

# Assertions Update for SystemVerilog

**Bruce S. Greene**  
Technical Marketing Manager



October 16, 2003

 *Your Interoperability Partner*

## Agenda

- Requirements for Assertions
- Introduction to SystemVerilog Assertions
- Future of OVA
- Comparison
- Conclusion



## Requirements

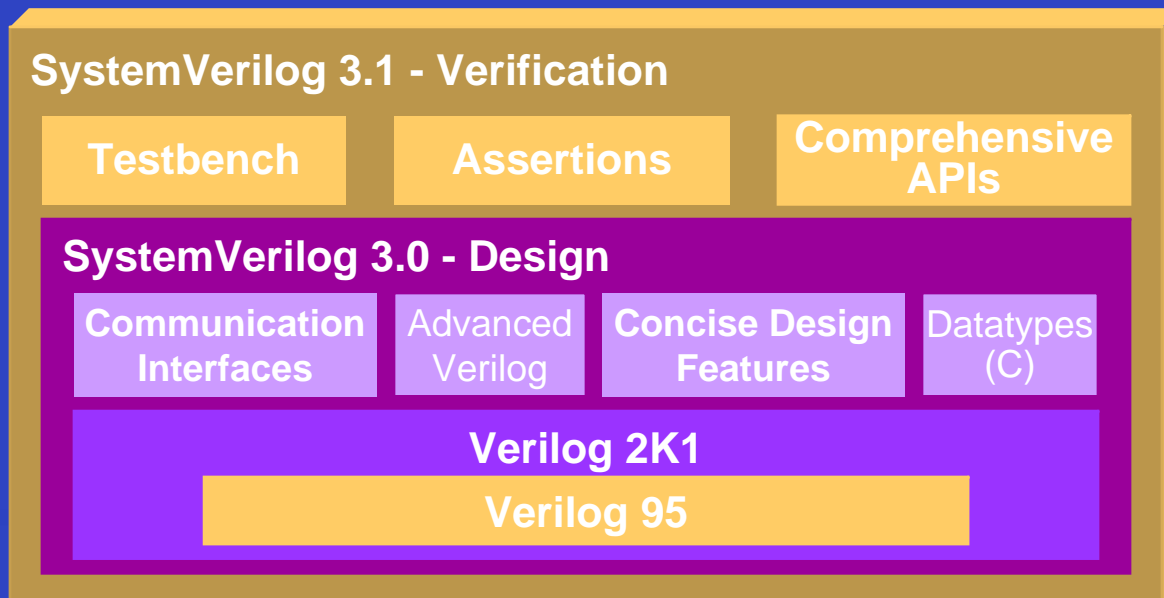
- Assertions must easily capture protocols and specifications
- Need to reuse assertions across a variety of verification tools
- Powerful functionality required, but ...
- ... assertions must ultimately live and interact with design / testbench

## SystemVerilog Assertions

- Part of Verilog
  - no pragmas, no specialized language
  - maximum user productivity
- Easily usable by both design and verification engineers
- Reusable across both simulation & formal
  - no modifications and consistent results!
- Semantically the same as OVA – familiar usage
- Flexible implementation - inlined or separate file

For maximum usability, assertions must be part of the Verilog language with same look and feel!

# SystemVerilog: Unifying Design and Verification



- Assertion features now accessible to design & verification engineers
- Assertions fully integrated into Verilog
- 100% compatible with Verilog

**Next-Generation  
HDVL**

 *Your Interoperability Partner*

## Features of SVA

- Immediate Assertions
- Concurrent Assertions
- Contextual Extraction
- Action Block
- Multiple Clocks
- Inlined / Binding
- Tasks/Functions
- API
- Scheduling



# Immediate Assertions

```
if (condition1) then
  if (condition2)
    else if (condition3) then
      begin
        req1 = siga || sigb
        req2 = sigc || sigd
        assert (req1 ^ req2);
      end
    end if;
end if;
```

- Great for embedding simple assertions in design code

# Concurrent Assertion with Action Block

```
assert property (foo)
else begin
    my_notifier = 1;
    $my_custom_error;
    $display($time,,"%d failed",error_signal);
end
```

- Great for reacting to assertion failures

# Contextual Extraction

```
always @(posedge clk)
  case (st)
    ACK:
      if (foo == 1)
        begin
          P5: assert property (!req[*5]);
          ...
        end
  ..
  ..
```

- Great for writing assertions without having to duplicate control logic

# External Instantiation

```
bind router my_checker u1 (...);  
  
// my_checker is an interface/module/program  
// router is my dut  
// u1 is instance name of checker
```

- Good for keeping assertions separate from design

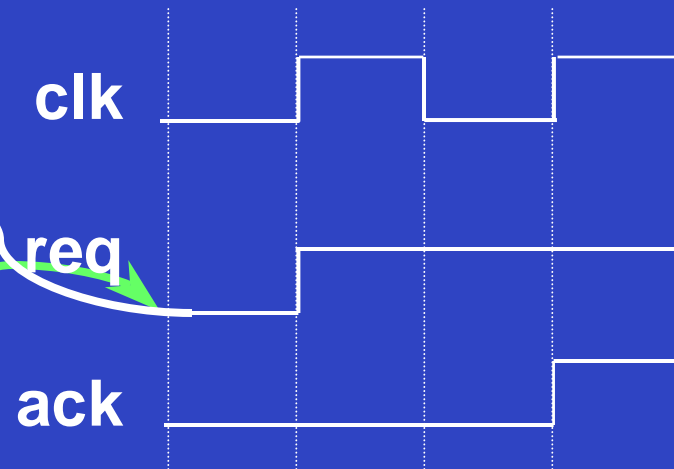


# Assertion Scheduling

```
property p1;
@( posedge clk )

  req |-> ##[1:3] ack;

endproperty
```



- 1) Preponed
- 2) Pre-active
- 3) Active
- 4) Inactive
- 5) Pre-NBA
- 6) NBA
- 7) Post-NBA
- 8) Observed
- 9) Post-observed
- 10) Reactive
- 11) Postponed

SV LRM

Clearly defined scheduling prevents race conditions and inconsistent results

# Assertion Usage



```

module router ( clk,
               rst,
               frame_data,out_a,
               valid,
               frame_o,
               channel0,
               channel1,
               channel2,
               channel3,
               frame0,
               frame1,
               frame2,
               frame3);

input  clk;
input  rst;
input  valid;
input [423:0] frame_data;

output out_a,frame_o,frame0,frame1,frame2,frame3;
output [423:0] channel0,channel1,channel2,channel3;

reg    out_a,frame_o,frame0,frame1,frame2,frame3;
reg [7:0] value,value_out;

a_frame0 : assert property (@(posedge clk)
    (valid & ch0 ) -> ##[0:3] frame0)
    else $display ("Bad Frame on Channel 0");

a_frame1 : assert property (@(posedge clk)
    (valid & ch1) -> ##[0:3] frame1)
    else $display ("Bad Frame on Channel 1");

reg [7:0] packet_memory[52:0];
reg [7:0] packet_memory_out[52:0];
reg [423:0] channel0,channel1,channel2,channel3;
reg [7:0] mem;
reg [7:0] mem_out;
integer i,j,k;
channel2 <= 0;
channel3 <= 0;
end
else begin

a_frame1 : assert property (@(posedge clk)
    (valid & ch1) -> ##[0:3] frame1)
    else $display ("Bad Frame on Channel 1");

```

```

module router ( clk,
               rst,
               frame_data,out_a,
               valid,
               frame_o,
               channel0,
               channel1,
               channel2,
               channel3,
               frame0,
               frame1,
               frame2,
               frame3);

input  clk;
input  rst;
input  valid;
input [423:0] frame_data;

output out_a,frame_o,frame0,frame1,frame2,frame3;
output [423:0] channel0,channel1,channel2,channel3;

reg    out_a,frame_o,frame0,frame1,frame2,frame3;
reg [7:0] value,value_out;

a_frame0 : assert property (@(posedge clk)
    (valid & ch0 ) -> ##[0:3] frame0)
    else $display ("Bad Frame on Channel 0");

a_frame1 : assert property (@(posedge clk)
    (valid & ch1) -> ##[0:3] frame1)
    else $display ("Bad Frame on Channel 1");

reg [7:0] packet_memory[52:0];
reg [7:0] packet_memory_out[52:0];
reg [423:0] channel0,channel1,channel2,channel3;
reg [7:0] mem;
reg [7:0] mem_out;
integer i,j,k;
channel2 <= 0;
channel3 <= 0;
end
else begin

a_frame1 : assert property (@(posedge clk)
    (valid & ch1) -> ##[0:3] frame1)
    else $display ("Bad Frame on Channel 1");

```

Your Interoperability Partner

# Pre-written, Pre-verified Checkers

```
module assert_zero_one_hot(clk, reset_n, test_expr);
    parameter          severity_level      = 0;
    parameter          width              = 32;
    parameter          options            = 0;
    parameter          msg                 = "VIOLATION";
    input               clk;
    input               reset_n;
    input [(width - 1):0] test_expr;

    ..
    ..

    assert_zero_one_hot:
        assert property(@(posedge clk) not_resetting |->
            (($countones(test_expr) == 1) || (!(|test_expr))))
            else $display( msg);
endmodule
```

## Greater Productivity



```

module router ( clk,
               rst,
               frame_data, outa,
               valid,
               frame_o,
               channel0,
               channel1,
               channel2,
               channel3,
               frame0,
               frame1,
               frame2,
               frame3);

input  clk;
input  rst;
input  valid;
input  [423:0] frame_data;

output outa, frame_o, frame0, frame1, frame2, frame3;
output [423:0] channel0, channel1, channel2, channel3;

reg [7:0] outa, frame_o, frame0, frame1, frame2, frame3;
reg [7:0] value, value_out;

assert_zero_one_hot #(0,4) frame_one_hot (clk, 1'b1, frame_arr);

reg [7:0] packet_memory[52:0];
reg [7:0] packet_memory_out[52:0];
reg [423:0] channel0, channel1, channel2, channel3;
reg [7:0] mem;
reg [7:0] mem_out;
integer i, j, k;
channel2 <= 0;
channel3 <= 0;

a_frame1 : assert property (@(posedge clk)
    (valid & ch1) -> ##[0:3] frame1)
    else $display ("Bad Frame on Channel 1");

module router ( clk,
               rst,
               frame_data, outa,
               valid,
               frame_o,
               channel0,
               channel1,
               channel2,
               channel3,
               frame0,
               frame1,
               frame2,
               frame3);

input  clk;
input  rst;
input  valid;
input  [423:0] frame_data;

output outa, frame_o, frame0, frame1, frame2, frame3;
output [423:0] channel0, channel1, channel2, channel3;

reg [7:0] outa, frame_o, frame0, frame1, frame2, frame3;
reg [7:0] value, value_out;

assert_zero_one_hot #(0,4) frame_one_hot (clk, 1'b1, frame_arr);

reg [7:0] packet_memory[52:0];
reg [7:0] packet_memory_out[52:0];
reg [423:0] channel0, channel1, channel2, channel3;
reg [7:0] mem;
reg [7:0] mem_out;
integer i, j, k;
channel2 <= 0;
channel3 <= 0;

a_frame1 : assert property (@(posedge clk)
    (valid & ch1) -> ##[0:3] frame1)
    else $display ("Bad Frame on Channel 1");

module router ( clk,
               rst,
               frame_data, outa,
               valid,
               frame_o,
               channel0,
               channel1,
               channel2,
               channel3,
               frame0,
               frame1,
               frame2,
               frame3);

input  clk;
input  rst;
input  valid;
input  [423:0] frame_data;

output outa, frame_o, frame0, frame1, frame2, frame3;
output [423:0] channel0, channel1, channel2, channel3;

reg [7:0] outa, frame_o, frame0, frame1, frame2, frame3;
reg [7:0] value, value_out;

assert_zero_one_hot #(0,4) frame_one_hot (clk, 1'b1, frame_arr);

reg [7:0] packet_memory[52:0];
reg [7:0] packet_memory_out[52:0];
reg [423:0] channel0, channel1, channel2, channel3;
reg [7:0] mem;
reg [7:0] mem_out;
integer i, j, k;
channel2 <= 0;
channel3 <= 0;

a_frame1 : assert property (@(posedge clk)
    (valid & ch1) -> ##[0:3] frame1)
    else $display ("Bad Frame on Channel 1");
    
```

`assert_zero_one_hot #(0,4) frame_one_hot (clk, 1'b1, frame_arr);`

## Example: Synopsys Checker Library

Over 50 checkers available!

### Value Integrity

- always
- always\_on\_edge
- bits
- change
- data\_used
- decrement
- delta
- driven
- even\_parity
- hold\_value
- increment
- mutex
- never
- no\_contention
- no\_overflow
- no\_underflow

### Value Integrity

- odd\_parity
- one\_cold
- one\_hot
- proposition
- quiescent\_state
- range
- value
- width
- zero\_one\_hot

### Protocol

- arbiter
- cycle\_sequence
- dual\_clk\_fifo
- fifo
- fifo\_index
- frame
- handshake
- implication
- memory\_async
- memory\_sync
- multiport\_fifo
- req\_loaded
- req\_ack\_unique
- req\_requires
- stack
- time
- valid\_id
- window

### State Integrity

- code\_distance
- next
- next\_state
- no\_transition
- transition
- unchange
- win\_unchange
- win\_change

# Interfaces + Assertions

- “Interfaces” deliver simplification and are highly reusable
- SystemVerilog’s interfaces encapsulate connectivity, communication
- Interfaces define a common set of signals and functions for block hook-up and access
- Assertions placed within the interface easily catch bugs between blocks

```
interface foo(input bit clk);  
  ...  
  wire req, gnt,ack;  
  task write(...);  
    ...  
  endtask  
  task read(...);  
    ...  
  endtask  
  
  property p1;  
    @( posedge clk )  
      req |-> gnt##[1:3] ack;  
  endproperty  
  a1 : assert property (p1) ;  
  
endinterface
```

Specify assertions in interface once, reuse for all blocks

## OpenVera Assertions (OVA)

- OVA is mature and production proven
  - Support from multiple vendors
- Current users can continue to use OVA
  - Synopsys continuing OVA support in existing products for legacy support
- OVA is semantically the same as SVA
  - Using OVA is good preparation for using SVA
- VCS™ beta support of SVA by November

## Agenda

- Requirements for Assertions
- Introduction to SystemVerilog Assertions
- Future of OVA
- **Comparison**
- Conclusion



## SVA and OVA - Keywords



assert  
 cover  
 \$rose  
 \$fell  
 \$past  
 throughout  
 countones  
 matched  
 ended  
 &&  
 ||  
 sequence  
 bind  
  
 property  
 endproperty  
 disable iff  
 not  
 sequence  
 endsequence  
 first\_match  
 \$stable

**SVA**

assert  
 cover  
 posedge  
 negedge  
 past  
 istrue  
 count  
 matched  
 ended  
 and  
 or  
 event  
 bind  
  
 bool  
 unit  
 forbid  
 check  
 clock

**OVA**

# SVA/OVA Syntax Comparison

delays, repetition

	SVA	OVA
simple delay	##1	#1
delay range	##[1:3]	#[1..3]
open-ended range	##[1:\$]	#[1..]
repetition	[*3]	*[3]

# SVA/OVA Syntax Comparison (cont)

sequence operation, implication

	SVA	OVA
sequence	and	&&
sequence or	or	
implication	( )  -> ( )	if ( ) then ( )
sequence condition	throughout	istrue
sequence match	e1.ended	ended(e1)

# SVA/OVA Comparison

other

	SVA	OVA
edge expression	\$rose	posedge
	\$fell	negedge
	\$stable	edge
past value	\$past	past
1's count	\$countones	count

## SVA/OVA

same binding style

### SVA

```
bind router my_checker u1 (...);
```

*where my\_checker is an interface/module/program*

### OVA

```
bind module router : my_checker u1(...) ;
```

*where my\_checker is the OVA unit*

## SVA contains functionality above OVA

- Immediate assertions
- More styles of repetition
- More sequence joining operators
- First\_match operator
- Sequence within sequence operator
- Local variable declaration within sequence
- More system functions

# SVA and OVA Share Same Semantics

```
property p1;  
  @( posedge clk )  
  
  req |-> gnt##[1:3] ack;  
  
endproperty  
  
a1 : assert property (p1);
```

SVA

```
clock posedge clk {  
  event p1 :  
    if (req) then gnt #[1..3]  
  ack;  
}
```

```
assert a1 : check(p1);  
}
```

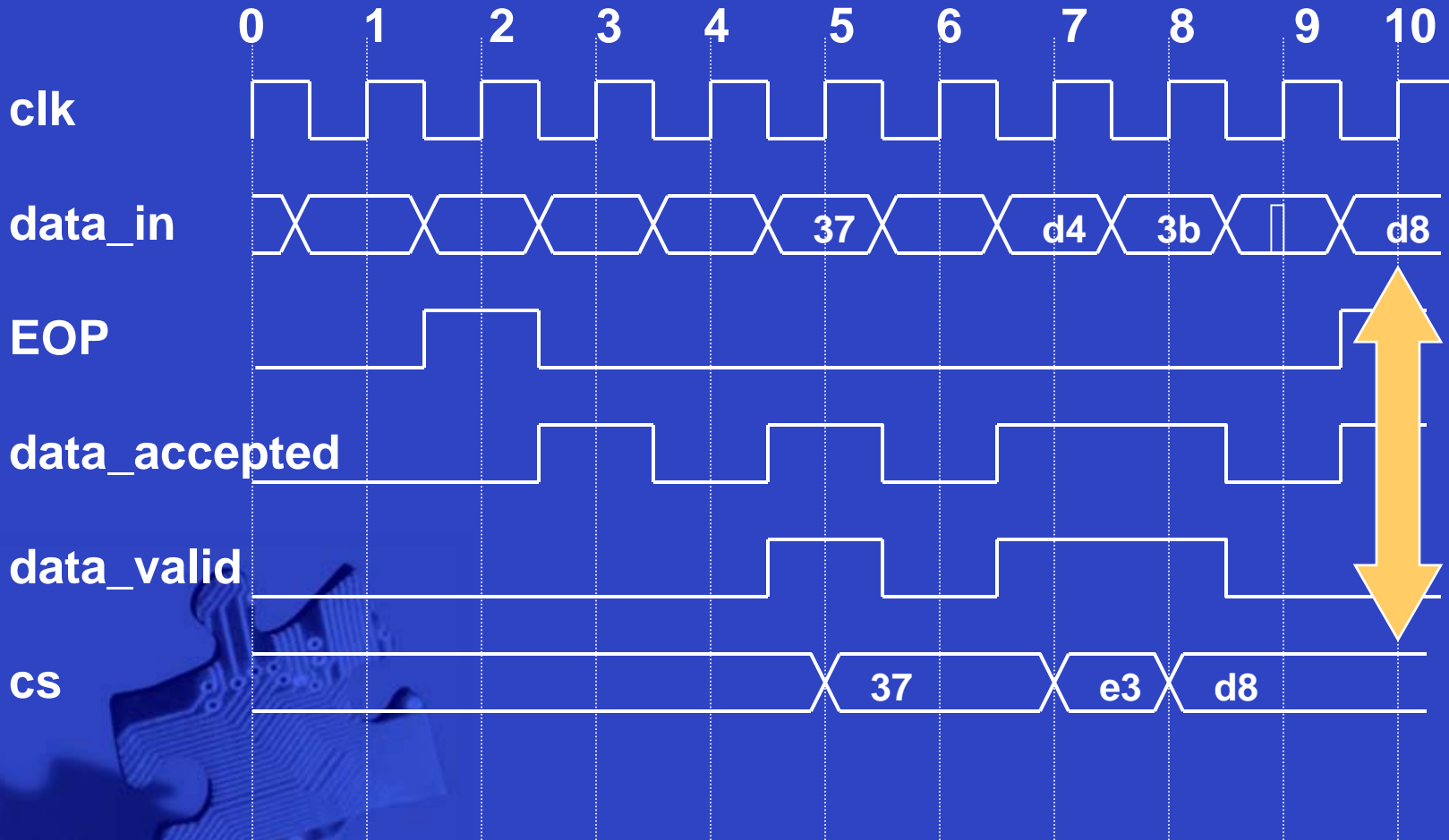
OVA

# Benefits: Local Variables

Parity calculation on the fly using SVA local variables

```
logic data_accepted = data_valid && next_data;
property packet_parity_check_p;
logic [7:0] cs;
  @(posedge clk) disable iff (!reset_n)
    ($rose(reset_n) || $past(EOP && data_accepted))
    ##0 (!data_valid)[*0:$] // await start of packet
    ##1 (data_accepted, cs = data_in) // start
    ##1 ((!data_valid)[*0:$] ##1
        (!EOP && data_accepted,
            cs = cs ^ data_in))[*0:$])
    ##1 (!data_valid)[*0:$]
    ##1 (EOP && data_accepted)
        |-> (cs == data_in); // check the parity
endproperty
```

# Waveforms



## Conclusion

- OVA is production proven
- SVA builds on that success
- Customers have choice of staying with OVA or using SVA

