SystemVerilog Assertions Design Tricks and SVA Bind Files

CLIFFORD E. CUMMINGS
SUNBURST DESIGN, INC.
CLIFFC@SUNBURST-DESIGN.COM
WWW.SUNBURST-DESIGN.COM

Life is too short for bad or boring training!

World Class Verilog & SystemVerilog Training

Introduction

- SystemVerilog added assertion-based design & verification

- The SVA "circle of life"
  - Engineers take SVA training
  - Engineers get excited about SVA capabilities
  - Engineers start to use SVA
  - Engineers find SVA to be too verbose
  - Engineers abandon SVA

"The Disillusioned Design Engineer !!"

aargh

Paper & Presentation Agendas

The three best tricks from the paper

In the paper ...

- Assertion fundamentals
- Long label usage
- Immediate assertions
- Casting & randomization
- Concurrent assertions
- Concise assertion coding styles
- **`default clocking`** / **`$assertkill`**
- Macros with arguments
- Benchmark assertion coding styles
- SVA bind files
- SVA bind file abuse
- SVA file methodologies
- Summary & conclusions

In the presentation ...

Documentation & debug

- Long label usage

Concise SVA coding styles

- Macros with arguments
- Benchmark assertion coding styles

Binding SVA to golden RTL code

- SVA bind files

- Conclusions

What Is An Assertion? Who Should Add Assertions?

- An assertion is a "*statement of fact*" or a "*claim of truth*" about the design

"...an *assertion* is a statement about a design's *intended behavior*"

  - If the assertion is not true, the assertion fails

- *Design assertions are best added by design engineers*

Assertions are active design comments **used to document intended behavior of a design and help to identify both design and verification flaws related to the design under development and test**

- How do you get design engineers to add assertions??

tell them ... **"you will spend less quality time with your verification engineer!"**

Assertion Labeling Technique

Assertion with No Label

Example: dff With Obvious Error

Look at a simple (and flawed) d flip-flop example

```
module dff (
  output logic q,
  input        d, clk, rst_n);

  assign q = d;
endmodule
```

This is clearly a mistake!!

Assertion: q should equal the value of d from the previous cycle

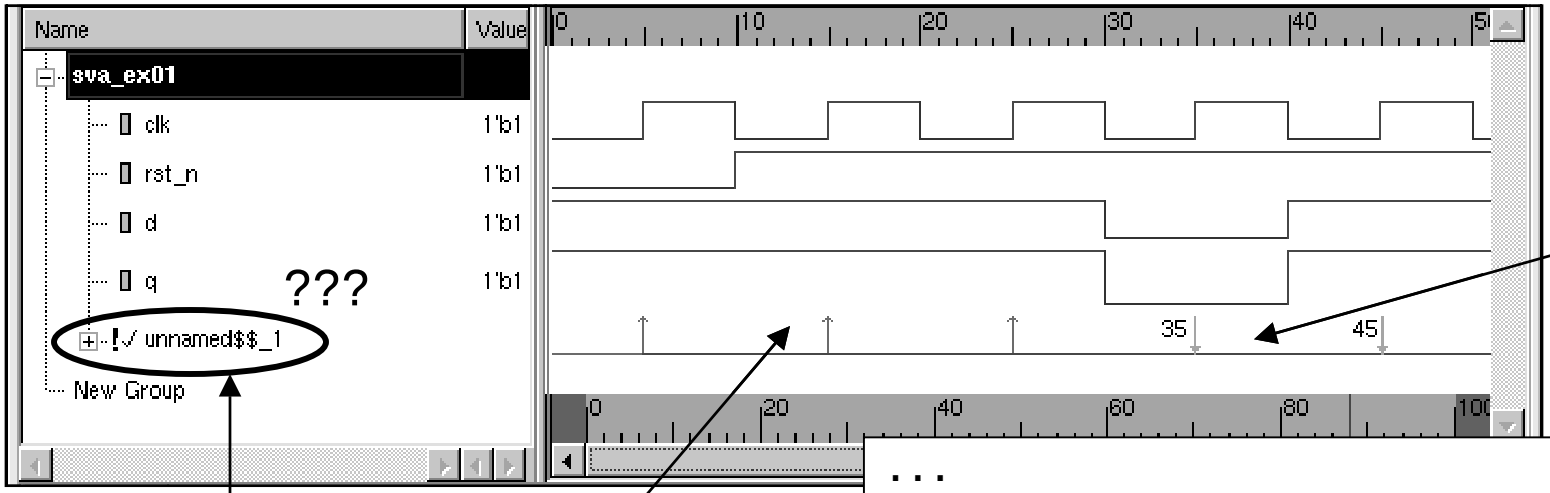
No label on the assertion

```
assert property (@(posedge clk) disable iff (!rst_n) (q==$past(d)))
  else $display("ERROR: q did not follow d");
```

Action block error message

Assertion Waveforms with Error $display

```
assert property (@(posedge clk) disable iff (!rst_n) (q==$past(d)))
  else $display("ERROR: q did not follow d");
```



What is this error???

No label yields default label in the waveform display

Error message in the STDOUT but not in the waveform display

```
...
 20ns: clk=0  rst_n=1  d=1  q=1
 25ns: clk=1  rst_n=1  d=1  q=1
 30ns: clk=0  rst_n=1  d=0  q=0
"sva_ex01.sv", 20: sva_ex01.unnamed$$_1:
         started at 35ns failed at 35ns
         Offending '(q == $past(d))'
ERROR: q did not follow d
 35ns: clk=1  rst_n=1  d=0  q=0
 40ns: clk=0  rst_n=1  d=1  q=1
...
```

Assertion with Label

Example: dff With Obvious Error

Same simple (and flawed) d flip-flop example

```
module dff (
  output logic q,
  input        d, clk, rst_n);

  assign q = d;
endmodule
```

This is clearly a mistake!!

"ERROR_" label added to the assertion - *includes description of error if assertion fails*

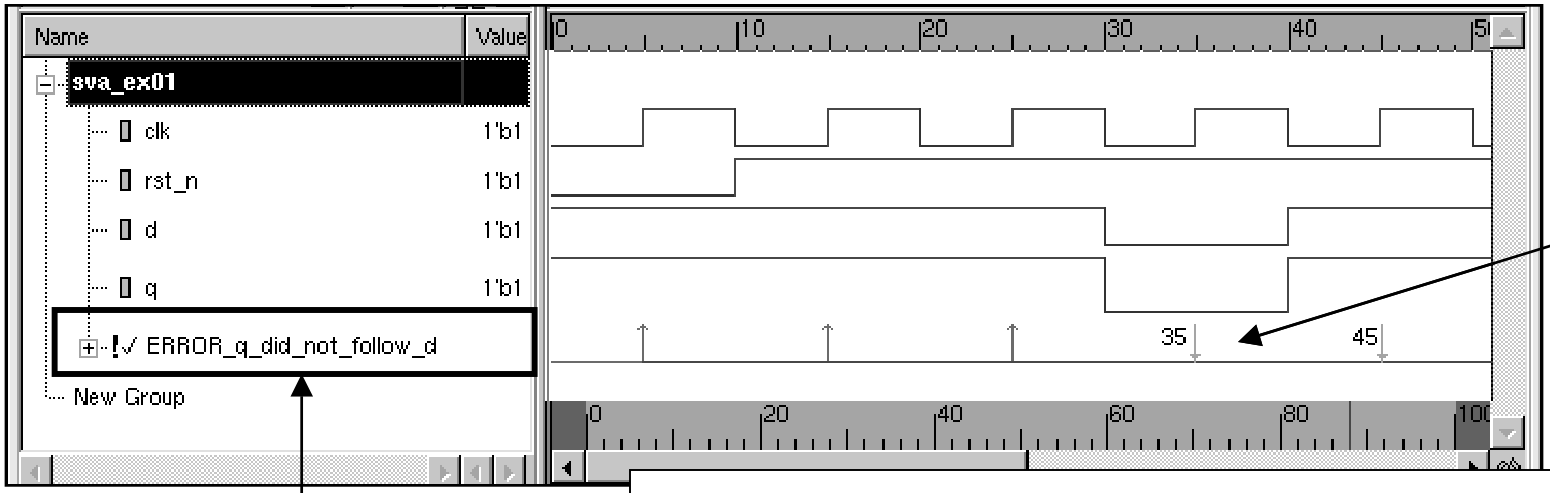
```
ERROR_q_did_not_follow_d:
  assert property (@(posedge clk) disable iff (!rst_n) (q==$past(d)));
```

No SVA action block error message

Assertion Waveforms with Assertion Label

Long descriptive assertion label

```
ERROR_q_did_not_follow_d:
  assert property (@(posedge clk) disable iff (!rst_n) (q==$past(d)));
```



Error with description

Long label helps to debug the assertion in the waveform display

```
...
 20ns: clk=0  rst_n=1  d=1  q=1
 25ns: clk=1  rst_n=1  d=1  q=1
 30ns: clk=0  rst_n=1  d=0  q=0
"sva_ex01.sv", 17: sva_ex01.ERROR_q_did_not_follow_d:
          started at 35ns failed at 35ns
         Offending '(q == $past(d))'
 35ns: clk=1  rst_n=1  d=0  q=0
 40ns: clk=0  rst_n=1  d=1  q
...
```

Long label also displayed in the STDOUT error message

Long labels help debug designs with assertions !!

SystemVerilog Concurrent Assertions

Concurrent Assertions

- Concurrent assertions are sampled
  - Typically once each clock cycle
  - Typically at the end of each clock cycle

Most valuable assertion style

```
assert property ( property_definition );
```

Simple form for a concurrent assertion

```
assert property (
  @(sample_signal)
  disable iff ( expression )
  property_expression_or_sequence
);
```

Property definition

Sample signal is typically the clock

Optional disable condition

Expression or sequence that should always be true
(this is what the assertion is testing)

Separate Property Definition & Assertion

- Commonly taught assertion coding style

Generally not recommended for design engineers

Define the property ...

```
property p1;
  @sample_signal
  disable iff ( expression )
  property_expression_or_sequence
endproperty
```

... then assert the property

```
assert property ( p1 ) optional_action_block ;
```

This style is good if someone is defining a separate library of properties for reuse

This style is typically bad for most design engineers

Too verbose, tedious and time consuming

Assert Property without Separate Definition

Generally not recommended for design engineers

- Second commonly taught assertion coding style

  Assert a defined property

No property name is declared

```
assert property (
  @sample_signal
  disable iff ( expression )
  property_expression_or_sequence );
```

This style is more concise than using a separate property definition

This style is still typically bad for most design engineers

Still too verbose, tedious and time consuming

Verilog-95 Macro Capabilities

```
`define CYCLE 100
...
initial  begin
  clk <= '0;
  forever #(`CYCLE/2) clk = ~clk;
end
```

Define and use a macro
(every Verilog coder knows this style)

**Line continuation character **

```
`define assert_clk(  ...  ) \
  assert property (@(posedge clk) disable iff (!rst_n)  ...  )
```

Define a *multi-line macro*
(not all Verilog coders know this)

Pass an argument to the macro `arg`

```
`define assert_clk( arg ) \
  assert property (@(posedge clk) disable iff (!rst_n) arg )
```

Define a *multi-line macro with input arguments*
(not all Verilog coders know this)

Example Assertion Pieces

```
ERROR_q_did_not_follow_d:
  assert property (@(posedge clk) disable iff (!rst_n) (q==$past(d)));
```

Assertion from slide 8

- Macro pieces that are likely to be repeated include:

```
assert property

(  ...  );

@(posedge clk)

disable iff (!rst_n)
```

Keywords to start the definition of an assertion

Placeholder for the assertion

Sample signal for the concurrent assertion

Definition of when the assertion should become inactive

- Only piece that is likely to be unique to the assertion is:

```
(q==$past(d))
```

Actual assertion test

Example Assertion Defined Using a Macro

```
property p1;
  @(posedge clk)
  disable iff (!rst_n)
  (d |=> q);
endproperty

ERROR_q_did_not_follow_d:
  assert property (p1);
```

Define a separate `property` and then `assert` it

`assert` a property without a separate definition

```
ERROR_q_did_not_follow_d:
  assert property (@(posedge clk) disable iff (!rst_n) (q==$past(d)));
```

Define an `assert_clk` macro ...

```
`define assert_clk( arg ) \
  assert property (@(posedge clk) disable iff (!rst_n) arg )
```

```
ERROR_q_did_not_follow_d:
  `assert_clk(q==$past(d));
```

... then use the `` `assert_clk `` macro multiple times (very concise!)

Synchronous 16-Deep FIFO Assertions Examples

Synchronous 16-Deep FIFO Assertion Subset

- When the FIFO is reset, the FIFO **`empty`** flag should be set and the **`full`** flag, **`wptr`**, **`rptr`** and **`cnt`** should all be cleared
- If **`cnt`** is greater than 15, the FIFO is **`full`**
- If **`cnt`** is less than 16, the FIFO is not **`full`**
- If **`cnt`** is 15 and there is a write operation without a simultaneous read operation, the FIFO should go **`full`**
- If the FIFO is **`full`**, and there is a write operation without a simultaneous read operation, the full flag should not change
- If the FIFO is **`full`**, and there is a write operation without a simultaneous read operation, the write pointer should not change

Separate Properties & Assertion

Synchronous FIFO Assertion Subset - part 1


```
property reset_rptr0_wptr0_empty1_full0_cnt0;
  @(posedge clk)
    (!rst_n |->
        (rptr==0 && wptr==0 && empty==1 && full==0 && cnt==0));
endproperty

property full_fifo_condition;
  @(posedge clk) disable iff (!rst_n)
      (cnt>15 |-> full);
endproperty

property not_full_fifo_condition;
  @(posedge clk) disable iff (!rst_n)
      (cnt<16 |-> !full);
endproperty
...
```

FIFO reset property

FIFO full condition properties

Separate Properties & Assertion

Synchronous FIFO Assertion Subset - part 2


```
...
property fifo_should_go_full;
  @(posedge clk) disable iff (!rst_n)
      (cnt==15 && write && !read |=> full);
endproperty

property full_write_full;
  @(posedge clk) disable iff (!rst_n)
      (full && write && !read |=> full);
endproperty

property full_write_wptr_no_change;
  @(posedge clk) disable iff (!rst_n)
      (full && write && !read |=> $stable(wptr));
endproperty
...
```

More FIFO full condition properties

Separate Properties & Assertion

Synchronous FIFO Assertion Subset - part 3

Sunburst Design

```
...
ERROR_FIFO_RESET_SHOULD_CAUSE_EMPTY1_FULL0_RPTR0_WPTR0_CNT0:
  assert property (reset_rptr0_wptr0_empty1_full0_cnt0);

ERROR_FIFO_SHOULD_BE_FULL:
  assert property (full_fifo_condition);

ERROR_FIFO_SHOULD_NOT_BE_FULL:
  assert property (not_full_fifo_condition);

ERROR_FIFO_DID_NOT_GO_FULL:
  assert property (fifo_should_go_full);

ERROR_FIFO_FULL__WRITE_CAUSED_FULL_FLAG_TO_CHANGE:
  assert property (full_write_full);

ERROR_FIFO_FULL__WRITE_CAUSED_WPTR_TO_CHANGE:
  assert property (full_write_wptr_no_change);
```

Now assert the predefined FIFO properties

Asynchronous reset assertion

FIFO full condition assertions

Lines = 37
Characters = 1,225
(blank lines omitted)

Too much effort to test six FIFO features !!

Combined Assert Property Style

Synchronous FIFO Assertion Subset - part 1


```
ERROR_FIFO_RESET_SHOULD_CAUSE_EMPTY1_FULL0_RPTR0_WPTR0_CNT0:
  assert property (@(posedge clk)
    (!rst_n |->
        (rptr==0 && wptr==0 && empty==1 && full==0 && cnt==0)));

ERROR_FIFO_SHOULD_BE_FULL:
  assert property (@(posedge clk) disable iff (!rst_n)
      (cnt>15 |-> full));

ERROR_FIFO_SHOULD_NOT_BE_FULL:
  assert property (@(posedge clk) disable iff (!rst_n)
      (cnt<16 |-> !full));

ERROR_FIFO_DID_NOT_GO_FULL:
  assert property (@(posedge clk) disable iff (!rst_n)
      (cnt==15 && write && !read |=> full));
...
```

FIFO reset assertion

FIFO full condition assertions

Combined Assert Property Style

Synchronous FIFO Assertion Subset - part 2


```
...
ERROR_FIFO_FULL__WRITE_CAUSED_FULL_FLAG_TO_CHANGE:
  assert property (@(posedge clk) disable iff (!rst_n)
      (full && write && !read |=> full));

ERROR_FIFO_FULL__WRITE_CAUSED_WPTR_TO_CHANGE:
  assert property (@(posedge clk) disable iff (!rst_n)
      (full && write && !read |=> $stable(wptr)));
```

More FIFO full condition assertions

Lines = 19
Characters = 809
(blank lines omitted)

Assertion Macro Style

Useful Macro Definitions

```
`define assert_clk(arg) \
    assert property (@(posedge clk) disable iff (!rst_n) arg)

`define assert_async_rst(arg) \
    assert property (@(posedge clk) arg)
```

assert_clk macro definition

; is not part of the macro definitions

assert_async_reset macro definition

When macros are used, add the end-of-expression ;

Assertion Macro Style

Synchronous FIFO Assertion Subset

```
ERROR_FIFO_RESET_SHOULD_CAUSE_EMPTY1_FULL0_RPTR0_WPTR0_CNT0:
    `assert_async_rst(!rst_n |->
       (rptr==0 && wptr==0 && empty==1 && full==0 && cnt==0));

ERROR_FIFO_SHOULD_BE_FULL:
    `assert_clk (cnt>15 |-> full);

ERROR_FIFO_SHOULD_NOT_BE_FULL:
    `assert_clk (cnt<16 |-> !full);

ERROR_FIFO_DID_NOT_GO_FULL:
    `assert_clk (cnt==15 && write && !read |=> full);

ERROR_FIFO_FULL__WRITE_CAUSED_FULL_FLAG_TO_CHANGE:
    `assert_clk (full && write && !read |=> full);

ERROR_FIFO_FULL__WRITE_CAUSED_WPTR_TO_CHANGE:
    `assert_clk (full && write && !read |=> $stable(wptr));
```

FIFO reset assertion

FIFO full condition assertions

Lines = 13
Characters = 725
(blank lines omitted)

Concise coding style!

The *overhead-code* is defined in the macros

Synchronous 16-Deep FIFO
13 Assertions Benchmark

Synchronous 16-Deep FIFO Assertions Benchmark

- 13 different assertions coded using the previously shown three different assertion coding styles:
  - Separate **`property`** and **`assert`**
  - Combined **`assert property`**
  - Using macro definitions

Commonly taught coding styles

New style - recommended for design engineers

Assertion Coding Efforts

Blank Lines Omitted

```
property reset_rptr0_wptr0_empty1_full0_cnt0;
  @(posedge clk)
    (!rst_n |->
       (rptr==0 && wptr==0 && empty==1 && full==0 && cnt==0));
endproperty
property full_fifo_condition;
  @(posedge clk) disable iff (!rst_n)
      (cnt>15 |-> full);
endproperty
property not_full_fifo_condition;
  @(posedge clk) disable iff (!rst_n)
      (cnt<16 |-> !full);
endproperty
property fifo_should_go_full;
  @(posedge clk) disable iff (!rst_n)
      (cnt==15 && write && !read |=> full);
endproperty
property full_write_full;
  @(posedge clk) disable iff (!rst_n)
      (full && write && !read |=> full);
endproperty
property full_write_wptr_no_change;
  @(posedge clk) disable iff (!rst_n)
      (full && write && !read |=> $stable(wptr));
endproperty
property empty_fifo_condition;
  @(posedge clk) disable iff (!rst_n)
      (cnt==0 |-> empty);
endproperty
property not_empty_fifo_condition;
  @(posedge clk) disable iff (!rst_n)
      (cnt>0 |-> !empty);
```

Property/Assert Property
Lines = 79
Characters = 2,599
(blank lines omitted)

193% more lines of code
126% more characters

```
ERROR_FIFO_FULL__WRITE_CAUSED_WPTR_TO_CHANGE:
  assert property (full_write_wptr_no_change);
ERROR_FIFO_SHOULD_BE_EMPTY:
  assert property (empty_fifo_condition);
ERROR_FIFO_SHOULD_NOT_BE_EMPTY:
  assert property (not_empty_fifo_condition);
ERROR_FIFO_DID_NOT_GO_EMPTY:
  assert property (fifo_should_go_empty);
ERROR_FIFO_EMPTY__READ_CAUSED_EMPTY_FLAG_TO_CHANGE:
  assert property (empty_read_empty);
ERROR_FIFO_EMPTY__READ_CAUSED_RPTR_TO_CHANGE:
  assert property (empty_read_rptr_no_change);
ERROR_FIFO_WORD_COUNTER_IS_NEGATIVE:
  assert property (illegal_fifo_cnt_condition);
ERROR_FIFO_READWRITE_ILLEGAL_FIFO_FULL_OR_EMPTY:
assert property (fifo_not_empty_not_full);
```

```
ERROR_FIFO_RESET_SHOULD_CAUSE_EMPTY1_FULL0_RPTR0_WPTR0_CNT0:
  assert property (@(posedge clk)
    (!rst_n |->
       (rptr==0 && wptr==0 && empty==1 && full==0 && cnt==0)));
ERROR_FIFO_SHOULD_BE_FULL:
  assert property (@(posedge clk) disable iff (!rst_n)
      (cnt>15 |-> full));
ERROR_FIFO_SHOULD_NOT_BE_FULL:
  assert property (@(posedge clk) disable iff (!rst_n)
      (cnt<16 |-> !full));
ERROR_FIFO_DID_NOT_GO_FULL:
  assert property (@(posedge clk) disable iff (!rst_n)
      (cnt==15 && write && !read |=> full));
ERROR_FIFO_FULL__WRITE_CAUSED_FULL_FLAG_TO_CHANGE:
  assert property (@(posedge clk) disable iff (!rst_n)
      (full && write && !read |=> full));
ERROR_FIFO_FULL__WRITE_CAUSED_WPTR_TO_CHANGE:
  assert property (@(posedge clk) disable iff (!rst_n)
      (full && write && !read |=> $stable(wptr)));
ERROR_FIFO_SHOULD_BE_EMPTY:
  assert property (@(posedge clk) disable iff (!rst_n)
      (cn
ERROR_FIF
  assert
      (cn
ERROR_FIF
  assert
      (cn
ERROR_FIF
  assert
      (em
ERROR_FIF
  assert
      (em
ERROR_FIF
  assert property (@(posedge clk) disable iff (!rst_n)
      (not (cnt<0)));
ERR
  as
  (
```

Assert Property
Lines = 40
Characters = 1,707
(blank lines omitted)

48% more lines of code
48% more characters

```
ERROR_FIFO_RESET_SHOULD_CAUSE_EMPTY1_FULL0_RPTR0_WPTR0_CNT0:
    `assert_async_rst(!rst_n |->
        (rptr==0 && wptr==0 && empty==1 && full==0 && cnt==0));
ERROR_FIFO_SHOULD_BE_FULL:
    `assert_clk (cnt>15 |-> full);
ERROR_FIFO_SHOULD_NOT_BE_FULL:
    `assert_clk (cnt<16 |-> !full);
ERROR_FIFO_DID_NOT_GO_FULL:
    `assert_clk (cnt==15 && write && !read |=> full);
ERROR_FIFO_FULL__WRITE_CAUSED_FULL_FLAG_TO_CHANGE:
    `assert_clk (full && write && !read |=> full);
ERROR_FIFO_FULL__WRITE_CAUSED_WPTR_TO_CHANGE:
    `assert_clk (full && write && !read |=> $stable(wptr));
ERROR_FIFO_SHOULD_BE_EMPTY:
    `assert_clk (cnt==0 |-> empty);
ERROR_FIFO_SHOULD_NOT_BE_EMPTY:
    `assert_clk (cnt>0 |-> !empty);
ERROR_FIFO_DID_NOT_GO_EMPTY:
    `assert_clk (cnt==1 && read && !write |=> empty);
ERROR_FIFO_EMPTY__READ_CAUSED_EMPTY_FLAG_TO_CHANGE:
    `assert_clk (empty && read && !write |=> empty);
ERROR_FIFO_EMPTY__READ_CAUSED_RPTR_TO_CHANGE:
    `assert_clk (empty && read && !write |=> $stable(rptr));
ERROR_FIFO_WORD_COUNTER_IS_NEGATIVE:
    `assert_clk (not (cnt<0));
ERROR_FIFO_READWRITE_ILLEGAL_FIFO_FULL_OR_EMPTY:
    `assert_clk (write && read |=> !full && !empty);
```

Using Macros
Lines = 27
Characters = 1,151
(blank lines omitted)

Base line

Common Assertion Labels

Should Be Used With All Assertion Coding Styles


```
ERROR_FIFO_RESET_SHOULD_CAUSE_EMPTY1_FULL0_RPTR0_WPTR0_CNT0:
ERROR_FIFO_SHOULD_BE_FULL:
ERROR_FIFO_SHOULD_NOT_BE_FULL:
ERROR_FIFO_DID_NOT_GO_FULL:
ERROR_FIFO_FULL__WRITE_CAUSED_FULL_FLAG_TO_CHANGE:
ERROR_FIFO_FULL__WRITE_CAUSED_WPTR_TO_CHANGE:
ERROR_FIFO_SHOULD_BE_EMPTY:
ERROR_FIFO_SHOULD_NOT_BE_EMPTY:
ERROR_FIFO_DID_NOT_GO_EMPTY:
ERROR_FIFO_EMPTY__READ_CAUSED_EMPTY_FLAG_TO_CHANGE:
ERROR_FIFO_EMPTY__READ_CAUSED_RPTR_TO_CHANGE:
ERROR_FIFO_WORD_COUNTER_IS_NEGATIVE:
ERROR_FIFO_READWRITE_ILLEGAL_FIFO_FULL_OR_EMPTY:
```

The labels are basically a minimal set of comments

Same labeling effort for all coding styles, so *remove labels and measure coding effort again*

Assertion Coding Efforts

Labels & Blank Lines Omitted

```
property reset_rptr0_wptr0_empty1_full0_cnt0;
  @(posedge clk)
    (!rst_n |->
       (rptr==0 && wptr==0 && empty==1 && full==0 && cnt==0));
endproperty
property full_fifo_condition;
  @(posedge clk) disable iff (!rst_n)
      (cnt>15 |-> full);
endproperty
property not_full_fifo_condition;
  @(posedge clk) disable iff (!rst_n)
      (cnt<16 |-> !full);
endproperty
property fifo_should_go_full;
  @(posedge clk) disable iff (!rst_n)
      (cnt==15 && write && !read |=> full);
endproperty
property full_write_full;
  @(posedge clk) disable iff (!rst_n)
      (full && write && !read |=> full);
endproperty
property full_write_wptr_no_change;
  @(posedge clk) disable iff (!rst_n)
      (full && write && !read |=> $stable(wptr));
endproperty
property empty_fifo_condition;
  @(posedge clk) disable iff (!rst_n)
      (cnt==0 |-> empty);
endproperty
property not_empty_fifo_condition;
  @(posedge clk) disable iff (!rst_n)
      (cnt>0 |-> !empty);
endproperty
property fifo_should_go_empty;
  @(posedge clk) disable iff (!rst_n)
      (cnt==1 && read && !write |=> empty);
endproperty
property empty_read_empty;
  @(posedge clk) disable iff (!rst_n)
      (empty && read && !write |=> empty);
endproperty
property empty_read_rptr_no_change;
  @(posedge clk) disable iff (!rst_n)
      (empty && read && !write |=> $stable(rptr));
endproperty
property illegal_fifo_cnt_condition;
  @(posedge clk) disable iff (!rst_n)
      (not (cnt<0));
endproperty
property fifo_not_empty_not_full;
  @(posedge clk) disable iff (!rst_n)
      (write && read |=> !full && !empty);
endproperty
assert property (reset_rptr0_wptr0_empty1_full0_cnt0);
assert property (full_fifo_condition);
assert property (not_full_fifo_condition);
assert property (fifo_should_go_full);
assert property (full_write_full);
assert property (full_write_wptr_no_change);
assert property (empty_fifo_condition);
assert property (not_empty_fifo_condition);
assert property (fifo_should_go_empty);
assert property (empty_read_empty);
assert property (empty_read_rptr_no_change);
assert property (illegal_fifo_cnt_condition);
assert property (fifo_not_empty_not_full);
```

Property/Assert Property
Lines = 66
Characters = 2,067
(blank lines omitted)

371% more lines of code
249% more characters

```
assert property (@(posedge clk)
    (!rst_n |->
       (rptr==0 && wptr==0 && empty==1 && full==0 && cnt==0)));
assert property (@(posedge clk) disable iff (!rst_n)
      (cnt>15 |-> full));
assert property (@(posedge clk) disable iff (!rst_n)
      (cnt<16 |-> !full));
assert property (@(posedge clk) disable iff (!rst_n)
      (cnt==15 && write && !read |=> full));
assert property (@(posedge clk) disable iff (!rst_n)
      (full && write && !read |=> full));
assert property (@(posedge clk) disable iff (!rst_n)
      (full && write && !read |=> $stable(wptr)));
assert property (@(posedge clk) disable iff (!rst_n)
      (cnt==0 |-> empty));
assert property (@(posedge clk) disable iff (!rst_n)
      (cnt>0 |-> !empty));
assert property (@(posedge clk) disable iff (!rst_n)
      (cnt==1 && read && !write |=> empty));
assert property (@(posedge clk) disable iff (!rst_n)
      (empty && read && !write |=> empty));
assert property (@(posedge clk) disable iff (!rst_n)
      (empty && read && !write |=> $stable(rptr)));
assert property (@(posedge clk) disable iff (!rst_n)
      (not (cnt<0)));
assert property (@(posedge clk) disable iff (!rst_n)
  (write && read |=> !full && !empty));
```

Assert Property
Lines = 27
Characters = 1,175
(blank lines omitted)

93% more lines of code
98% more characters

```
`assert_async_rst(!rst_n |->
       (rptr==0 && wptr==0 && empty==1 && full==0 && cnt==0));
`assert_clk (cnt>15 |-> full);
`assert_clk (cnt<16 |-> !full);
`assert_clk (cnt==15 && write && !read |=> full);
`assert_clk (full && write && !read |=> full);
`assert_clk (full && write && !read |=> $stable(wptr));
`assert_clk (cnt==0 |-> empty);
`assert_clk (cnt>0 |-> !empty);
`assert_clk (cnt==1 && read && !write |=> empty);
`assert_clk (empty && read && !write |=> empty);
`assert_clk (empty && read && !write |=> $stable(rptr));
`assert_clk (not (cnt<0));
`assert_clk (write && read |=> !full && !empty);
```

Using Macros
Lines = 14
Characters = 593
(blank lines omitted)

Base line

50%-80% reduction in assertion coding effort by using macros !!

SystemVerilog Assertion Bind Files

Adding Assertions to Golden RTL Design or VHDL Design

- Golden RTL Design

Not permitted to add assertions to the Golden Design

- VHDL model in mixed SystemVerilog/VHDL simulation

VHDL compilers to not recognize SystemVerilog assertions

- Secretly *(or not-so-secretly)* instantiate SVA into the Golden RTL Design

Indirect instantiation using the `bind` command

Note: IEEE1800 SystemVerilog Standard *does not define* binding into VHDL models but many vendors allow this

Assertions Module

(Assertions Wrapper)

```
`define assert_clk(arg) \  ...
`define assert_async_rst(arg) \  ...

module pLib_fifo (
  input [7:0] dout, din,
  input [4:0] cnt,
  input [3:0] wptr, rptr,
  input       empty, read, full,
  input       write, clk, rst_n);

  ERROR_FIFO_RESET_SHOULD_CAUSE_EMPTY1_FULL0_RPTR0_WPTR0_CNT0:
    `assert_async_rst(!rst_n |->   ...

  ERROR_FIFO_SHOULD_BE_FULL:
    `assert_clk (cnt>15 |-> full);
  ...
endmodule
```

Move the FIFO assertions to a separate module

In general, all assertion module ports will be inputs

`module` - *ports* - `endmodule` used to "wrap" the assertions

Exact same assertion code as used directly in the design

Instantiated Assertions Module Using .*

```
module tb1;
  ...
  fifo1 u1   (.*);

  ...
endmodule
```

```
module pLib_fifo (
  input [7:0] dout, din,
  input [4:0] cnt,
  input [3:0] wptr, rptr,
  input       empty, read, full,
  input       write, clk, rst_n);
  ...
endmodule
```

FIFO assertions module

fifo1 instantiated into tb1

```
module fifo1 (
  output logic [7:0] dout,
  output logic       full, empty,
  input  logic       write, read, clk, rst_n,
  input  logic [7:0] din);
  logic [7:0] fifomem [0:15];
  logic [3:0] wptr, rptr;
  logic [4:0] cnt;
  ...

  pLib_fifo p1 (.*);
  ...
endmodule
```

pLib_fifo instantiated into fifo1

All pLib_fifo port names match fifo1 signal names so .* instantiation is possible

Golden RTL model - pLib_fifo instantiation *not allowed !!*

Assertions Module

How does the bind command work??
(next slide)

```
module tb1;
  ...
  fifo1 u1   (.*);
  bind fifo1: u1 pLib_fifo p1 (.*);
  ...
endmodule
```

(2) Use bind in the testbench to indirectly instantiate the assertions into the RTL model

```
module fifo1 (
  output logic [7:0] dout,
  output logic       full, empty,
  input  logic       write, read, clk, rst_n,
  input  logic [7:0] din);
  logic [7:0] fifomem [0:15];
  logic [3:0] wptr, rptr;
  logic [4:0] cnt;
  ...

  pLib_fifo p1 (.*);
  ...
endmodule
```

(1) Remove the instantiation from the Golden RTL model

Bind Command - Closer Look

The bind keyword is followed by the target module name (fifo1) that we are binding to

The file to be bound into the target module is the assertion file (pLib_fifo)

Box #1 → `bind fifo1: u1` `pLib_fifo p1 (.*);` ← Box #2

Optional argument : u1 to only bind to fifo1 u1 instance

The bound assertion file has the instance name p1

Omit the argument : u1 to bind to *all* fifo1 instances

All of the ports on the bound assertion file are connected to signals in the target module with the same name ((.*);)

Not all SystemVerilog implementations permit binding to a single instance

The box #2 code is the equivalent instantiation if not bound

2nd legal form *(omit module name)* → `bind fifo1: u1` `pLib_fifo p1 (.*);`

Assertion Binding Styles

```
module tb1;
  logic [7:0] dout;
  logic       full, empty;
  logic       write, read, clk, rst_n;
  logic [7:0] din;
  ...
  fifo1 u1   (.*);
  bind fifo1: u1 pLib_fifo p1 (.*);
  ...
endmodule
```

SystemVerilog style .* port binding

```
module fifo1 (
  ... dout,
  ... full, empty,
  ... write, read, clk, rst_n,
  ... din);
  ...
  ... wptr, rptr;
  ... cnt;
  ...
endmodule
```

```
module tb1a;
  logic [7:0] dout;
  logic       full, empty;
  logic       write, read, clk, rst_n;
  logic [7:0] din;
  ...
  fifo1 u1   (.dout(dout),   .full(full), .din(din), .empty(empty),
              .write(write), .read(read), .clk(clk), .rst_n(rst_n));
  bind fifo1: u1 pLib_fifo p1 (
              .dout(dout),   .full(full), .din(din), .empty(empty),
              .write(write), .read(read), .clk(clk), .rst_n(rst_n),
              .wptr(wptr),   .rptr(rptr), .cnt(cnt));
  ...
endmodule
```

Verilog-2001 style named port binding

Assertions Module Instantiated In fifo2

```
module tb2;
  ...
  fifo2 u1   (.*);



  ...
endmodule
```

```
module pLib_fifo (
  input [7:0] dout, din,
  input [4:0] cnt,
  input [3:0] wptr, rptr,
  input       empty, read, full,
  input       write, clk, rst_n);
  ...
endmodule
```

```
module fifo2 (
  output logic [7:0] dout,
  output logic       full, empty,
  input  logic       write, read, clk, rst_n,
  input  logic [7:0] din);
  logic [7:0] fifomem [0:15];
  logic [3:0] qptr, iptr;
  logic [4:0] word_counter;
  ...
  pLib_fifo p1 (.wptr(qptr), .rptr(iptr),
                .cnt(word_counter), .*);
endmodule
```

New names for `wptr` and `rptr`

New name for `cnt`

Named port connections required for non-matching port names

Assertions Module Bound To fifo2

```
module tb2;
  ...
  fifo2 u1   (.*);
  bind fifo2: u1
    pLib_fifo p1 (.wptr(qptr), .rptr(iptr),
                  .cnt(word_counter), .*);
  ...
endmodule
```

```
module pLib_fifo (
  input [7:0] dout, din,
  input [4:0] cnt,
  input [3:0] wptr, rptr,
  input       empty, read, full,
  input       write, clk, rst_n);
  ...
endmodule
```

Named port connects required for non-matching port names

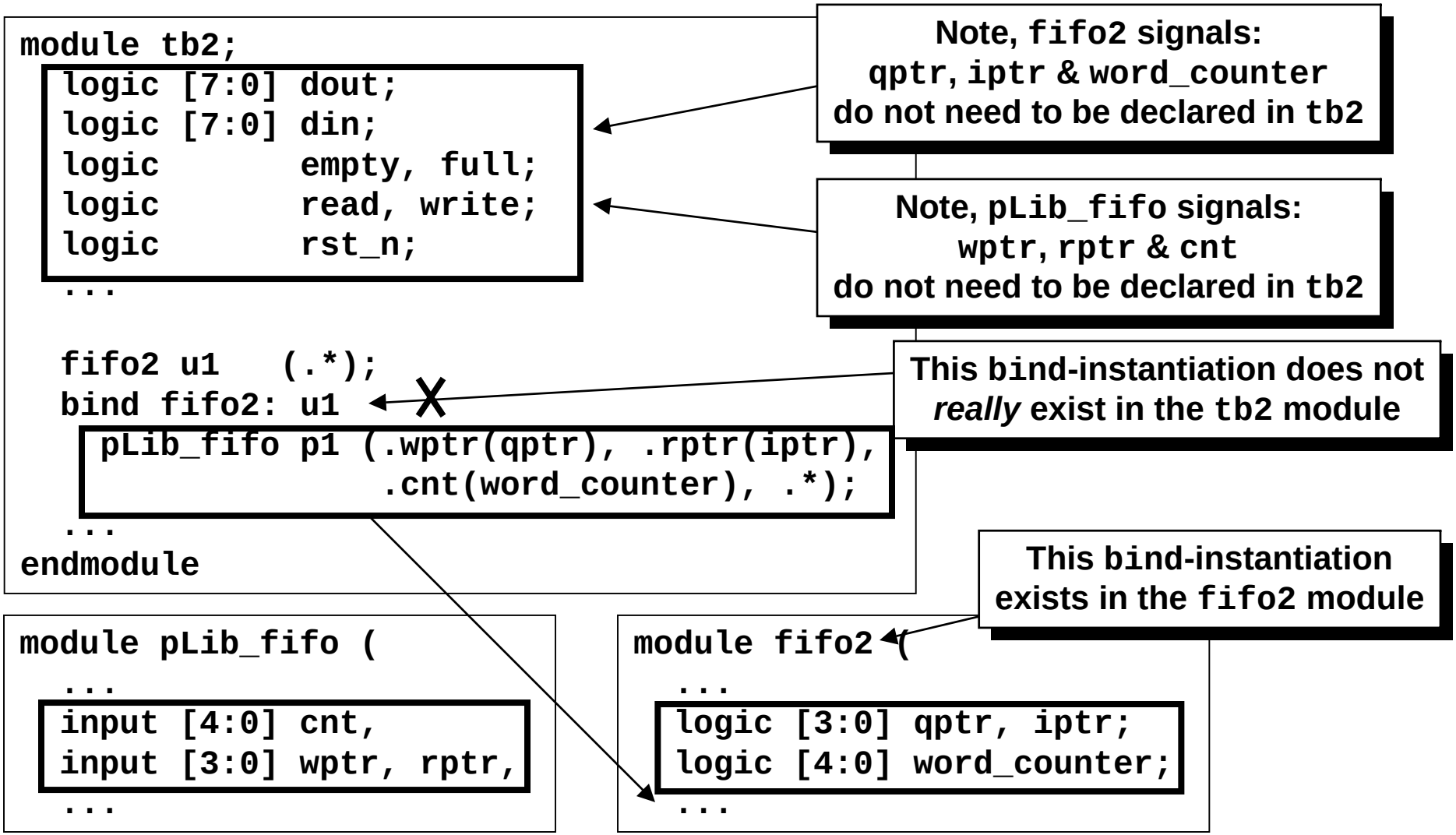
```
module fifo2 (
  output logic [7:0] dout,
  output logic       full, empty,
  input  logic       write, read, clk, rst_n,
  input  logic [7:0] din);
  logic [7:0] fifomem [0:15];
  logic [3:0] qptr, iptr;
  logic [4:0] word_counter;
  ...

endmodule
```

New names for `wptr` and `rptr`

New name for `cnt`

Bind File Signal Declarations?



Bind File Use & Abuse

Exception: John Dickol's 3rd party memory loader & reader
(see paper for more info)

- Guideline: *safest* usage dictates that bind-file only has inputs

`bind` originally intended to add assertions without modifying the original RTL code

`bind` was expanded to add simple non-obtrusive verification capabilities

---

- Potential problems & abuses ← ... as reported by EDA vendors
  - Bind-file invisibility ← Bind-file is not in the code
    - Could cause unexpected interactions between design and bound modules
    - *Again - safest when bind-module only has inputs*
  - Nested binding is illegal ← Cannot bind assertion-file to file bound to another module **
  - Complex design structure created through bind command
    - EDA report: bind-instantiation of an interface connected to hierarchically referenced interface ports
    - `bind` was not intended to create a structurally valid design

Conclusions

- Verbose SVA coding styles discourage SVA use by design engineers
- Concise macros reduce assertion coding efforts
- The easier it is to add assertions, the more design engineers will add assertions
- SVA macros will lead to more assertion usage by designers
- SVA bind files allow indirect insertion of SVA into a golden RTL model without modifying the RTL source file

50%-80% reduction in assertion coding effort by using macros !!

Engineers like "easy!"

Spend less quality time with your verification engineer!

Also allows binding of SVA to VHDL models

Do not abuse the `bind` command!!

Acknowledgements - *Great reviews and feedback*
Heath Chambers - HMC Design Verification
Kelly Larson - MediaTek
John Dickol - MediaTek
Anders Nordstrom - Formal Verification Expert

SystemVerilog Assertions Design Tricks and SVA Bind Files

CLIFFORD E. CUMMINGS
SUNBURST DESIGN, INC.
CLIFFC@SUNBURST-DESIGN.COM
WWW.SUNBURST-DESIGN.COM

Life is too short for bad or boring training!

World Class Verilog & SystemVerilog Training