Challenges of VHDL X-propagation Simulations

Karthik Baddam and Piyush Sukhija

Overview

- Genesis of the X
- Existing solutions
- Why simulation based X-propagation
- VCS X-propagation for VHDL
- False negatives
- X-propagation debug
- When to stop X-propagation simulations

Genesis of X

- X is a modelling construct and not present in real silicon
  - In real silicon, X can take the value of either logic 0 or 1
  - The problem with X is there is a modelling to silicon mismatch

- X means unknown or don't care design state
  - Simulation behavior is described in SV/VHDL LRM
  - In VHDL these are W, -, U, X, Z

Sources of X

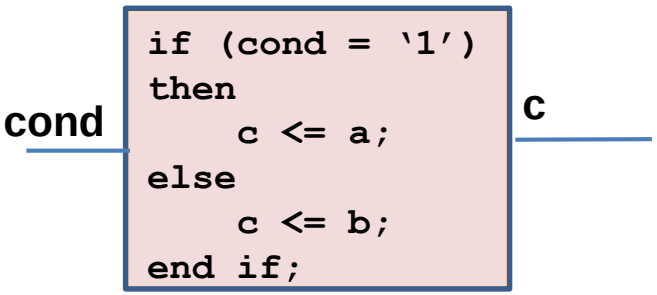
- Power aware simulations, isolation cells, power island
- Explicit X assignment in designs
- Implicit X sources that includes uninitialized flops, latches, memories and floating signals
- Functional violations such as floating buses, bus contention, range overflow, divide by 0
- Multiple drivers

Is X Safe?

- X ***can*** indicate functional bugs and they should get reviewed
- Some of the X sources can be caught with lint tools
- Some non-reset flop related X are safe, some can be bugs
- In RTL verification environments, the X needs to propagate to an observable point in order to be detected!
- Can X be caught at RTL ?
  - Current LRM compliant tools are X Optimistic

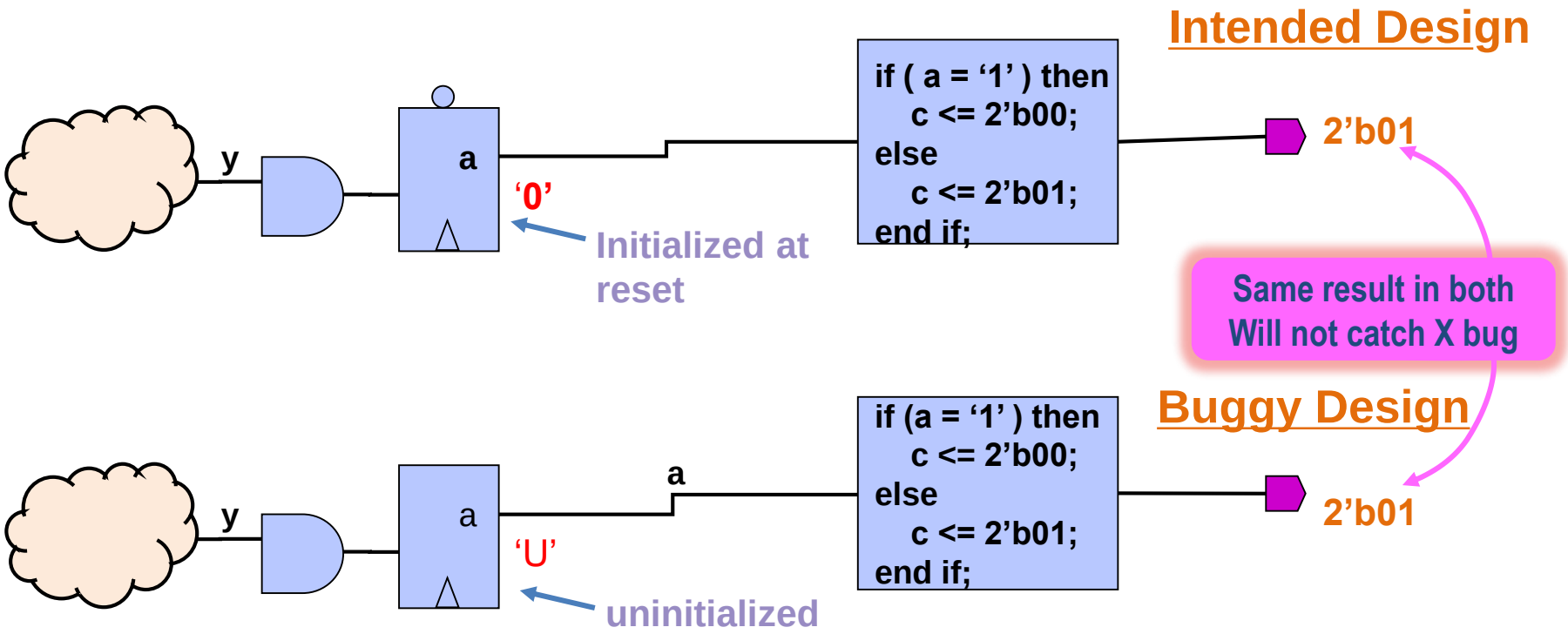
RTL X-Optimism Example: if-else

- Simulation model yields fewer unknown values than are really possible



cond	a	b	c (LRM)	c (silicon)
X	0	0	0	0
X	0	1	1	0/1
X	1	0	0	0/1
X	1	1	1	1

Example – Typical X Bug Escape in RTL



Existing solutions

- Gate level simulations
  - Traditionally used approach to find X issues (among other things)
  - Too pessimistic
  - Debug is difficult as design is now in netlist form
- Random initial values and 2 state simulation
  - Ensuring all combinations of initial values to test is difficult
- Formal model checking tools
  - Exhaustive and efficient, however needs functional design constraints if not already present
- Simulator specific X-propagation options
  - Non LRM compliant (details in the next slide)

XProp Propagation Modes

Example

```
if (cond = '1') then
    c <= a;
else
    c <= b;
```

cond	a	b	c (LRM)
X	0	0	0
X	0	1	1
X	1	0	0
X	1	1	1

XProp Propagation Modes

Example

```
if (cond = '1') then
    c <= a;
else
    c <= b;
```

cond	a	b	c (LRM)	c XProp Pessimistic
X	0	0	0	X
X	0	1	1	X
X	1	0	0	X
X	1	1	1	X

XProp Propagation Modes

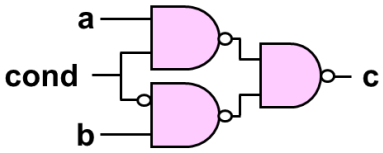
Example

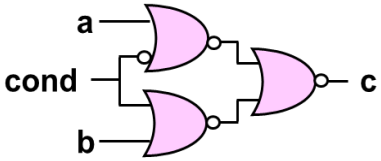
```
if (cond = '1') then
    c <= a;
else
    c <= b;
```

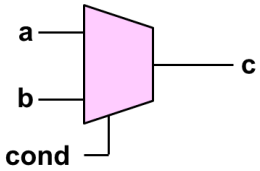
c = resolve_function (a,b)

cond	a	b	c (LRM)	c XProp Pessimistic	c XProp Realistic
X	0	0	0	X	0
X	0	1	1	X	X
X	1	0	0	X	X
X	1	1	1	X	1

Simple *if* Statement comparison








```
if(cond) then
    c <= a;
else
    c <= b;
end if;
```

cond	a	b	c (RTL)	c (HW)	c Realistic XProp	c Pessimistic XProp	GLS c (1)	GLS c (2)	GLS c (3)
X	0	0	0	0	0	X	0	X	0
X	0	1	1	0/1	X	X	X	X	X
X	1	0	0	0/1	X	X	X	X	X
X	1	1	1	1	1	X	X	1	1

Why simulation based X-propagation ?

- If simulation based verification environment
  - then using X-propagation on top ensure the sign-off coverage is validated for X
- Before gate level simulation
  - as it is more efficient to catch X bugs

- Key points about simulation based X-propagation
  - As good as tests and checks
  - Not a replacement for GLS in principle, but a better approach to find X

X-propagation for VHDL

- Conditional statements
  - All conditional statements should be X-propagation aware
  - These include if, case, when-select

X-propagation for VHDL -- Enums

- Enumerate types in VHDL default to two-state type behaviour
- Need to add an explicit X type to each enumerated type
- Branch statements should support these types appropriately

X-propagation for VHDL -- Integers

- Common to use integers in data path ops
- Problem is that X is lost when converting to integers
- Two solutions
  - Treat integers as true four state data type
  - Add an explicit X type for integers (similar to enumeration types)
- Divide by zero; should not stop the simulation but result in X

X-propagation for VHDL – Array Index

- In VHDL, arrays and multi-dimensional arrays are declared using types
  - Index of such arrays have to be integers
- Converting to integer is losses X
- Reading from an array, and writing to an array, with X index should propagate X

X-propagation for VHDL – Array Index

- Writing to an array with X index should merge the new value with existing value

X-propagation for VHDL – Array Index

- Reading from an array with X index should merge all the elements of the array, if the array is power of 2.
- If the array is non-power of 2, then it should result in X

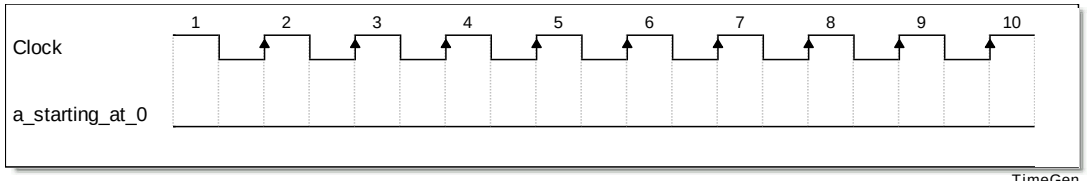
X-propagation for VHDL -- Data ops

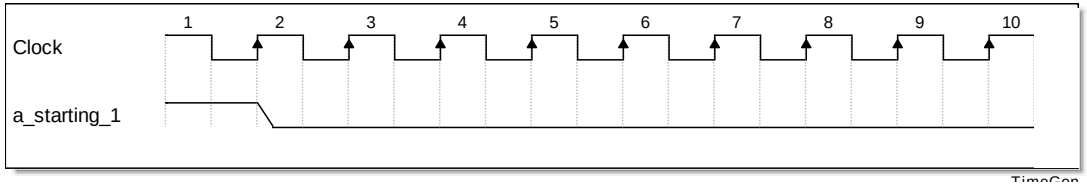
- Such as add, sub, mul on std_logic types should be X-propagation aware
  - Consider the addition involving two 4 bit vectors (`z <= a + b;`, result of the addition, z, will contain `x` if any of the operands contain `x`
- Useful in address calculations that need to be X-propagation aware

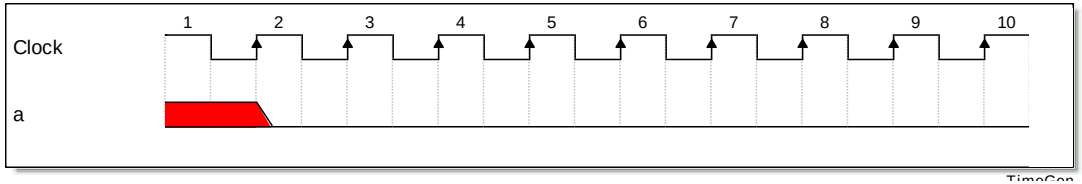
Signal name	Example 1	Example 1	Example 3
a	000X	000X	010X
b	0X00	0X0X	0X0X
z X-propagation aware	00X0X	00XXX	0XXXX
z default	XXXXX	XXXXX	XXXXX

Self gating

- Re-convergence problem (similar to GLS)
- Pessimistic than silicon behaviour








```
process (clk)
if (rising_edge(clk)) then
    if (a /= '0') then
       a <= 0;
    end if;
end if;
```

X-propagation debug

- Flip Flop output transitioning to X debug aid
  - Report flops that transition from known state to X
  - Some block do this on purpose, so need to waive depending on design
  - Helps in most cases

X-propagation debug

- Lockup Debug Methodology
  - Get a list of signals that have X at the end of simulation
  - Remove signals that did not have any event
  - Remove signals that are also unknown in non X-propagation simulation
  - Debug each signals in this list to root cause

X-propagation debug

- Is fully automated X-propagation debug possible ?
  - We encourage colleagues in this industry to investigate

When to stop X-propagation sim

- Classic coverage question
- If possible, rerun all functional tests with X-propagation
- But its not always possible
  - When using acceleration as primary sign-off
  - In SoC like environment where IPs are assembled, only integration tests are available

Flip Flop X State Coverage

- Check the state of every flip flop at the end of simulation, if any of the flop is X, then save it to a list in a file
- For each test, save such a list
- Intersection of all the lists will produce flip flops that were either not tested, or ended in an X

Flip Flop X State Coverage

- Intersection needs to be empty, if not more tests should be added targeting flops in this list
- This approach is similar to code coverage, in that, it doesn't prove lack of X bugs but it does highlight lack of tests to cover any potential bugs.

Questions ?