I’m Still In Love With My X!
(but, do I want my X to be an optimist, a pessimist, or eliminated?)

by

Stuart Sutherland
SystemVerilog Wizard (and instructor/consultant)
Sutherland HDL, Inc.
If X means something is wrong...

Why am I still in love with my X?

To answer this question, this paper examines:

- Where did my 1 (or 0) become my X?
- An optimistic X – is that good or bad?
- A pessimistic X – is that any better?
- Eliminating my X by using 2-state simulation models
- Eliminating some of X with 2-state data types
- Detecting and stopping my X at the door
- Guidelines on minimizing problems with my X
Introducing My X

- SystemVerilog uses 4-state logic
  - 0, 1 and Z are an abstraction of real silicon values
  - X is a simulation-only value – it does not represent silicon

- In simulation, X can be used to represent:
  - Uninitialized – variables that have not been set to a value
  - Don’t care – the designer does not care if silicon has 0 or 1
  - Ambiguous – simulation cannot determine whether actual silicon would have a 0 or a 1

- A major concern is how X values propagate through design logic
How did my one (or zero) become my X?

- It might be surprising how many ways you can get an X!
  - Uninitialized 4-state variables
  - Uninitialized registers and latches
  - Low power logic shutdown or power-up
  - Unconnected module input ports
  - Multi-driver conflicts (bus contention)
  - Operations with an unknown result
  - Out-of-range bit-selects and array indices
  - Logic gates with unknown output values
  - Setup or hold timing violations
  - User-assigned X values in hardware models
  - Testbench X injection

The paper explains all of these in more depth.
An Optimistic X... Is that good or bad?

**Optimism**: an inclination to put the most favorable construction upon actions and events or to anticipate the best possible outcome – Merriam-Webster online dictionary

- In simulation, *X-optimism* is when there is some uncertainty on an input to an expression or logic gate, but simulation comes up with a known result instead of an X.

X-optimism can hide design problems!

Is my design really working without any problems?

I'm Still In Love With My X – Stuart Sutherland, Sutherland HDL, Inc.
A Pessimistic X... Is that any better?

**Pessimism:** an inclination to emphasize adverse aspects, conditions, and possibilities or to expect the worst possible outcome – Merriam-Webster online dictionary

- In simulation, *X-pessimism* is when there is some uncertainty on an input to an expression or logic gate, and simulation passes it through as an unknown result (an X)

![](image)

X-pessimism does not always behave like actual silicon!

But real silicon would know the answer!

---

I'm Still In Love With My X – Stuart Sutherland, Sutherland HDL, Inc.
SystemVerilog is Sometimes an Optimist and Sometimes a Pessimist

**X-optimistic constructs**
- if...else statements
- case statements without a default-X assignment
- casex, casez, case...inside
- Certain bitwise, unary reduction, and logical operators
- and, nand, or, nor primitives
- Array index with X or Z bits for write operations
- Net data types
- posedge and negedge edge sensitivity
- Some user-defined primitives

**X-pessimistic constructs**
- if...else with X assignments
- ?: conditional operator
- case statements with X assignments
- Edge-sensitive X pessimism
- Certain bitwise, unary reduction, and logical operators
- Equality, relational, and arithmetic operators
- Bit-select, part-select, array index on right-hand side of assignments
- Shift operations
- Some user-defined primitives
X-Optimism is Essential!

- X-optimism allows simulation to predict and act like actual silicon behavior when there are ambiguous conditions.

---

ambigious data, represented with X

0 0 0

What would happen if the AND gate was pessimistic, and propagated the ambiguous X value of data?

---

The AND ( & ) operator and and primitive are X-optimistic

- A 0 and’ed with anything will result in 0
- The flip-flop will correctly reset when data is ambiguous

---

Without X-optimism, it would be almost impossible to simulate synchronous resets! (the paper discusses several other circumstances where X-optimism is essential)
X-Optimism Is Not Always a Good Thing!

- **if...else** decision statements are X-optimistic

```verbatim
counter_comb begin
if (sel) y = a;
else y = b;
end
```

Execute the **if** branch if the control condition evaluates to true (1'b1)

Execute the **else** branch for all other conditions (1'b0, 1'bX)

- **if...else** optimism does not match silicon behavior!

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>sel</td>
<td>a</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
</tr>
</tbody>
</table>

DANGER! Only some of the possible results that can occur in silicon have been verified in RTL simulation!

Are you willing to build your chip without verifying all possible conditions?
# Case Statements Without a Default Branch

- **case** statements without a **default** branch are X-optimistic, but do not match silicon behavior

```verilog
always_comb begin
    case (sel)
        1'b1: y = a;
        1'b0: y = b;
    endcase
end
```

- **sel** is the "**case expression**"
- **1'b1** and **1'b0** are "**case items**"

- Only the first matching case item is executed
- If no case item matches, and there is no **default** statement, no branch is executed

<table>
<thead>
<tr>
<th>sel</th>
<th>a</th>
<th>b</th>
<th>previous value of y</th>
<th>RTL simulation</th>
<th>silicon behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0 or 1</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0 or 1</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 or 1</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0 or 1</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**DANGER!** Only some of the possible results that can occur in silicon have been verified in RTL simulation, and sometimes the wrong value has been verified!
**Case Statements With a Known-Value Default Branch**

- **case** statements with a known-value **default** branch are also X-optimistic, and also do not match silicon behavior

```vhdl
always_comb begin
    case (sel)
    2'b01: y = a;
    2'b10: y = b;
    default: y = c;
    endcase
end
```

<table>
<thead>
<tr>
<th>sel</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>RTL simulation</th>
<th>silicon behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0 or 1</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0 or 1</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0 or 1</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 or 1</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0 or 1</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0 or 1</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

If no **case items**, the **default** branch is executed

**DANGER!** Only some of the possible results that can occur in silicon have been verified in RTL simulation

- **casez**, **casex** and **case...inside** are also X-optimistic
Is My X Hiding From Me?

- X-optimism can cause verification to miss that an ambiguous condition has occurred within a design
  - Verification is typically done on design outputs
  - An X that occurs inside a module might not propagate to the module outputs

```
always_ff
if... else...
always_comb
case...
```

X-Optimism can hide internal design bugs
Is it Better to Have a Pessimistic X?

- X-pessimism always propagates X values
  - Many SystemVerilog constructs are naturally pessimistic
  - `if...else` and `case` statements can be coded to be pessimistic

With X-pessimism, verification is more likely to detect ambiguous conditions that occur within a design.
X-Pessimistic
if...else and case Statements

- Decision statements can be deliberately coded to propagate X values

```verilog
always_comb begin
  if (sel)    y = a;
  else // synthesis translate_off
    if (!sel)
      // synthesis translate_on
      y = b;
    // synthesis translate_off
  else      y = 'x;
    // synthesis translate_on
end
```

```verilog
always_comb begin
  case (sel)
    1'b1:    y = a;
    1'b0:  y = b;
    default: y = 'x;
  endcase
end
```

with `if...else`, the extra code to propagate an X in the control condition is unknown is not synthesizable, and must be hidden from synthesis compilers.

with `case`, a default assignment of X will propagate an X if the case expression is unknown, and is synthesizable.

CAUTION: A default X assignment can cause synthesis to perform certain gate minimizations that might not be appropriate for the design.
The Optimistically Pessimistic Conditional Operator

- The ?: conditional operator finds a balance between X-optimism and X-pessimism

```
always_comb begin
  y = sel? a : b;
end
```

<table>
<thead>
<tr>
<th>sel</th>
<th>a</th>
<th>b</th>
<th>simulation</th>
<th>silicon</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>0 or 1</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>0 or 1</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- Some engineers recommend using ?: instead of if...else
- This author disagrees!
  - Complex decisions using ?: are difficult to write/debug/reuse
  - The conditional operator is not perfect… it can sometimes propagate an incorrect or incomplete optimistic result
X-Pessimism Causes Problems

- Pessimistically propagating X values is not always desirable
  - Must trace Xs through many lines of code and clock cycles
  - Can propagate Xs when silicon would have known values
  - Can lock-up simulation even though silicon would work fine

In simulation...
1. The flip-flop starts with an X
2. The X feeds back to the inverter
3. The inverter is pessimistic and propagates an X
4. The X gets clocked into the flop-flop
5. ...

In silicon...
1. The flip-flop starts with an ambiguous 0 or 1
2. The 0 or 1 feeds back to the inverter
3. The inverter output becomes 1 or 0
4. The 1 or 0 gets clocked into the flop-flop
5. ...

Pessimistic clock divider – this will lock up with an infinite X!
X-Optimism Summary

- **X-optimism is necessary**
  - Simulation will not work in some design conditions without X-optimism

- **X-optimism can be hazardous**
  - Some ambiguous conditions might not be simulated

- **X-optimism hides design bugs**
  - Ambiguous values might not propagate to where verification observes the ambiguity (an X does not propagate)

The paper discusses several more X-optimistic constructs in detail:

- `casex`, `casez`, `case...inside`
- Bitwise, unary reduction, and logical operators
- `and`, `nand`, `or`, `nor` primitives
- Array index with X or Z bits for write operations
- Net data types
- `posedge` and `negedge` edge sensitivity
- User-defined primitives
X-Pessimism Summary

- **X-pessimism is not necessary**
  - Simulation will work fine (and better) without propagating Xs

- **X-pessimism can help verification**
  - Ambiguous values deep within a design will propagate to where verification can observe the ambiguity

- **X-pessimism can cause problems**
  - Difficult debugging
  - False X propagation
  - Simulation X-state lockup

The paper discusses several more X-pessimistic constructs in detail:

- Edge-sensitive X pessimism
- Bitwise, unary reduction, and logical operators
- Equality, relational, and arithmetic operators
- Bit-select, part-select, array index on right-hand side of assignments
- Shift operations
- User-defined primitives
Eliminating My X

- 2-state simulation modes and data types get rid of all Xs
  - Advantages:
    + Makes simulation 100% optimistic – No X to worry about
    + Simulation and synthesis see RTL code the same way
    + Behaves more like silicon, which only has 0 or 1 (never X)
    + Faster simulation – simpler algorithms, less virtual memory
  - Disadvantages:
    - Verification cannot check for ambiguous conditions
    - Simulation can only use 0 or 1 when there is an ambiguity
      - Other possible values are not verified in RTL simulation
    - Lose any and all indications of silicon ambiguity

Your X is your friend – Use 4-state simulation and keep your X around!
A Better Way...
Stopping My X at the Door

- **X-propagation has disadvantages**
  - **Difficult debugging** – might need to trace an X back through many lines of code and many clock cycles
  - **False X propagation** – in real silicon, many ambiguities resolve to known values or are don’t cares
  - **Simulation X-state lockup** – synchronous resets, clock dividers, feedback loops get stuck-at-X without X-optimism

- **A better way is to detect Xs, rather than propagating Xs**
  - A simple SystemVerilog immediate assertion will do the job!

```verilog
always_comb begin
    assert (!$isunknown(sel)) else $error("%m, sel = X");
    if (sel) y = a;
    else     y = b;
end
```

Every decision statement should have an assertion to trap X values – it’s easy!

Design engineers should add these assertions as RTL code is created (read the paper for more details)
Coding Recommendations to Minimize X Problems

- These 4 coding guidelines will minimize problems with an X
  - 2-state versus 4-state guidelines
    - Use 4-state data types almost everywhere
      - Your X is your friend – let it tell you when there is ambiguity
    - Exception: Use 2-state types for random stimulus variables
  - Register initialization guidelines
    - Use simulator options or UVM to “power-up” hardware registers with random 0 or 1 values instead of the default of X
  - X-assignment guidelines
    - Don’t force decisions to be pessimistic by assigning an X
  - Trapping X guidelines
    - Use X-detect assertions on all decision statements
**Conclusions**

- **Your X is your best friend!**
  - An X indicates there is a design ambiguity where simulation cannot determine what logic value would be in actual silicon

- **X-optimism propagates a 0 or 1 when an X occurs**
  - X-optimism is essential in order for 4-state simulation to work
  - Many SystemVerilog constructs are optimistic
  - X-optimism can—and will—hide ambiguity bugs

- **X-pessimism propagates an X when an X occurs**
  - X-pessimism will not hide ambiguity bugs
  - X-pessimism causes simulation problems, such as X lock-up

- **Stop your X at the door using X-detection assertions**
  - This should be a mandatory RTL coding style rule!

---

I'm Still In Love With My X – Stuart Sutherland, Sutherland HDL, Inc.
Questions and Answers

Optimist
The glass is half-full

Pessimist
The glass is half-empty

Engineer
The container is twice the size that it needs to be... and I won't be able to sleep tonight until I have redesigned it!

I'm Still In Love With My X – Stuart Sutherland, Sutherland HDL, Inc.