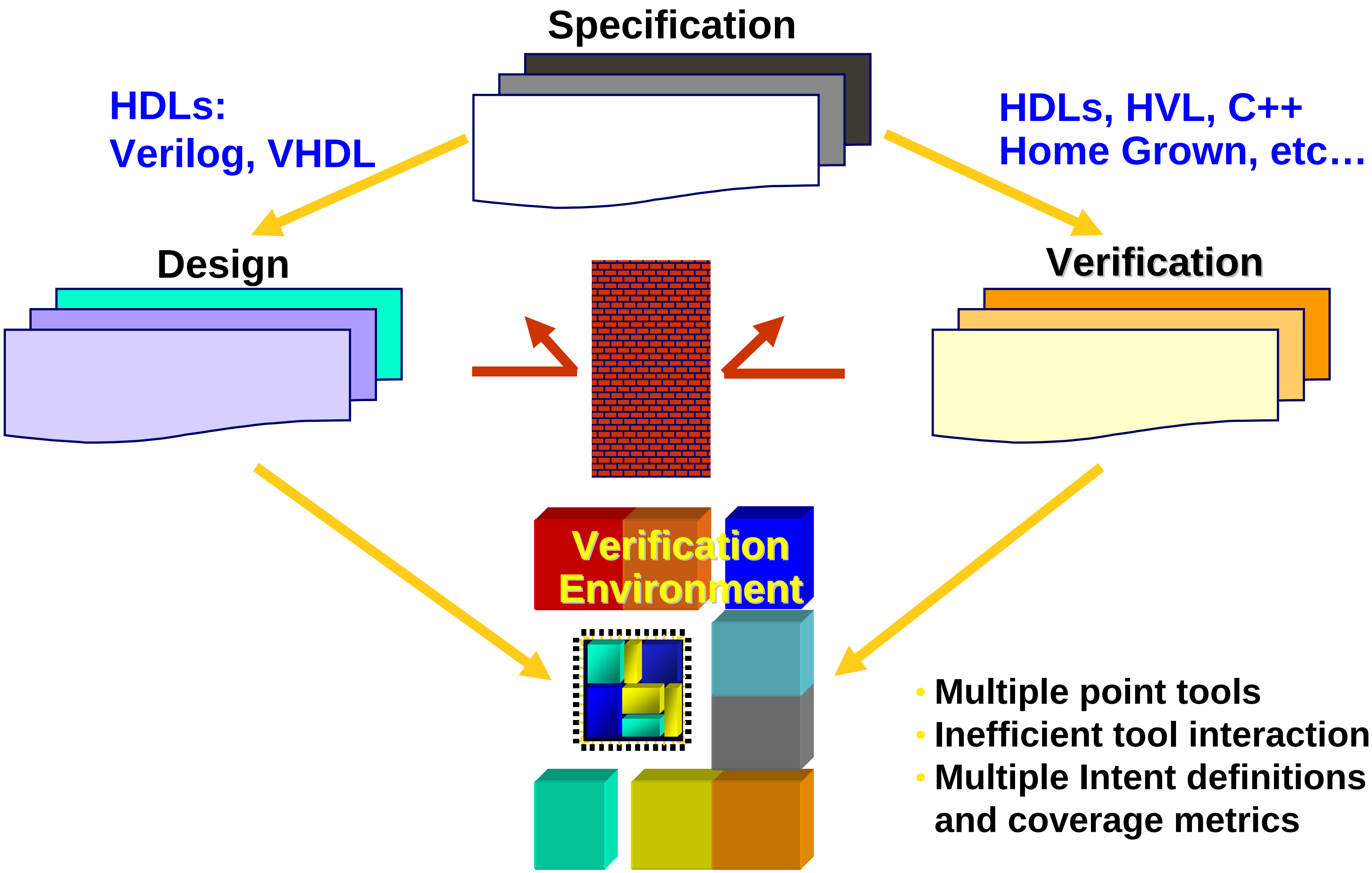
Using SystemVerilog for Design

Karen Pieper
R&D Manager, HDL Compiler
Co-Chair, SystemVerilog Design Committee

Overview

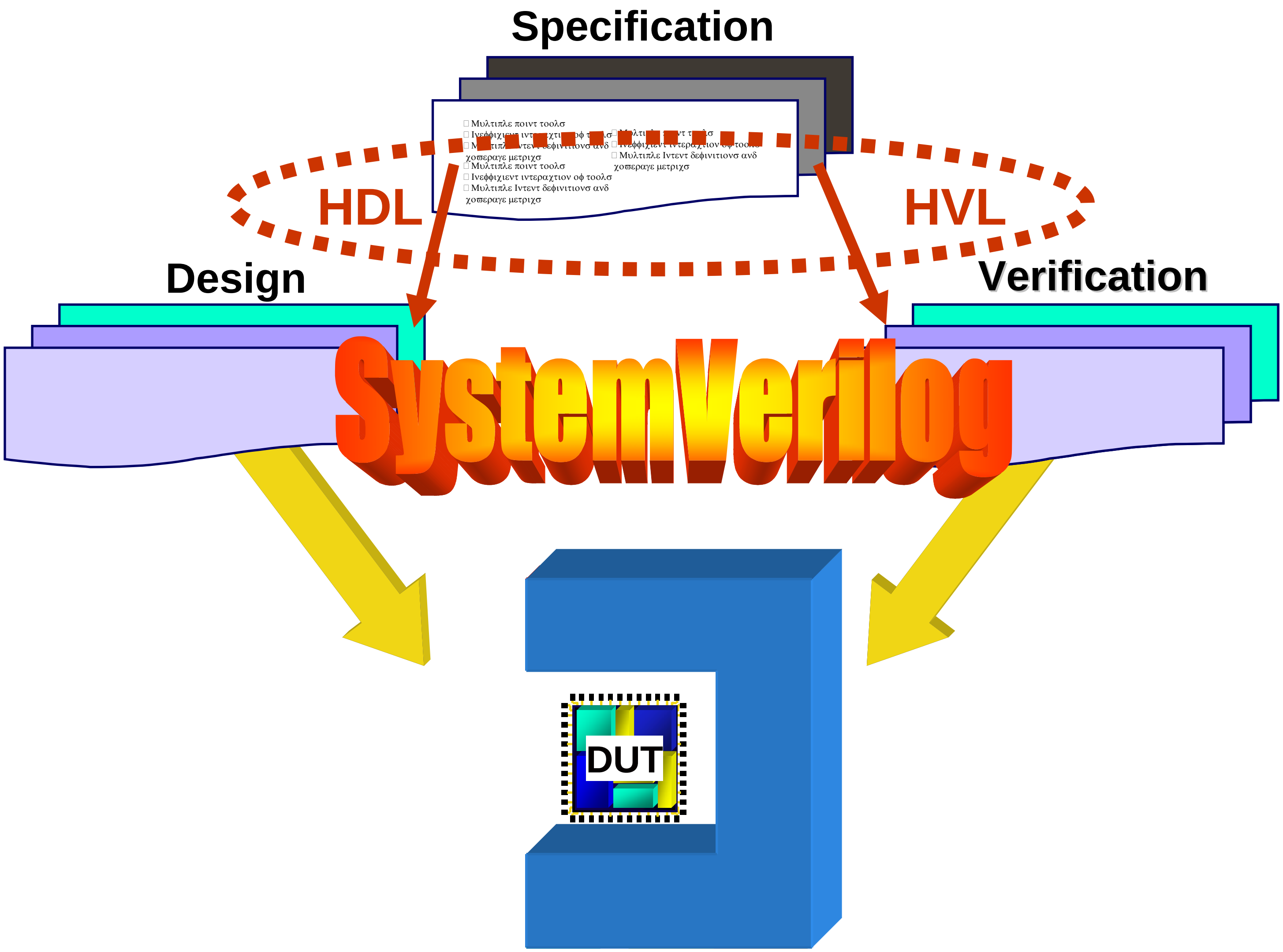
- What is SystemVerilog?
- Advantages for design
- Use model and synthesis results
- Food for thought
- Roadmap

RTL Design and Verification Today



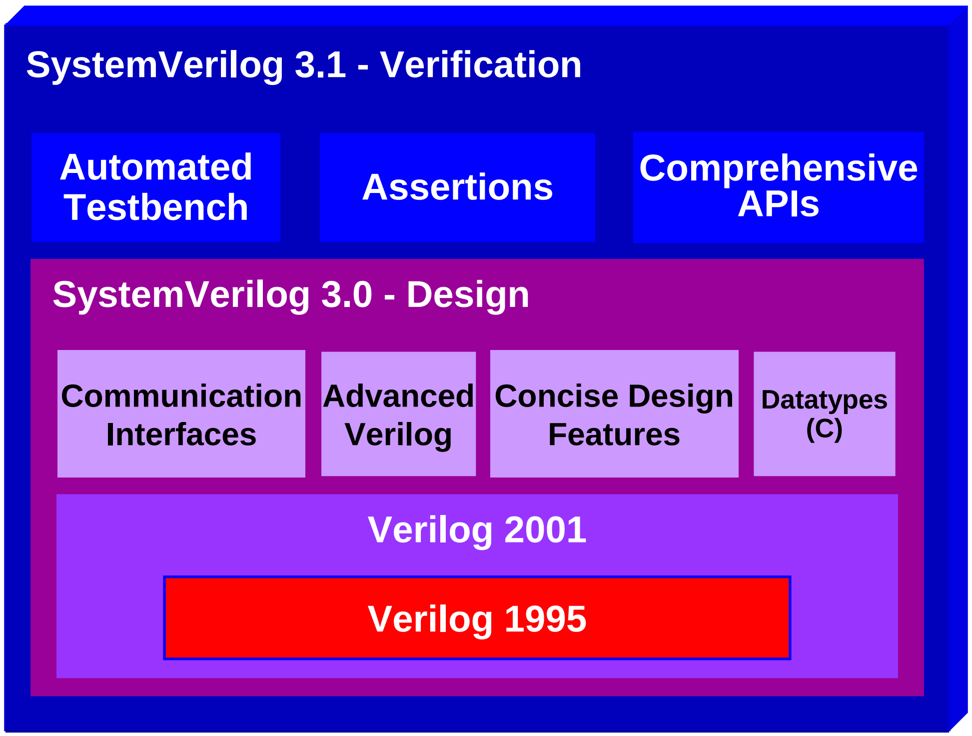
- Multiple point tools
- Inefficient tool interaction
- Multiple Intent definitions and coverage metrics

Unifying Design & Verification



What is SystemVerilog?

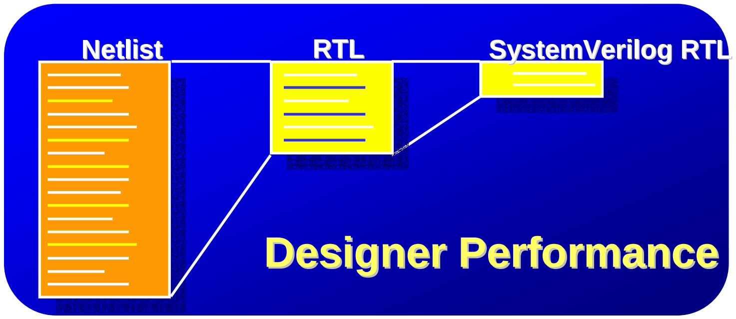
Unifies Design and Verification for SoCs



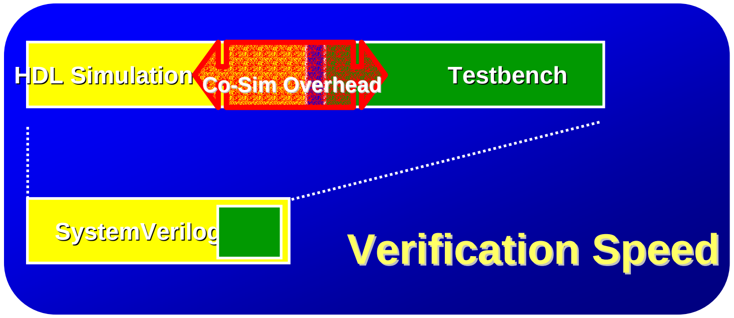
- SystemVerilog Benefits
  - Higher designer productivity
  - Faster and smarter verification
  - Evolves Verilog into first Hardware Description & Verification Language
- Accellera standard today
- Supported by multiple vendors
- Leading design teams are preparing for full-scale adoption

100% compatible with Verilog

SystemVerilog Benefits



- **100% Compatible with Verilog!!!**



- **2-5X Faster Verification**

- **Capture Design Intent with Single Point of Specification**

Design Extensions of Verilog

- Communications interfaces
  - Intended to describe communication among multiple modules
  - Collects all of the description of a bus in one location
- Enhanced Verilog
  - Specify design intent
  - Support transition from reg use to wire use without changing the declaration
  - Add full_case and parallel_case directives to the language
- Concise design features
  - Type as a parameter
  - C-style assignment operators +=, ++
  - Simplification of port connection on instantiation
- Datatypes (C)
  - Data abstraction is improved with structures, unions
  - 2-state and 4-state variables are available

Determining Synthesizability

- Design goal
  - Support common design methodologies
    - Multiple engineers synthesizing their own blocks independently
    - Use of formal verification techniques to ensure that RTL and gates represent the same design
- Requirements of the subset
  - Independent synthesis and verification of each module
  - Single “correct” interpretation of every design
  - Statically determined hardware

Advantages of SystemVerilog for Design

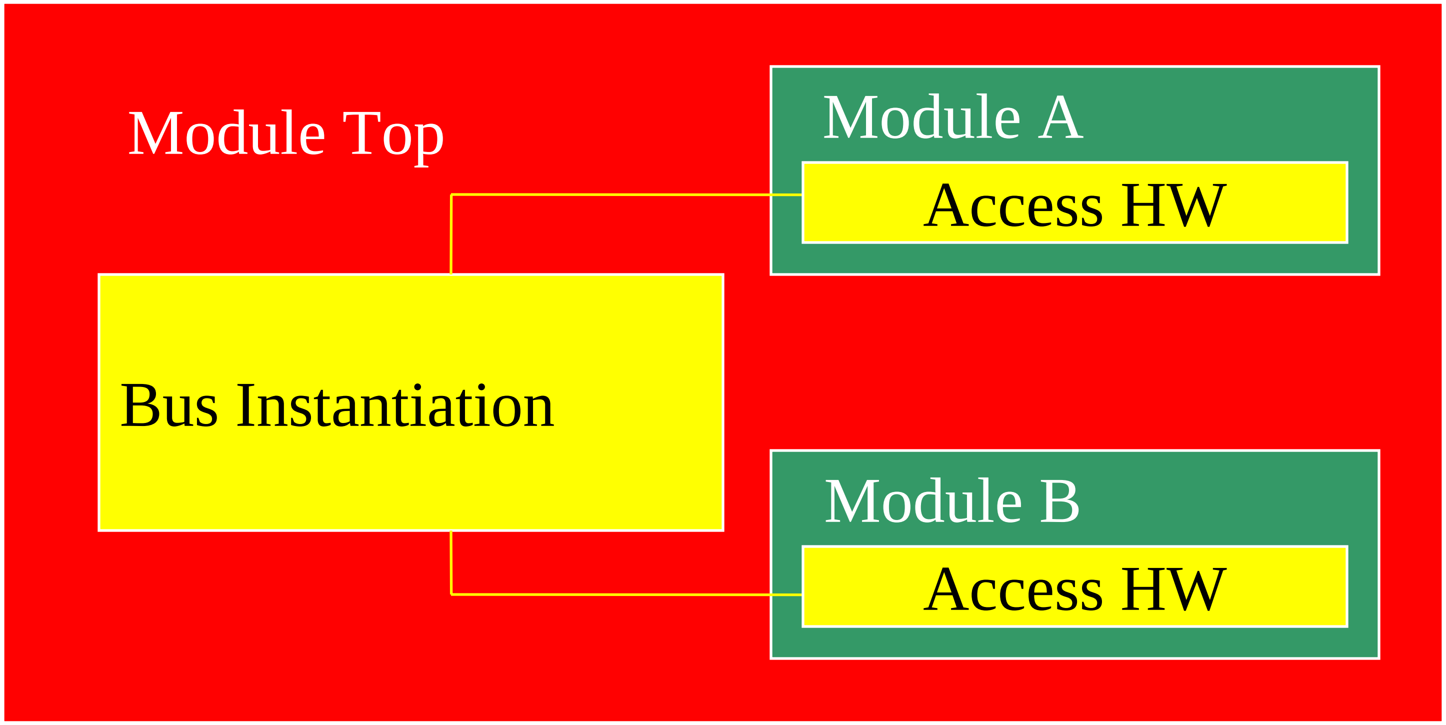
- Improved specification
  - Localization of functionality
  - Exact specification of intent
- Concise constructs for focused development
  - Line count correlates strongly with number of bugs
- Convenience in data abstraction
  - Correct data representation for many uses
- Unifying design and verification
  - Designs can specify requirements and guarantee they are met

Improved Specification

- Some issues in design today:
  - Adding a port to a bus, or changing the operation of a bus requires correct editing of all of the modules that touch that bus
    - ➢ “I don’t want to have to change all those places…”
  - Synthesis tools do what you say rather than what you want
    - ➢ “I did not mean to specify a latch there….”

SystemVerilog has a solution

No Localization of Functionality



- Descriptions are spread throughout the design
- All designers must know details of the access
- Any modifications must be made throughout the design

Localization of Functionality

Source code view:

- Module Top
  - A Instantiation
  - B Instantiation
  - Bus Instantiation

- Module A
  - Bus API Call

- Module B
  - Bus API Call

- Bus Description
  - Bus Instantion HW
  - Bus Access HW

- Descriptions are localized
- One designer knows details of access
- Modifications (and replacements) are made in one location
- Eases understanding, debugging, and reuse

Localization of Functionality

- Language features providing localization of functionality
  - Structures and user defined data types
  - Types as parameters
  - Interfaces
  - Global functions

Structures and User Defined Data Types

Verilog

```
module fifo (clk, rstp, din_src,
 din_dst, din_data,readp,writep,
 dout_src,dout_dst, dout_data,
 emptyp, fullp);
 input                clk;
 input                rstp;
 input [7:0]          din_src;
 input [7:0]          din_dst;
 input [31:0]         din_data;

 input                readp;
 input                writep;
 output [7:0]         dout_src;
 output [7:0]         dout_dst;
 output [31:0]        dout_data;

 output               emptyp;
 output               fullp;
.
.
```

SystemVerilog

Define Once

```
typedef struct {
        bit [7:0] src;
        bit [7:0] dst;
        bit [31:0] data;

} packet_t;
```

Use many times

```
module fifo (
 input                clk,
 input                rstp;
 input packet_t       din,
 input                readp;
 input                writep;
 output packet_t dout;
 output  logic        emptyp;
 output  logic        fullp
);
```

Easy to read, debug

Structures and User Defined Data Types

Verilog

```
module fifo (clk, rstp, din_src,
 din_dst, din_data2, din_data,readp,
 writep, dout_src, dout_dst,
 dout_data, dout_data2, emptyp,
 fullp);
 input                  clk;
 input                  rstp;
 input [7:0]            din_src;
 input [7:0]            din_dst;
 input [31:0]           din_data;
 input [31:0]           din_data2;
 input                  readp;
 input                  writep;
 output [7:0]           dout_src;
 output [7:0]           dout_dst;
 output [31:0]          dout_data;
 output [31:0]          dout_data2;
 output                 emptyp;
 output                 fullp;
.
- And referencing code below
```

SystemVerilog

```
typedef struct {
        bit [7:0] src;
        bit [7:0] dst;
        bit [31:0] data;
        bit [31:0] data2;
} packet_t;
```

Define Once

```
module fifo (
 input                  clk,
 input                  rstp,
 input packet_t         din,
 input                  readp,
 input                  writep,
 output packet_t        dout,
 output  logic          emptyp,
 output  logic          fullp
);
```

Use many times

Easy to read, debug, modify, reuse

Structure / Array Duality

- Unpacked structures have strong type checking
  - As in VHDL
- Packed structures have an array overlay

```
typedef struct packed {
    logic [3:0] a;
    logic [3:0] b;
} packed_t;

packed_t pt;
logic [7:0] array;

array[3:0] = pt.b;
array[3:0] = pt[3:0];
pt = pt;
pt = array;
```

```
typedef struct {
    logic [3:0] a;
    logic [3:0] b;
} unpacked_t;

unpacked_t pt;
logic [7:0] array;

array[3:0] = pt.b;    // ok
array[3:0] = pt[3:0]; // error
pt = pt;              // ok
pt = array;           // error
```

- Synthesis
  - Recommend unpacked structures for better area, timing

Types as Parameters

Fifo Example:

```
typedef struct {
        bit [7:0] src;
        bit [7:0] dst;
        bit [31:0] data;
        bit [31:0] data2;
} packet_t;
```

```
module fifo #(parameter type t = int)
(
 input                 clk,
 input                 rstp,
 input t               din,
 input                 readp,
 input                 writep,
 output t              dout,
 output  logic         emptyp,
 output  logic         fullp
);
```

- One module, many types

  fifo int_fifo (….);

  fifo packet_fifo #(packet_t) (…);

- Creates generic IP for reuse
- Address issues in one module
- Synthesis
  - Same as integer parameters

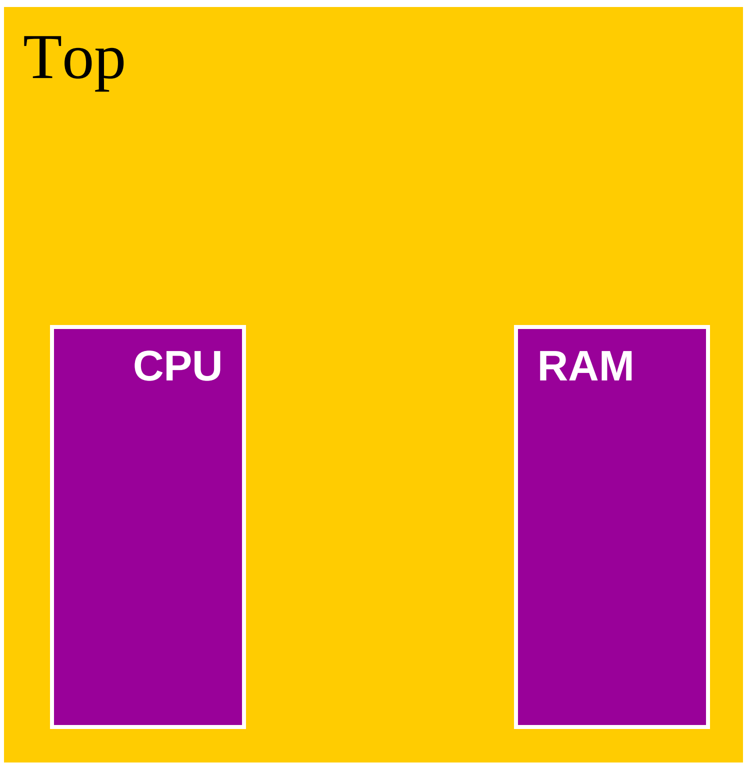
- VHDL has unconstrained arrays
  - Not arbitrary types
  - Avoid conversion functions

Interfaces

- Separate intermodule communication description and use
- Communication objects are described in one place
  - Localizes port updates
  - Allows an API description of bus interface to users
  - Simple to exchange bus style
  - Allows verification to be embedded supporting early, thorough testing
- Functionality does not exist in VHDL today

Simple Design

Synthesized Design:



Source Code:

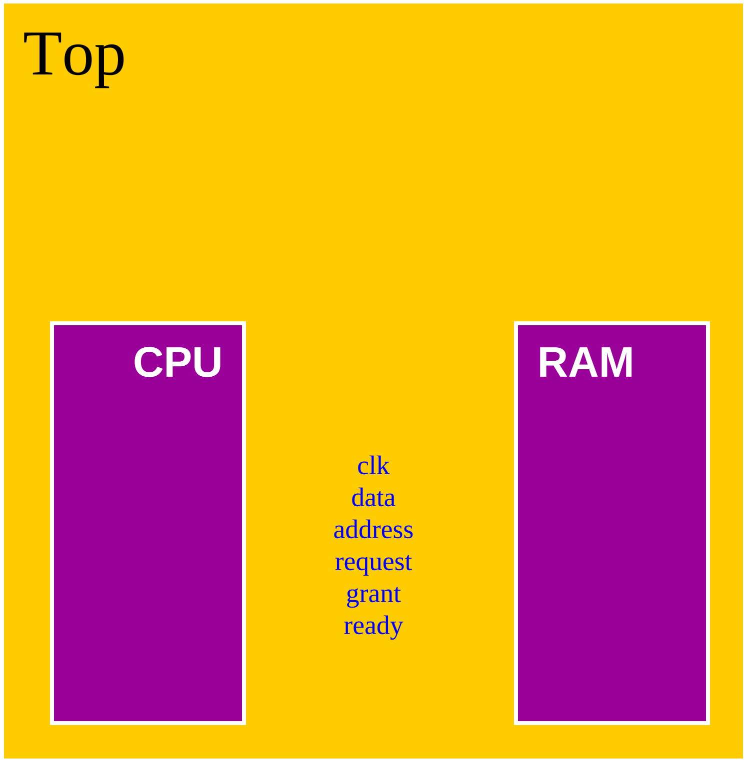
```
module CPU ();
  …
endmodule
```

```
module RAM ();
  …
endmodule
```

```
module Top;


  CPU CPU();
  RAM RAM();
endmodule
```

Interface Instantiation

Synthesized Design:




```
interface chip_bus ();
  wire        request, grant, ready;
  wire [47:0] address;
  wire [63:0] data;

endinterface
```

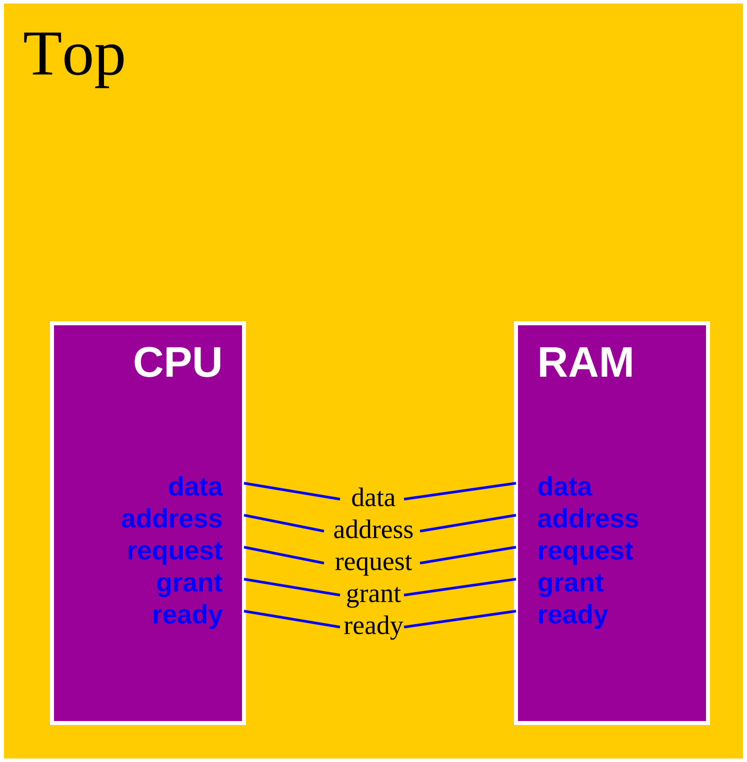
```
module CPU ();
  …
endmodule
```

```
module RAM ();
  …
endmodule
```

```
module Top;

  chip_bus a();
  CPU CPU();
  RAM RAM();
endmodule
```

Interface As A Port

Synthesized Design:




```
interface chip_bus ();
  wire        request, grant, ready;
  wire [47:0] address;
  wire [63:0] data;

endinterface
```

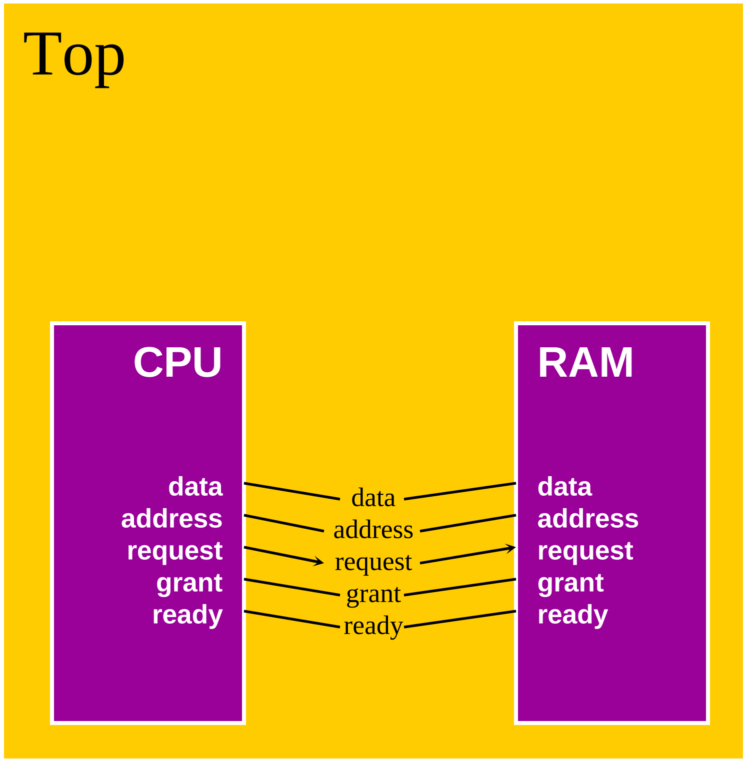
```
module CPU (chip_bus io);
  …
endmodule
```

```
module RAM (chip_bus pins);
  …
endmodule
```

```
module Top;

  chip_bus a();
  CPU CPU(a);
  RAM RAM(a);
endmodule
```

Interface Modports

Synthesized Design:




```
interface chip_bus ();
  wire        request, grant, ready;
  wire [47:0] address;
  wire [63:0] data;

  modport cpu (output request...);
  modport ram (input request...);

endinterface
```

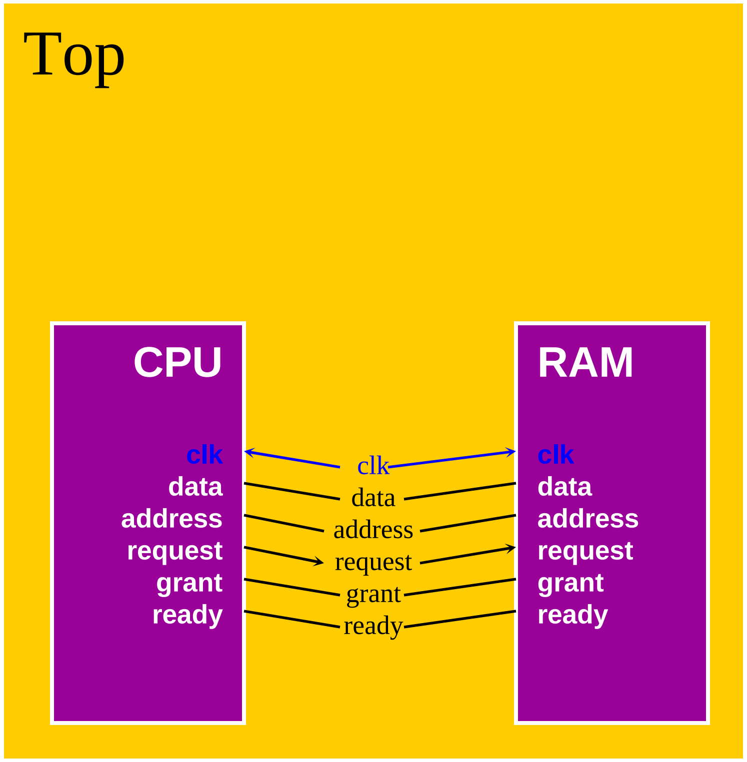
```
module CPU (chip_bus.cpu io);
  …
endmodule
```

```
module RAM (chip_bus.ram pins);
  …
endmodule
```

```
module Top;

  chip_bus a();
  CPU CPU(a.cpu);
  RAM RAM(a.ram);
endmodule
```

Interface With a Port

Synthesized Design:




```
interface chip_bus (input wire clk);
  wire        request, grant, ready;
  wire [47:0] address;
  wire [63:0] data;

  modport cpu (input clk, output request...);
  modport ram (input clk, input request...);

endinterface
```

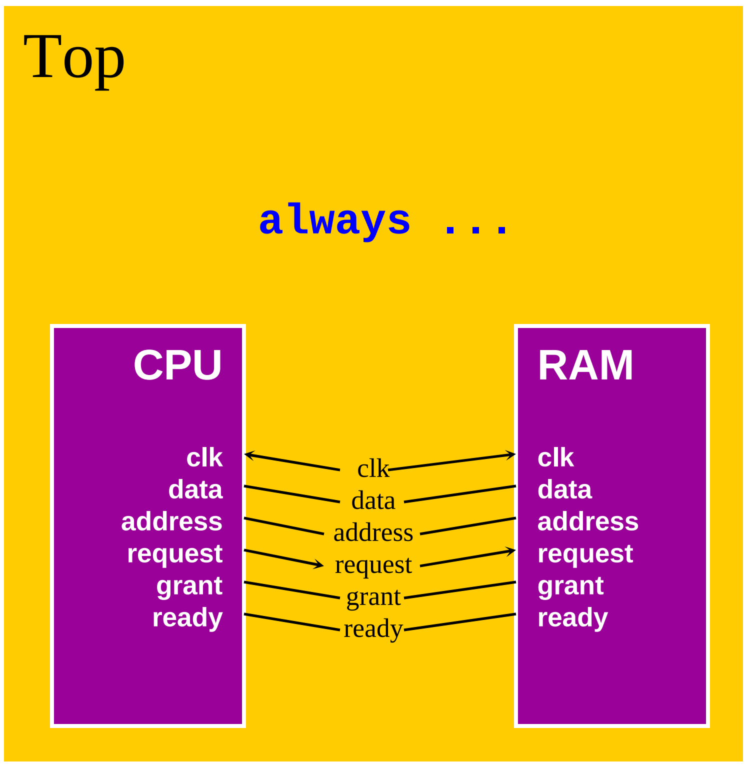
```
module CPU (chip_bus.cpu io);
  …
endmodule
```

```
module RAM (chip_bus.ram pins);
  …
endmodule
```

```
module Top;
  wire clk;
  chip_bus a(clk);
  CPU CPU(a.cpu);
  RAM RAM(a.ram);
endmodule
```

Interface Always Blocks

Synthesized Design:




```
interface chip_bus (input wire clk);
  wire        request, grant, ready;
  wire [47:0] address;
  wire [63:0] data;

  always …


  modport cpu (input clk, output request...);
  modport ram (input clk, input request...);

endinterface
```

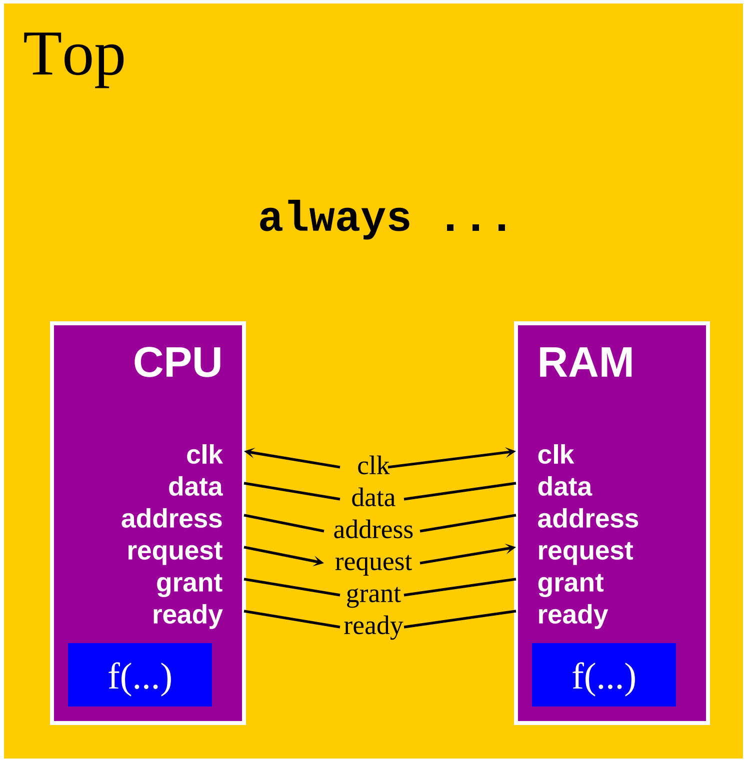
```
module CPU (chip_bus.cpu io);
  …
endmodule
```

```
module RAM (chip_bus.ram pins);
  …
endmodule
```

```
module Top;
  wire clk;
  chip_bus a(clk);
  CPU CPU(a.cpu);
  RAM RAM(a.ram);
endmodule
```

Interface Functions

Synthesized Design:




```
interface chip_bus (input wire clk);
  wire        request, grant, ready;
  wire [47:0] address;
  wire [63:0] data;

  always …

  function automatic f …

  modport cpu (…, import f);
  modport ram (…, import f);

endinterface
```

```
module CPU (chip_bus.cpu io);
  … io.f(…) …
endmodule
```

```
module RAM (chip_bus.ram pins);
  … pins.f(…) …
endmodule
```

```
module Top;
  wire clk;
  chip_bus a(clk);
  CPU CPU(a.cpu);
  RAM RAM(a.ram);
endmodule
```

Interfaces

- Interfaces localize the description of a bus
  - Localizes port updates
  - Allows an API description of bus interface to users
  - Simple to exchange bus style
  - Allows verification to be embedded
- Synthesis distributes hardware
  - Hardware is instantiated in the appropriate modules
  - Automatic tasks and functions must be used
  - Changing an interface requires resynthesizing all modules referring to that interface

Global Tasks, Functions

```
function automatic
    int f (input in, output out)
{
  ...
}

task automatic
    t (input in, output out)
{
...
}

module mod (input in, output out);
  ... f(in, out) ...
  ... t(in, out) ...
endmodule

module mod2 (input in, output out);
  ... f(in, out) ...
  ... t(in, out) ...
endmodule
```

- One function, many uses
- Creates generic IP for reuse
- Address issues in one place

- Synthesis
  - Inlined at callsite
  - Automatic required

- SystemVerilog 3.1a
  - Packages are introduced

Advantages of SystemVerilog for Design

- Improved specification
  - Localization of functionality
  - Exact specification of intent
- Concise constructs for focused development
  - Line count correlates strongly with number of bugs
- Convenience in data abstraction
  - Correct data representation for many uses
- Unifying design and verification
  - Designs can specify requirements and guarantee they are met

Exact Specification of Intent

- Synthesis can infer latches where they weren't intended
- Incomplete description of activation creates issues
- Tools can inform of problems if they know the intent
- Results
  - Reduce synthesis/simulation mismatch
  - Bugs found earlier
  - Area (possibly timing) improves because unnecessary hardware is not created

Always_* Forms

- Always_comb -- for combinational logic
  - Simulation activation better matches synthesis activation
  - Warning if latches or flip-flops are inferred
- Always_latch -- for latch logic
  - Simulation activation better matches synthesis activation
  - Warning if a latch is not present
- Always_ff @(exp) -- for flip-flop logic
  - Limits always block to one activation
  - Warning if a flip-flop is not present

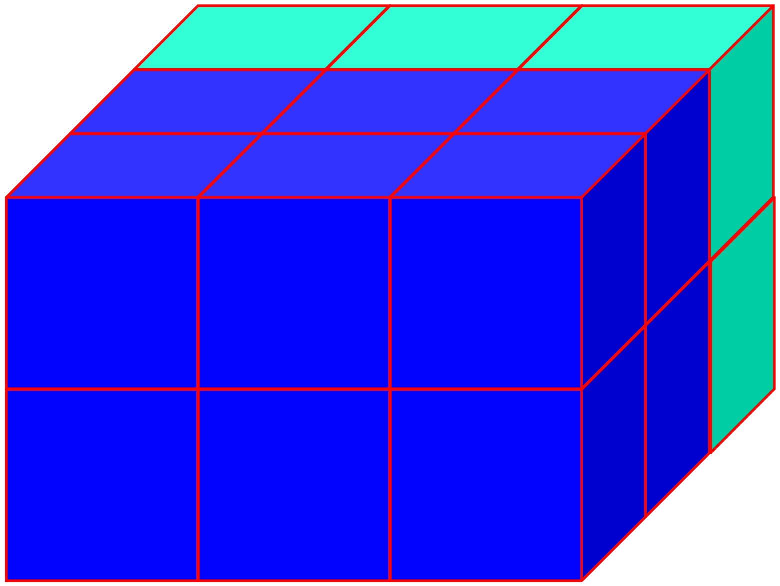
```
// and gate
always_comb
    o = a & b;


// latch
always_latch
    if (reset) then
       l <= 0;
    else
       l <= data;


// flip-flop
always_ff @(posedge clk,
            posedge rst)
    if (reset) then
       ff <= 0;
    else
       ff <= data;
```

SNUG EUROPE 2004

Multi-dimensional Array Slices

- Array slices allow more than the last dimension of the array to be accessed

```
reg [2:0][1:0][2:0] a;
a[1:0] = ….;
```



- Unlike VHDL, in Verilog 2001, slices had to be created using for loops
- Latches cannot be inferred by accident

Advantages of SystemVerilog for Design

- Improved specification
  - Localization of functionality
  - Exact specification of intent
- Concise constructs for focused development
  - Line count correlates strongly with number of bugs
- Convenience in data abstraction
  - Correct data representation for many uses
- Unifying design and verification
  - Designs can specify requirements and guarantee they are met

Concise Constructs for Focused Design

- Fewer lines of code usually means fewer bugs
- Less code requires less time to write and debug

- Language features improving conciseness
  - Described earlier
    - Structures, interfaces, global tasks and functions
    - Multi-dimensional array slices
  - Port connections
  - Return, break, continue, do … while
  - Assignment operators

Port connections

```
mod mod (.out(out), .in1(in1), .in2(in2));
```

- .name port expansion

```
mod mod (.out, .in1, .in2);
```

  - Synthesis is simple expansion
- .* port expansion

```
mod mod (.*);
mod mod (.*, .in1(a));
```

  - Synthesis requires analysis of instantiated module
  - Implicit wires are not created
  - Type checking is much stricter

C Control Flow

- *Return, break, continue*
  - Interpreted as you would expect in C

- *Do…while*
  - Interpreted as you would expect in C
  - Synthesis
    - Iterations must be statically determinable

Assignment Operators

- New operators

  +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=, <<<=, >>>=, ++, --

  `A[f(x)] += 1` becomes `tmp = f(x); A[tmp] = A[tmp] + 1`

- Assignment as an expression
  – Convenient, but not recommended for synthesis

Advantages of SystemVerilog for Design

- Improved specification
  - Localization of functionality
  - Exact specification of intent
- Concise constructs for focused development
  - Line count correlates strongly with number of bugs
- Convenience in data abstraction
  - Correct data representation for many uses
- Unifying design and verification
  - Designs can specify requirements and guarantee they are met

Convenience in Data Abstraction

- Correct data representation for many uses
  - Earlier
    - Structure / Array duality
  - 2-state and 4-state variables
  - Easing the reg / wire duality
  - Multiple type interpretation of the same bits
  - System functions for type analysis
- Simplification of coding process
- Easier understanding

2-state and 4-state variables

- Verilog can represent 4-state variables: 0, 1, x, z
- SystemVerilog
  - A new 4-state variable type: logic
  - A new 2-state variable types: bit, byte, int, short
    - Ease connection to C
- x, z assignments to bit variables become 0
- Synthesis
  - 4-state are preferred
  - 2-state variables require strict bounds checking
    - Simulation/synthesis mismatch or timing/area penalty
  - Typedef can be used to allow exchange of representation

Easing the reg / wire Duality

- Moving from RTL to an instance based description
- Replacing regs with wires
- Bit, logic, and reg types can be assigned as regs or driven by a single instantiation

	Verilog	SystemVerilog

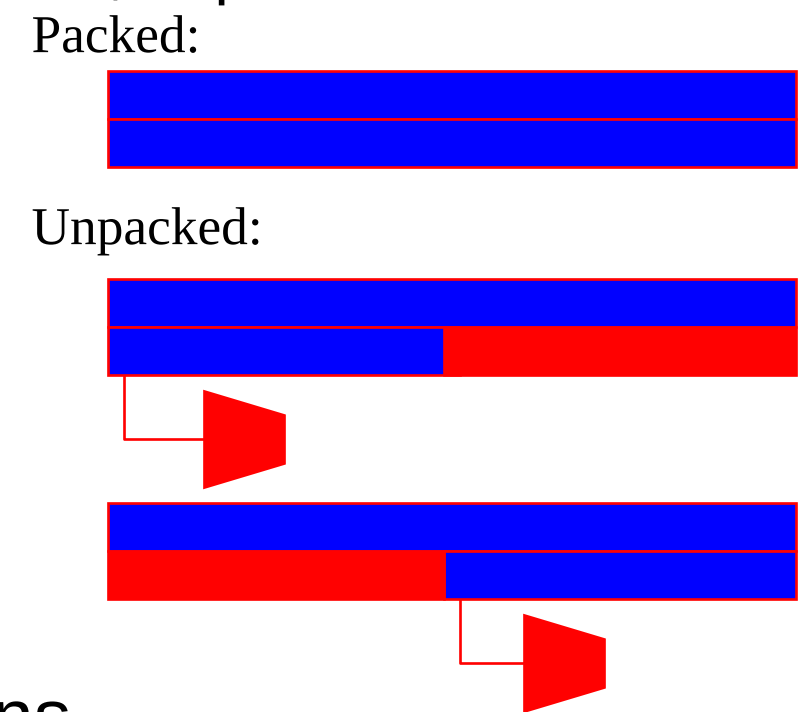
RTL	`reg o;`
	`always @(a or b)`
	`    o = a & b;`
		`reg o;`
		`always_comb`
		`    o = a & b;`

Gate	`wire o;` `and aa (o, a, b);`	`reg o;` `and aa (o, a, b);`

Multiple Types for the Same Bits

- Unions document various type views of the same bits
  - Packed unions force alignment, unpacked do not

```
typedef union packed {
    int signed int_r;
    byte [3:0] byte_r;
} union_t;

union_t ut;
signed int i;
byte [3:0] b;

i = ut.int_r;
b = ut.byte_r;
```



- Support only packed unions
  - Ensures single interpretation independent of the tool

System Functions for Type Analysis

- $typeof returns the type of its parameter
- $left, $right return the left and right indexes
- $high, $low return the largest and smallest indexes
- $length returns the number of elements
- $dimensions returns the number of dimensions
- $bits returns the number of bits

```
for (int i = $left(array); i <= $right(array); i++) begin
    array[i] = 0;
    end
```

– For loop index declarations are automatic

Advantages of SystemVerilog for Design

- Improved specification
  - Localization of functionality
  - Exact specification of intent
- Concise constructs for focused development
  - Line count correlates strongly with number of bugs
- Convenience in data abstraction
  - Correct data representation for many uses
- Unifying design and verification
  - Designs can specify requirements and guarantee they are met

Unifying Design and Verification

- Reducing the infamous “simulation synthesis mismatch”
  - Described earlier
    - Always_comb, always_latch
  - Unique, priority
- Documenting and checking assumptions as the design is coded
  - Assertions
- Issues frequently found late in integration, or even after RTL freeze, can be exposed earlier

Unique and Priority

- Reduces simulation/synthesis mismatches

```
unique      // synopsys parallel_case full_case
priority    // synopsys full_case
```

- Unique
  - Simulation error if there is more than one true condition
  - Simulation error if condition is not enumerated
- Priority
  - Simulation error if condition is not enumerated
- Apply to both case statements and if…else…if chains
- Synthesis
  - As if the appropriate pragma were applied

Assertions

- Combinational assertions exist in VHDL today
- Allow designers to ensure assumptions are met

```
module one_hot_mux (output o, input [3:0] sel, input a, b, c, d);

    assert($onehot(sel));
```

- Usage issues are caught early in the design cycle
- Synthesis
  - Assertions are ignored
  - Discuss synthesis of hardware in Food for Thought

Overview

- What is SystemVerilog?
- Advantages for design
- Use model and synthesis results
- Food for thought
- Roadmap

Flow Continuity

- Verification Flows
  - DC, VCS, Leda and Formality teams are working together to ensure seamless interoperability
    - Sharing testcases
    - Agreeing on finer points in language semantics
- Implementation flow
  - read -f sverilog, analyze -f sverilog, read_sverilog
  - Other commands remain unchanged
- Netlists for backend flows
  - Output netlists will continue to work as before
    - SystemVerilog is not required

SystemVerilog and VHDL Interoperability

- DesignCompiler supports a mixed design language environment today
  - Limited Verilog language forces ports to be arrays
- With SystemVerilog, structures and multi-dimensional arrays can be used as ports between SystemVerilog and VHDL modules
- Interfaces are not currently crossing the language boundary
  - Specification of functions
  - Instantiation interfaces and port interfaces do not have to exactly match in SystemVerilog

QoR and Runtime Results

- Two customer examples of Verilog designs rewritten using SV constructs
  - Typedefs
  - Structures
  - Interfaces
  - Modports
  - Always_* forms
  - Enums
  - ++ operator

QoR Analysis: Risc Core

	Verilog	SystemVerilog
Levels of Logic	25.00	25.00
Critical Path Length	3.36	3.36
Critical Path Slack	0.00	0.00
Total Negative Slack	0.00	0.00
No. of Violating Paths	0.00	0.00
Combinational Area	10809	10809
Noncombinational Area	11786	11786
Net Area	11	11
Cell Area	22595	22595
Design Area	22606	22606
Overall Compile Time	133.85	132.53

QoR Analysis: SDRam Controller

	Verilog	SystemVerilog
Levels of Logic	10.00	10.00
Critical Path Length	6.38	6.38
Critical Path Slack	-6.38	-6.38
Total Negative Slack	-701.23	-701.91
No. of Violating Paths	204.00	204.00
Combinational Area	10590	10590
Noncombinational Area	14903	14907
Net Area	0	0
Cell Area	25493	25497
Design Area	25493	25497
Overall Compile Time	105.60	105.67

Overview

- What is SystemVerilog?
- Advantages for design
- Use model and synthesis results
- Food for thought
- Roadmap

Food for Thought: Pipelines

- *Fork … join_none* **describes a pipeline**

```
always @(posedge clk)
    fork begin
            A = B + C * D + E - F;
            out <= repeat (latency) @(posedge clk) A;
        end
    join_none
```

- **Becomes computation followed by flip-flops**
- **Retiming can spread the calculation across the indicated number of flip flops**

Food for Thought: Assertions in Synthesis

```
assert(I<j) $display (“true”); else $display (“false”);
```

- Allow a user to specify in procedural code that a fact is true
- Then and else clauses allow testbench actions to occur depending upon the truth of the assertion
- In the short term, synthesis will ignore the content of the then and else clauses

- Discussion applies to both SystemVerilog and VHDL

Possible Uses for Assertions

- Use assertions to specify pragmas
  - *assert($onehot(…));*
- Use assertions to achieve better QoR
  - *assert(I!= j);*
  - *mem[I] = …;*
  - *mem[j] = …;*
- Synthesize assertions to enable better prototyping and emulation
  - Each assertion to a different wire dictated by the then and else clauses?
- Other ideas?

Food for Thought: SystemVerilog 3.1, 3.1a

- Primarily testbench and assertion extensions to the SystemVerilog design subset
  - May be used in the future to improve synthesis
- Synthesizable constructs in 3.1, 3.1a
  - From VHDL
    - Alias
    - Variable width function and task ports []
    - Default arguments to tasks and functions
    - Operator overloading
    - Packages
  - Tagged unions
  - Static queues (FIFO)

SystemVerilog End-user Momentum

"We believe Synopsys' strong support of the Accellera SystemVerilog standard, and its use of the standard to drive the design-for-verification methodology will further enhance verification productivity and quality."
Rich Heye, vice president and general manager, AMD Microprocessor Business Unit

"A unified design and verification language, such as the Accellera SystemVerilog standard, will pave the way for the productivity and efficiency improvements needed to stay ahead of SoC challenges."
Shrenik Mehta, director, Sun Microsystems

"SystemVerilog 3.1 will provide significant enhancements to the Verilog language… These enhancements, when combined with good methodology, will significantly improve design and verification productivity."
Vassilios Gerousis, chief scientist, Infineon Technologies

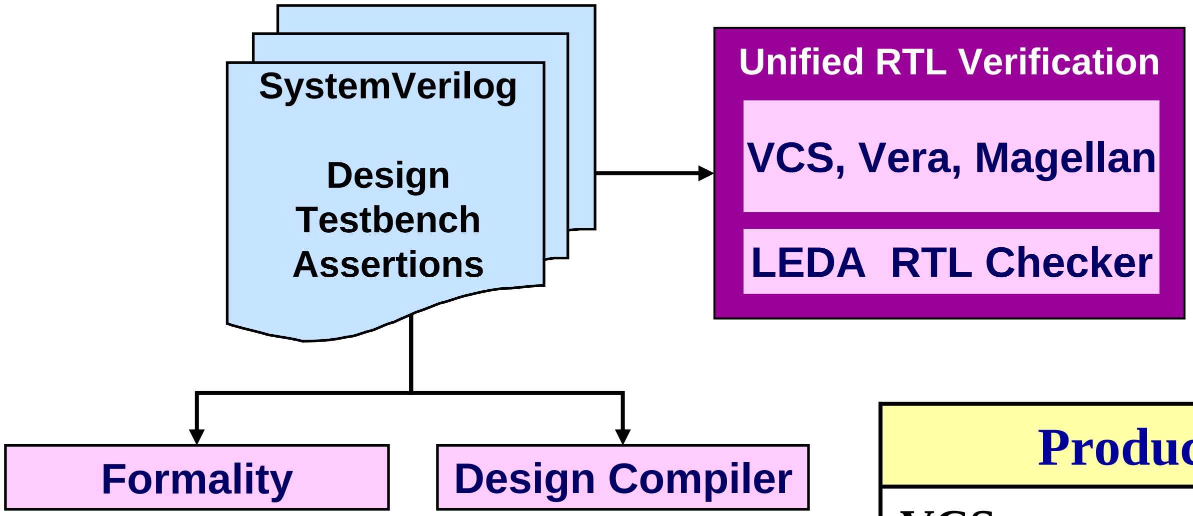
"… we are impressed with SystemVerilog's powerful design features and advanced verification capabilities. With Synopsys' creation of a complete design and verification flow utilizing SystemVerilog, we expect rapid adoption of the standard."
Ulrich Hummel, manager of CAD, Micronas Gmbh

SUG EUROPE 2004

SystemVerilog Catalyst Program Members

SAFELOGIC Property based verification.
bluespec
SPIKE Technologies
ingot
eve
avery design systems
SYNAPTICAD Tools for the Thinking Mind
Tharas Systems
NOVAS
ALATEK Leading Innovation of Design Verification
ARM
PROVER TECHNOLOGY
INTRINSIX
interra systems
Real Intent
Verific Design Automation
ATTEST Design Automation
VhdlCohen Training
DOULOS
Atrenta
ComputerBasedEducation.com
JASPER design automation
SUTHERLAND HDL
0-IN DESIGN AUTOMATION
Axis systems
veriEZ The verification Tools Company
PROVIS
GDA Technologies
Beach Solutions
Verifica
ALDEC The Design Verification Company
TRANSEDA VERIFICATION FROM CONCEPT TO REALITY
nSys
TERA SYSTEMS The New Frontend
tNi-VALIOSYS
chipvision
Veritools
atinec
SiConcepts
hdlLab
WILLAMETTE HDL
Veritable
Silicon Interfaces a software and vlsi design center
SUMMIT
TenisonEDA
SunburstDesign
WSFDB Consulting
SILICOMOTIVE SOLUTIONS
SEQUENCE
InTime SOFTWARE
Aptix
ControlNet INDIA

SystemVerilog Roadmap



We are moving to support SystemVerilog across the board.

Product	Release
VCS	NOW
Design Compiler	NOW
LEDA	NOW
Formality	NOW IN BETA
VCS (SV 3.1)	Q2, 2004
Vera (SV 3.1 TB)	Q2, 2004
Magellan (SV 3.1)	Q2, 2004

SystemVerilog *More Info*

- Additional information available in the SNUG Panel Discussion
  - (G1) R&D Panel -- SystemVerilog Design and Verification Implications for Users of DesignCompiler and VCS
    - 3:45 to 5:15pm today

- Useful websites on SystemVerilog:

  www.systemverilog.org

  www.systemverilognow.com

  www.accellera.org

Summary

- Migration to SystemVerilog
  – Verilog designs will largely work unchanged
  – Implementation flows are not affected
- SystemVerilog raises the abstraction level improving productivity
  – Improved specification
  – Concise constructs for focused development
  – Convenience in data abstraction
  – Unifying design and verification
- Advantages are available with minimal impact to existing flows