its models
- efforts needed – similar to RTL design, only more productive
  - fully understand the specs
  - code the model
  - profiling and tuning
  - test/verify correctness
- modeling need expert domain knowledge from architecture and coding skill to build the model
  - difficult to find people good in both areas
  - cooperating efforts needed
- productivity tools can help speed up development
  - language and compiler techniques
  - multi-domain simulation environment with GUI

Languages For Modeling Productivity

Languages

- high-level languages
- hardware description languages (HDL)
- electronic/executable system-level languages (ESL)
- verification description languages (VDL)
- architecture description languages (ADL)
- model description languages (MDL)

High-Level Languages

- high-level languages (C/C++)
- scripting languages (Perl, Python, etc.)
- for functional/behavior model only, no timing/clock

Hardware Description Languages (HDL)

- HDL - Verilog, VHDL
- multi-level description languages
  - behavior/functional - for high-level modeling and testbenches
  - register transition level (RTL) - for logic design
  - gate level
  - switch level - for circuit design
- synthesis flow in ASIC: translate RTL into gate/switch design by compiler technology
- custom design flow - for high performance design
  - need both logic design and circuit design
  - need equivalence checking (beware of exponential explosion)

Electronic/Executable System-Level Languages (ESL)

- can use open source SystemC kernel (http://www.systemc.org) and gcc instead of licensed hardware simulator
- executable specifications: SystemC with master/slave and TLM libraries
- single language for both model and HDL: based on high level language C++, grows downward by adding clocks, parallel execution scheduler and sensitivity list to mimic the hardware simulation
- multi-level description languages (though low-levels not recommended)
  - untimed/timed functional level
  - bus cycle accurate (BCA) level
  - cycle accurate (CA) level
  - register transition level (RTL)
  - gate level
- served as a glue mechanism for integrating different models
- code/class documentation can be automatically generated via doxygen
- most commercial synthesis tools support SystemC (though not recommended)

Verification Description Languages (VDL)

- VDL
  - SystemVerilog – based on Verilog and grows upward to support system leverl modeling, emphasis on verification
  - SystemC Verification (SCV)
  - Vera – design verification language
  - Specman – coverage-driven verification
- Features
  - constrained/biased random test generation
  - function coverage and coverage-driven verification
    - line coverage, block coverage, or segment coverage
    - branch coverage
    - expression coverage: if all possible legal Boolean values of the expression is reached
    - toggle coverage: which bits in RTL are toggled – for power analysis
    - FSM coverage: if all states and possible transitions are reached
  - assertion language/aspect

Architecture Description Languages (ADL)

- ADL for application-specific processors
  - Lisatech from CoWare
  - nML from Target
- processor architecture description and exploration
- automatically generate templates and manually insert implementation code or automatically generate implementation code
- tool generation from ADL
  - assembler
  - ISS
  - linker/loader
  - debugger

- compiler
- RTL

Model Description Languages (MDL)

- MDL
  - compiled code - Virtutech DML
  - graphics/spreadsheet - Escalate
  - flowchart - Synopsys Innovator
  - block diagrams and templates – Ptolemy
  - hierarchical drag and drop - CoWare
  - math formula/equation - Matlab
- automatically translate MDL into model
  - more productive
  - encapsulated/standalone or environment-independent models

Model Creation And Verification

Model Creation

- manually coding
- generated by DML
- converted from other model
- composition from other models
  - manually coding the interconnections
  - connection template automatically generated
  - drag and drop using GUI tools with wrappers
  - mix and match models with different levels and formats
- hierarchical composition and viewing
  - easy to understand the design and debugging

Model Conversion

- compiler technology - translate from high-level models to lower level models
  - synthesis - translate from RTL to gate/switch, e.g., Design Compiler
  - behavior synthesis - translate from functional/behavior model into RTL
    - Synopsys Behavior Compiler (BC)
    - SystemC to RTL compiler from Forte, Synfora, etc., provides signal environment from architecture modeling to synthesis, good for algorithmic design only
- translator technology - translate one model to a different model with same level
  - gain execution speed (w/ or w/o optimization) and easy to integrate
    - Carbon VSP Compiler
    - Tenison
  - productivity tools: translate mixture of models into HDL
    - Vperl: translates mixture of Verilog and Perl into pure Verilog
- dis-compiler technology - translate from lower level model to higher level
  - reverse engineering??

Model Verification

- model checking
  - theorem proving
  - static/dynamic property checking
  - pre-/post-condition
  - assertion
  - automatically provide counter examples for debugging
- simulation technology
  - software simulation
  - hardware simulation
  - hardware-assisted acceleration (e.g., Palladium - expensive)
  - FPGA (need partitioning, limited visibility, slow setups/compiling, synthesizable RTL, little support)
  - symbolic simulation
  - co-simulation
- equivalence checking
  - cycle by cycle simulation and comparison
  - formal verification

Simulation Technology

Models of Computation and Simulation Technologies

- communicating sequential processes (CSP)
- continuous time (CT)
- discrete event (DE) *
- distributed discrete event (DDE)
- discrete time (DT)
- finite state machine (FSM), hierarchical FSM *
- process network (PN)
- synchronous date flow (SDF) *
- dynamic data flow (DDF) *
- synchronous reactive (SR) *
- rendezvous models (RM) *
- timed multitasking (TM)
- genetic or automatic programming

Simulation Environment

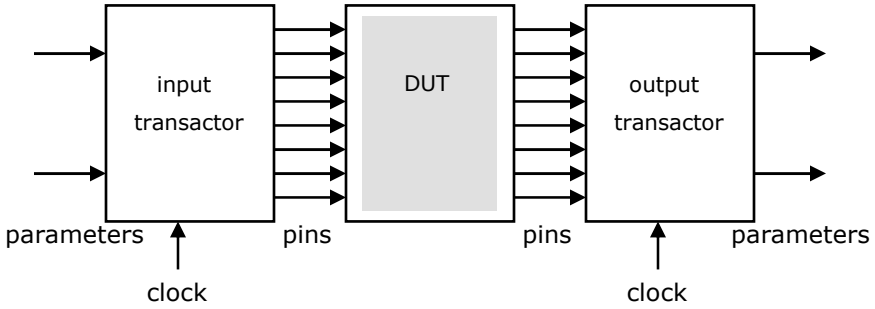
- modeling frame work
  - user interfaces or GUIs
  - kernel: event scheduling
  - kernel functionalities - debug, sim control, check
  - custom/std-based modeling languages
  - libraries (components, math)
  - glue/compilation/sim/regression automation
  - expression parser (e.g., tcl, custom)
  - result/data display
- architectural models, carbonized models, VIPs or peripheral BFM, 3rd party block/device/peripheral models

Separation of Simulation Environment and Models

- simulation environment can get from commercial tools or open source
- models can get from vendors independent of environment or be developed in-house
  - preferred order: IP vendor's models, other vendor's models, in-house
- separation of simulation environment and models to ship to customers
- models and IPs
  - buy from or contract to vendors - more robust
  - developed in-house
- advantages of separation
  - model portable across different tools
  - higher bargain power with vendors
- disadvantages
  - compatibility issues and tools specific features/performance

Co-Simulation

- two simulators: hardware simulator and software simulator
- communication between HW & SW simulators - socket, rpc, PLI, VPI, DPI/DKI
- transactors between high-level arguments and pin connections with timing
  - use verification IP (VIPs)
  - pin- and cycle-accurate: high level parameters vs. low level pins



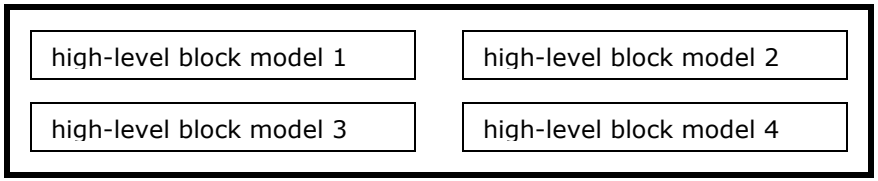
  - bit-accuracy is necessary (or use approximation)
- performance improvement
  - backdoor memory access model
- commercial co-simulation tools
  - Cadence - ncsim
  - Synopsys - vcs
  - Mentor Graphics - Questa + Seamless

A mix and Match Environment

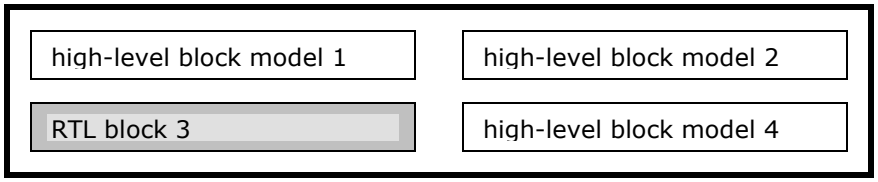
- mix of high-level and low-level models with different formats
- most block models are high-level, few models in low-level for accuracy without sacrifice much speed
- dynamic change combinations of high/low level block models for speed/accuracy requirements and configure/build whole SoC
- accommodate different format models by using wrappers or transctors
- good for unit test (on block level) and system test (on SoC level)
- using co-simulation to verify model and SoC for consistency

Example Mix and Match Models

- software model: e.g., all high-level models



- hybrid model: e.g., replace one block model by RTL



SoC Modeling Execution

Scheduling

- criteria definition (budget, goals/values, resource/time, domain expertise)
- specs collection and analysis for a project
- top level model type determination
- methodology evaluation and prototyping
  - automatic simulation environment generation
- vendor model evaluation/validation - on-going
- in-house model development/verification
- integration with SW platform
- phased development and deployment
  - minimum feasible subset
  - internal and then external customers

Execution plan

- basic framework
- ISS, cache, and memory
- system bus modeling
- models developed and tested independent of the project
  - iterative integration and testing
- integration and testing
  - modeling integration
  - SW integration

- validation and instrument attachment

Top Level Model Type

- requirements
  - easy to ship to internal/external customers
  - user friendly and easy to use/debug
  - can support mix and match environment
  - embedded Linux OS
  - ability to integrate into customer's environment
- language choice for top level model - SystemC/C++/C
- simulation control and scripting environment
  - an environment to control interaction and simulation control
  - control and scripting language choice - Tcl/Python

Methodology and Simulation Environment

- commercial tools - Virtutech, VaST, Synopsys, CoWare, Arm, Target, etc.
- free-domain open source tools - Giotto/Ptolemy/Metropolis, Esterel
- OSCI

Need to Have Features

	Synopsys	Virtutech	Giotto or Ptolemy	Esterel	VaST	CoWare
ISS	Arm	PPC, Arm			Arm	PPC, Arm
IPs	Standalone blocks	proprietary	sub blocks	sub blocks	From 3rd party	
model creation w/ GUI	yes	model, no GUI	yes	yes	XML	yes
speed	fast	Very fast	fast	fast	fast	fast
debugger	yes	GDB	GDB	GDB	yes	yes
boot Linux OS		yes	TinyOS			
accuracy	PV/PVT	PV	PV/PVT	PV/PVT	PV/PVT	PV/PVT
support	yes	yes	open source	open source	yes	yes

Nice to Have Features

	Synopsys	Virtutech	Giotto or Ptolemy	Esterel	VaST	CoWare
SystemC or Carbon integration	yes	no	Mirablilis*		yes	yes
co-simulation	yes	no	yes	yes	yes	yes
support modeling			UML	XML	SML	XML
source code gen	yes	yes	yes	yes	no	yes
cost	800K	376K	free	free	760K	800K
maturity/years	8	13	20	25	10	12

Automated Simulation Environment and Model Generation

- mix and match may need different simulation environments for different applications
- build system to support automatic configuration and build of simulation combinations
- deliverables
  - an environment for SoC modeling and simulation
  - ability to deliver SoC model to be used in customer environments

Simulation Platform

- use evaluation board and connect SoC model using socket
- use ISS to replace evaluation board, connect to GDB

Instruction Set Simulator (ISS)

- requirements
  - speed
  - dual-/multi-core and cache coherency
  - external memory synchronization
  - cycle approximate or cycle accurate
- open source - IBM PEK, MAME, Qemu, PearPC, etc.

Memory Coherency

- memory models
  - inside ISS, or external memory models
  - memory map partitioning
  - backdoor memory access for efficiency
- cache/memory interactions
  - memory management
  - cache coherency between the cores

Bus Models

- abstract or high-level bus model
  - preferred - more efficient
  - connect low-level models to the bus with transactors
- pin-/cycle-accurate or low-level bus model
  - Carbon VHM model
  - connect high-level models to the bus with transactors/VIPs

Block/Device/Peripheral Models

- each block/device/peripheral may have multiple models with different levels/formats

- behavior/functional models in C/C++/SystemC
  - TLM models (PV, PV/T, cycle accurate) in SystemC/SystemVerilog
  - RTL models in Verilog/VHDL, or Carbonized VHM
- transactor/VIP for each block/device/peripheral
  - high-level models connect to pin-/cycle-accurate bus model
  - low-level models connect to abstract bus model
- models procured form vendors independent of the environment are preferred
- models to be developed in-house only if not available

Uniform Model Verification Environment

- an uniform model verification environment for unit (block/IP) and system (SoC) tests
- write directed tests based on compliance test suites
- wrap functional model with pin-/cycle-accurate transactors to test against proven VIPs - Denali used the same approach
- co-simulation to validate against RTL
- regression with automated test self-checking - Dejagnu
  - short or release regression
  - daily regression
  - weekly or full/long regression

Debugger and GUI

- support GDB
- support other vendor's debuggers
- multi-core debugging support
- SoC peripheral debugging support
- integrate the debugger to Eclipse programming environment

Instrument Attachment

- need instruments support commands and vector file generation
  - most well-known instrument companies (e.g., Agilent, Rhodes & Shwartz, etc.) support these, small instrument companies may not have these support