



# **System Verilog Assertions**

SE303 – Conception des  
systèmes sur puces (SoC)

Ulrich Kühne  
29/11/2017





# Outline

Introduction

Sequences

Strength & Infinity

Advanced Operators



# A Practical Verification Language?

- LTL and CTL have emerged from **theoretical** interest
- Bound to specific complexity classes and equivalence notions
- Nested CTL/LTL properties are **hard to understand**
- Subtle semantic differences

$$\mathbf{FX} p \equiv \mathbf{XF} p \equiv \mathbf{AX AF} p \not\equiv \mathbf{AF AX} p$$

$$\mathbf{FG} p \not\equiv \mathbf{AF AG} p$$

# System Verilog Assertions

```
module monitor( foo.MONITOR I );

property slave_data_notunknown_when_ready;
  @(posedge I.clk)
    I.ready | -> $isunknown(I.s) == 0;
endproperty

assert_slave_data_notunknown_when_ready: assert property (slave_data_notunknown_when_ready)
  else $error("%m: ready is asserted but data from slave is non valid");

property slave_ready_until_valid;
  @(posedge I.clk)
    $rose(I.ready) | -> I.ready throughout I.valid [->1]; //ou I.ready [*0:$] ##1 I.valid;
endproperty

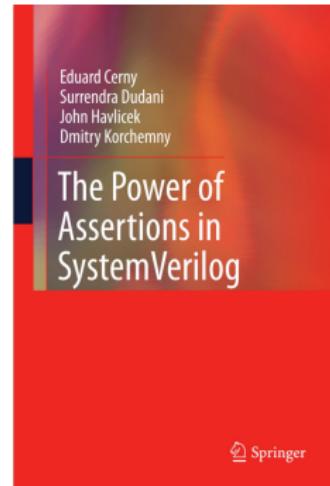
assert_slave_ready_until_valid: assert property(slave_ready_until_valid)
  else $error("%m: slave's ready must be held until valid is set");

property slave_data_held_when_ready;
  bit [7:0] s;
  @(posedge I.clk) disable iff (I.nrst == 0)
    (I.ready && !I.valid , s = I.s) | => s == I.s; //ou $stable(I.s);
endproperty

assert_slave_data_held_when_ready: assert property(slave_data_held_when_ready)
  else $error("%m: data must be held stable when slave is ready");
endmodule
```

# System Verilog Assertions

- Industrial standard (IEEE 1800-2012)
- Embedded in SystemVerilog HDL
- Superset of LTL
- Sequences and regular expressions
- Supports simulation and formal verification





# Basic Property Structure

```
// basic property structure
property foo;
  @(posedge clk) disable iff (rst)
    expr;
endproperty // foo

// verification directives
assert_foo: assert property(foo);
assume_foo: assume property(foo);
```

# Past Values and Value Changes

- Value of a signal in the preceding cycle:

`$past(a)`

- Shortcut for rising edge:

`$rose(a)` is equal to `!$past(a) && a`

- Shortcut for falling edge:

`$fell(a)` is equal to `$past(a) && !a`

- Shortcut for stable signal:

`$stable(a)` is equal to `$past(a) == a`

- Shortcut for changed signal:

`$changed(a)` is equal to `$past(a) ^ a`



# Outline

Introduction

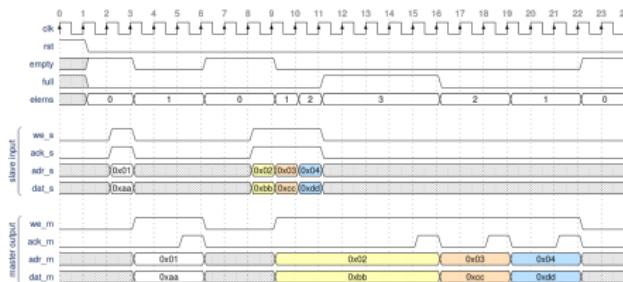
Sequences

Strength & Infinity

Advanced Operators

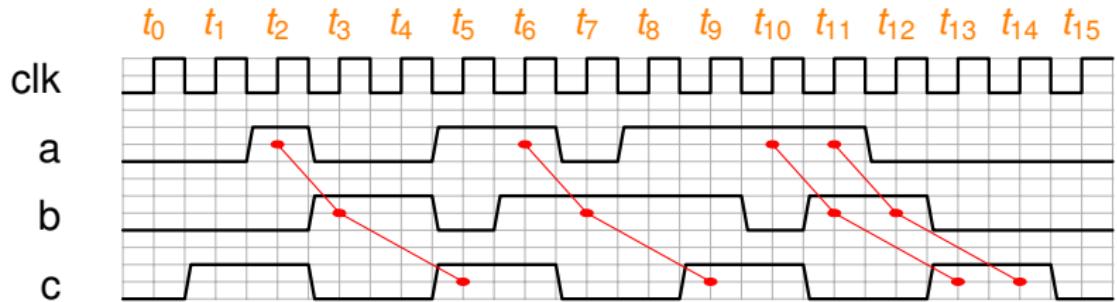
# Sequentially Extended Regular Expressions (SERE)

- Typical use case: Chains of events
- Awkward to describe in LTL
- Intuitive description by **regular expressions**
- Syntax resembles known languages (bash, Python, ...)



# Sequences

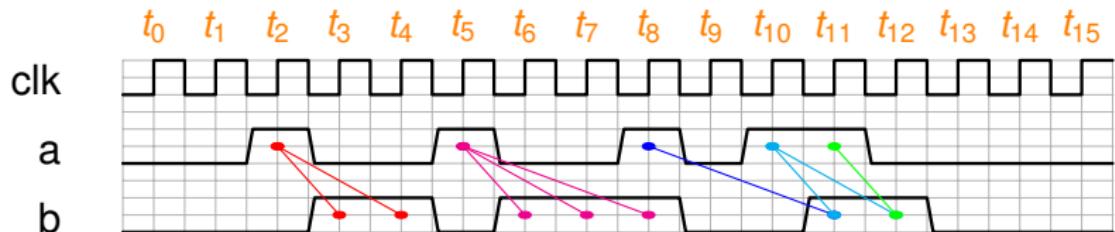
a ##1 b ##2 c



- Find all matching cycles...

# Variable Delay

```
a ##[1:3] b
```



- Find all matches...



# Repetition

- Consider sequence:

a ##1 a ##1 a ##1 b ##1 b

- Shortcut for repeating sequence:

a[\*3] ##1 b[\*2]

- Variable repetition:

a[\*1:3]

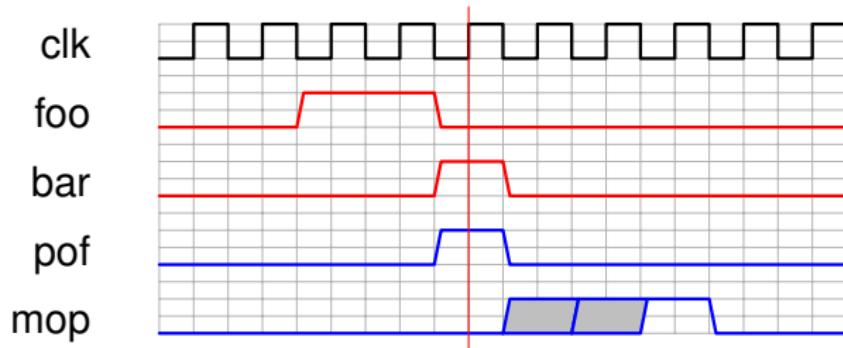
# Assertion Semantics

```
property foo;  
  @(posedge clk)  
    a ##[1:3] b;  
endproperty  
  
assert_foo: assert property(foo);
```

- What are we actually verifying here?
- Sequence `a ##[1:3] b` must match in **all cycles**
- Implicit **always** operator (**G** in LTL)

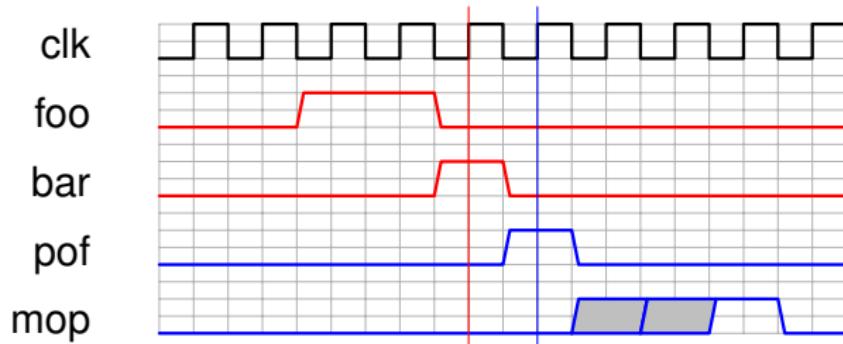
# Suffix Implication

```
// suffix implication
foo ##1 bar |-> pof ##[1:3] mop
```



# Non-Overlapping Suffix Implication

```
// non-overlapping suffix implication
foo ##1 bar |=> pof ##[1:3] mop
```





# Example

Prop. 1: “Whenever signal rdy is asserted, it must stay asserted for 5 clock cycles”

```
property rdy_stable;  
  @(posedge clk)  
    !rdy ##1 rdy |=> rdy[*4];  
endproperty
```



# Outline

Introduction

Sequences

Strength & Infinity

Advanced Operators



# Infinity

- Special symbol  $\$$  for infinity
- Can be used in variable delay and repetition
  - $a[*]$  is a shortcut for  $a[*0:\$]$
  - $a[+]$  is a shortcut for  $a[*1:\$]$

**Exercise:** What is the meaning of this sequence?

```
(start ##1 busy[*] ##1 done)[+]
```

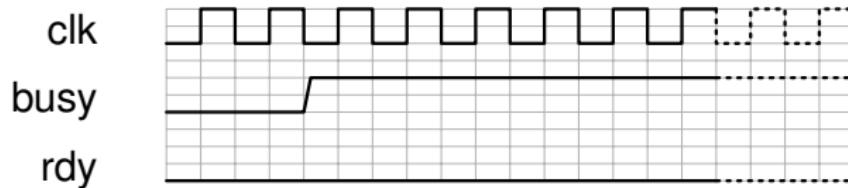
Prop. 2: “Whenever signal busy is asserted, rdy must be asserted eventually.”

```
property rdy_after_busy;  
  @(posedge clk)  
    busy | -> ##[0:$] rdy;  
endproperty
```

This property is wrong!

- This assertion has **no counter-example**

```
busy | -> ##[0:$] rd़y
```





# Strength

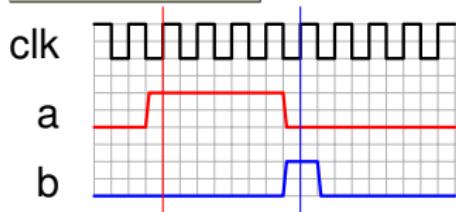
- Use of **strong**(...) operator
- Enforces a match before the end of evaluation (which is infinity in formal verification)
- Weak and strong versions of many operators

```
A1: assert property (busy |-> ##[0:$] rdy);  
A2: assert property (busy |-> strong( ##[0:$] rdy ));  
A3: assert property (busy |-> eventually rdy);  
A4: assert property (busy |-> s_eventually rdy);
```

# Until

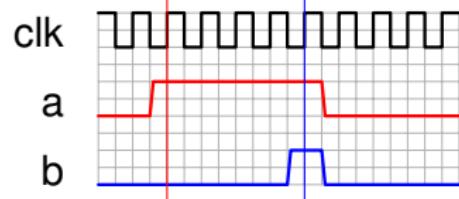
a **until** b

a **s\_until** b

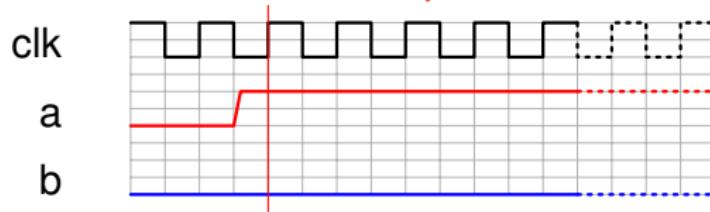


a **until\_with** b

a **s\_until\_with** b



Attention: Weak until operators allow infinite wait!





# Outline

Introduction

Sequences

Strength & Infinity

Advanced Operators

# Goto Repetition

Prop. 3: “After signal write is serviced by ack,  
signal ready should be asserted.”

```
(write ##1 !ack[*] ##1 ack) |=> ready
```

Prop. 4: “After signal wr\_burst is serviced **twice** by ack,  
signal ready should be asserted.”

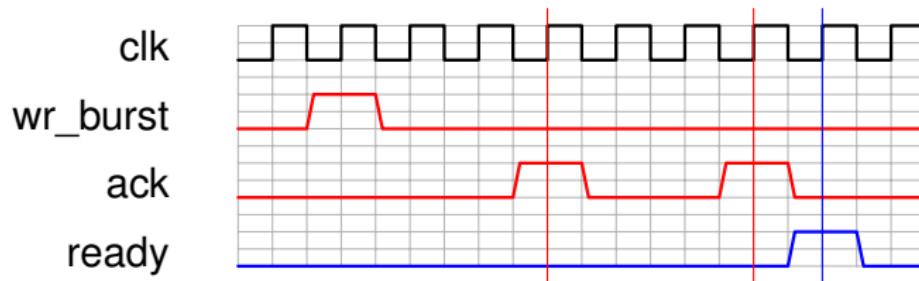
```
(wr_burst ##1 !ack[*] ##1 ack ##1 !ack[*] ##1 ack) |=> ready
```

```
(wr_burst ##1 (!ack[*] ##1 ack)[*2] ) |=> ready
```

```
wr_burst ##1 ack[>-2] |=> ready
```

# Goto Repetition

```
wr_burst ##1 ack[->2] |=> ready
```



# Within / Throughout

Prop. 5: “Throughout the whole burst cycle, the signal ready should be low.”

```
!ready throughout (wr_burst ##1 ack[->2])
```

Prop. 6: “Within a granted bus cycle, a write transaction should be completed.”

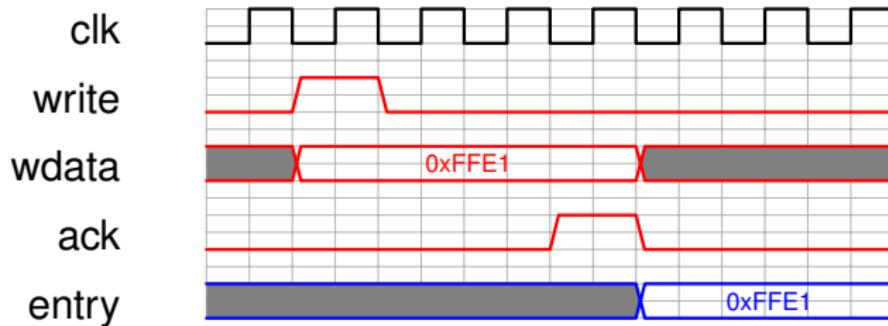
```
(write ##1 ack[->1]) within (gnt ##1 !gnt[->1])
```

This property is (probably) wrong!

```
(write ##1 ack[->1] ##1 1) within (gnt ##1 !gnt[->1])
```

# Local Variables

**Prop. 7:** “After a completed write transaction, the value of wdata is stored in the register entry.”



# Local Variables

Prop. 7: “After a completed write transaction, the value of wdata is stored in the register entry.”

```
property foo;  
  logic[15:0] tmp;  
  @ (posedge clk)  
    (write, tmp = wdata) ##1 ack[ ->1 ] |=>  
      entry == tmp;  
endproperty
```

# Practical Exercise

- Formalization of a textual specification
- Implementation & verification with *qformal*
- See exercise on website  
<https://sen.enst.fr/verification-formelle>
- Until December 11: Complete the formalization!



# References I



Cerny, E., Dudani, S., Havlicek, J., and Korchemny, D. (2015).  
*SVA: The Power of Assertions in SystemVerilog.*  
Springer.