how big can you dream?™

cadence

Cadence Technical Analysis of System Verilog

DECEMBER 2002

Outline

- Criteria for Language Critique/Design
- Critique of Existing LRM/Donations
- Recommendations Moving Forward

Criteria - Goals of Language Extension[CG]

- Types of Changes
  - Higher level of design abstraction
  - Advanced Verification Methodologies
  - Encourage reusable design/verification
- Characteristics to Preserve
  - Refinement of Abstract -> RTL -> Gate
  - High-Performance Event-driven simulation semantics
  - Synthesizability of modeled components

Criteria – Retain Style Of Verilog 1364^{CV}
For Better or Worse – Verilog …

- Implicitly declares objects
- Allows use before declaration
- Passes parameters as copy-in, copy-out
- Does limited/no/deferred type checking
- Performs implicit type coercions (reg to wire assignment for instance)
- Permits/Encourages global scope and visibility (allowance of Out-Of-Module-References)

Criteria – Scalability/Interoperability[CS]

- Language should define basic building blocks
- Same concept in different language extensions should use the same building blocks
- Reuse of building blocks creates a scalable philosophy on how to extend the language
- Macro concepts specific to a tool or technology should be modeled with these building blocks, not added as a new language feature

Criteria - Backward Compatibility[CB]

- Consistency with Verilog 2001
- Tons of Verilog code no one ever wants to touch again must be syntactically legal without modification
- Keywords must be treated as VERY precious items and added only when absolutely necessary
- Semantics must be strictly compatible – old designs must continue to work
- Don’t add new ways of doing things that are already possible

Criteria – Decomposition/Refinement[CD]
Issue Peculiar to HDL's

- Devices are remodeled/refined at many levels
  - Detail is added for refinement of implementation
  - Detail is removed for model abstraction and performance
- Interface to the device should remain as constant/reusable as possible
- Testbenches should be reusable across these modeling levels
- Due to SOC/block-based design at some level a system-level model is indistinguishable from a testbench

Outline

- Criteria for Language Critique/Design
- ➢Critique of Existing LRM/Donations
- Recommendations Moving Forward

Data Types - Goals[CG]

- Expand modeling abstraction
  - 2-state types, enums, records, unions, etc.
- Provide more robust simple interface to ‘C’
  - Short, char, int, real, …..
- Permit composite structural interconnect
  - Pass something other than 4-state bits on wires
  - Define a whole set of ports required for a device or protocol
- Create dynamically allocated structures in test benches

Data Types - Logic

- After meaning was clarified in sv-bc, no new semantics are present -CB
- More general extension is to allow regs to be ports without a wire +CS
  - accomplishes the same thing
  - If combined with type extensions to regs, gives structural interconnect +CS
- Current restrictions on assignments from different scopes are a remote elaboration-time error -CD
  - makes re-use difficult
  - makes IP legal in one context but not another
  - author of module has no control, only users of module
- SDF Back-annotation to procedurally assigned interconnect is completely undefined to date -CD

Data Types - Composite Structures

- Three answers in three places -CS
  - SV 3.0 has allowed structs/unions except wires can not take on these types
  - Interfaces added to allow structural interconnect, behaviorally assigned
  - VeraLite classes extend composites way beyond current capabilities and language style
- Difference solutions for different language components prohibit decomposition and refinement -CD
  - As implementation moves from Interface to RTL, ports must be remodeled
  - Testbench for system-level description using interfaces can not be reused against implementation in RTL

Data Types - References/Pointers

- Requirements appear to be:CG
  - Interoperability w/ ‘C’
  - Pass by reference to System Verilog (or ‘C’)
  - Verification data structure creation (scoreboards, lists, hashes)
- Massive potential negative impact on simulation performanceCG
- If allowed, must be severely restricted w.r.t:CG
  - Sensitivity
  - Kinds of objects they can point to
  - Timing controls in tasks to which they have been passed

Data Types – Higher-level Structures (lists, queues, fifo, hashes, assoc. arrays, mailboxes, semaphores, …)

- Are these fundamental language constructs or should the language be modified so that these can be modeled? CG,CS
- Option 1 - Fundamental Constructs
  - New keywords, operators for each one -CB
  - Where does it end? -CS
    - List, Stack, Tree, Balanced Tree, Lattice, Graph
    - Statistics gathering may be required for verification (queue length, time empty, time full, etc.) (i.e. 0-in CheckerWare Library)
- Option 2 - References/Pointers
  - Functionality would have to be restricted CG
  - Users probably want implicit allocation/deallocation (new, garbage collect) +CV
  - May be separable into portion of the language explicitly for verification (mimic region like specify block but for verification) CG
- Cadence believes adding dynamic allocation in tightly controlled regions provides the fundamental building block for all higher-level structures +CV

New Features - $root

- Didn't we learn our lesson with compiler directives? -CV
  - (i.e. a lot of time being spent to make `timescale local)
- Incredible potential for non-determinism -CD
  - Global scope leads to name collisions
  - incorrect interpretation as IP is reused
  - File ordering could lead to incorrect, undetectable interpretation
- Top-level modules can accomplish the same thing +CS
  - Create a module with the global information +CV
  - Use an Out-of-module reference to get to it +CV
  - Possibly add an attribute indicating a module should be at the root +CS
  - If users insist, add something like a 'use' or 'with' statement to shorten syntax of reference to hierarchical names -CV

New Features - Interfaces
Lumps un-related concepts in one construct

- Structural interconnect with direction
  - Very incompletely defined (completely by example) -CV
  - Does not decompose well vis-à-vis SDF Backannotation -CD
  - Should be a general extension of ports and structs +CS
- Task-level abstraction
  - Most of this already exists in Verilog 2001 -CV
  - Good proposed extensions to explicitly export from a module +CD
  - Multiple users of the interface each need a unique driver. These are note created by the task calls making it unusable for refinement as specified -CD
- Parameterized-hierarchical reference +CD, +CV
  - Cool concept but is simply passing a reference to a module instance hierarchically
  - Adding module names as a form of parameter would solve same problem without new construct

New Features - Redundant Additions
Add no new functionality, just keywords

- Alias statement (just clarify semantics of feed through module) -CV
- Always_{ff, latch} add no semantic content -CV
  - should be attributes from IEEE on the always block +CS
  - Where does this end as new restrictions evolve -CS
  - Not a sustainable language extension (expands keywords linearly) -CS
- Unsized literal bit values (in 2001) -CV
- Constants (Verilog has them already) -CV
- “unique” and “priority” should be synthesis attributes -CS
- Iff –CV, -CS
- Nested Modules –CV, -CS

New Features – Functions/Tasks

- Function inout and out ports -CV
  - Previously disallowed intentionally
  - Allows functions to have side effects
  - Creates need for default type and direction of formal arguments
- Functions as statements -CV
  - already have tasks
  - creates need for the void type
- This brings in a feature from ‘C’ which does not have a separate task/function distinction into a language that does -CV

New Features - Verilog 2001 Conflicts

- 'static' keyword was rejected by IEEE -CV
  - Needed static variable can always be promoted to enclosing scope
  - IEEE spent a huge amount of time debating this and rejected it, why is it being revisited? A huge waste of energy.
- Always @(*) vs. always_comb –CV, -CS
  - Minor variation in semantics should be migrated to one construct
  - Separate event control from assignment semantics
- Variable initialization does not cause events -CV
- Contradiction on semantics of posedge/negedge of multi-bit objects -CV

C-Interfaces
Need for 3 interfaces identified

- Simple, fast access to ‘C’ (nice progress lately) +CG
  - Useful for algorithms written in ‘C’
  - Useful to perform things hard in Verilog (sockets, I/O)
  - Parameters are only access Verilog objects
- Handle-based interface to language constructs – VPI +CG
  - All new language extensions must have equivalent VPI +CV
  - This definition should not be left to later
  - Can actually clarify data-model of new constructs
- C++ hierarchy, interconnect, processes +CG, +CV
  - Any coordination with C++ models should be compatible with SystemC rather than invent new C++ classes for hierarchy and interconnect.

C-Interfaces - Coverage

- ‘C’-level access to coverage statements should be a VPI extension, not a new API -CV
- Are we modifying the language every time a verification methodology is used to process it? -CS
  - Application or methodology specific information can be added to a design via attributes +CV,+CS
  - Behavioral extensions can typically be done through $tasks() +CV,+CS
- Great opportunity for adjunct standard +CG
  - Standardize $task names
  - Standardize set of attributes

C-Interfaces – Assertion API

cadence

- How can this be done in any detail until the Assertion Committee results are known?
- If Assertions become a part of the core language then this entire mechanism should just be extensions to VPI to access the new language constructs. +CV,+CS
  - New properties for static traversal to find them
  - New values relating to assertion state
  - New callback types for interacting with them at runtime
- Existing requirements document is completely insufficient -CG
  - http://www.eda.org/sv-cc/requirements/assertions_requirements.doc

VeraLite Donation

- 3 documents in play are difficult to reconcile -CG
  - Original donation
  - Clarification document (and presentation)
  - Latest randomization/constraint proposal (donation?, approved?)
- Position as new language vs Verilog extension is even more unclear than status of SystemVerilog -CV
- At a minimum constraints and assertions need to be unified -CS

Committee Ping Pong
These need cross-committee resolution

- References, pointers – sv-cc, sv-ec
- Structures, classes, interfaces – sv-bc, sv-ec, sv-cc
- Assertions, API – sv-ac, sv-cc
- Enumeration Types – sv-ec, sv-bc

Outline

- Criteria for Language Critique/Design
- Critique of Existing LRM/Donations

➢Recommendations Moving Forward

Recommendations – Data Types
Fundamental Re-Evaluation Necessary

- New System Verilog 3.1 requirements have made System Verilog 3.0 decisions obsolete
  - More robust interaction with ‘C’ code
  - Possible addition of references/pointers
  - Structural interconnect
- Orthogonal treatment of object class and object type
  - Class (reg, wire, logic, var) vs Type (4-state, 2-state, enum, struct)
  - Would need some semantic restrictions (references)
  - Difficult (but not impossible) backward compatibility
- Required for interoperability of testbench, abstract design, RTL design, and gate-level implementation

Recommendations – Language Fundamentals

- For Better or Worse, Verilog …
  - Implicitly declares objects
  - Allows use before declaration
  - Does limited/no/deferred type checking
  - Performs implicit type coercions
  - Permits/Encourages global scope and visibility (allowance of Out-Of-Module-References)
- Are we going to change these fundamentals?
- Maybe we follow like ‘C’/C++ with a well-defined distinction for:
  - Object-Oriented extensions
  - Classes
  - Inheritance
  - Type Templates

Recommendations - Process

- Re-evaluate speed of standardization process vs. quality of resulting standard
- Restructure overlapping/entangled responsibilities of technical committees to resolve technical problems where global consideration is needed
- Take into consideration amount of rework at IEEE level and identify portions destined for IEEE
- Formal documentation on goals of extensions and language design style
- Need for revised LRM/Proposals with change bars
  - Document resolved SystemVerilog 3.0 issues
  - Incorporate “clarifications” to donations

Recommendations – Summary

- SystemVerilog is solving the right problems
  - Modeling at more abstract levels
  - Adding verification technologies
  - Encouraging reusable/robust design and verification
- Goal should be scalable additions
  - Limit keywords
  - Provide building blocks for abstract concepts
- Keep style of Verilog-1364 intact
  - Ensure backward compatibility
  - Extensions beyond this should be explicitly separate lexical scope or new languages

how big can you dream?™