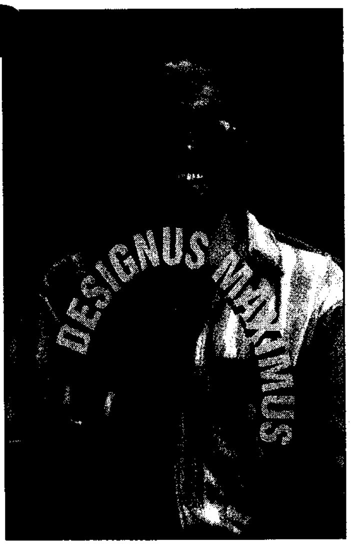
"Xs" IN DIGITAL SIMULATION: BEWARE, HERE BE DRAGONS!



CLIVE "MAX" MAXFIELD, INTERGRAPH ELECTRONICS

Anyone using a digital simulator can expect to encounter the dreaded "X" value, but there are few who truly understand its precocious sense of fun. This article tracks the nefarious "X" to its lair and reveals why users should be warned: Beware, here be dragons!

Centuries ago, Medieval cartographers penned the phrase *Beware, here be dragons* on maps to indicate uncharted territory. Today, the same words of caution should be applied to the "X" value in digital simulation. This value, which is used to indicate an unknown state, directly affects anyone using a digital simulator; yet, few users fully appreciate all of its ramifications. This lack of understanding can be extremely unfortunate, because the quality of simulation results can be drastically compromised by the underlying (and often incompatible) assumptions made by the people who write simulation models and the poor souls who end up having to use them.

Each digital simulator can represent a specific number of logic values. For example, certain specialized simulators only consider signals as carrying logic 0 and logic 1 values, but others might consider 128 different values or more. Additionally, the way in which these values are represented and evaluated varies among simulators. Some simulators employ cross-product techniques; others prefer to use an interval-value approach.

Unfortunately, a detailed comparison of cross-product versus interval-value logic sets is beyond the scope of this article. For the purpose of this discussion, I will concentrate on the (almost) universally available logic value subset comprising 0, 1, "X," and "Z," where "X" and "Z" represent "unknown" and "high impedance," respectively. However, if you should happen to be overcome with a raging curiosity about the minutiae of logic value sets, feel free to email me to request this topic for a future article.

Don't know, don't care

In addition to 0, 1, "X," and "Z," most hardware-description languages (HDLs) also support a "?" value, meaning "don't care." In fact, "?s" aren't true values in the sense that they can't actually be assigned to outputs as driven states. Instead, "?s" are predominantly used to describe the way in which a model's inputs should respond to any signals that

VHDL don't care → '-'

are presented to them. For example, consider the model for a simple 2:1 multiplexer (**Fig 1**).

Although it's conceivable to enumerate all of the possible combinations of logic values that could be presented to the inputs, the use of "?" values to indicate "don't care" conditions both eases the task of creating the model and more accurately reflects the model's intent.

For the sake of completeness (and to avoid receiving a mailbag of irate letters), I should also note that some HDLs do permit "don't care" values to be assigned to outputs. However, in this case, the "don't cares" are intended for use by a logic-synthesis utility and not by the digital simulator itself. (The simulator automatically converts these assignments into unknown "X" values at runtime.)

One of the most common traps novice simulation modelers fall into involves the data-book convention of using "X" characters to represent "don't care" conditions. If the model writer neglects to translate these into simulator "don't cares" ("?s"), then, somewhere down the line, someone is going to spend one heck of a long time trying to figure out what's happening—and I'm tired of that someone being me.

What does "unknown" actually mean?

A significant problem with today's digital simulators is that they tend to use "Xs" to represent a variety of conditions. For example, consider the circuit in **Fig 2**, which, among other things, contains a D-type flip-flop **(a)**, a loop

THE "XS" OF THE FUTURE

The "X" value that we know and love today may not necessarily be the same "X" we'll use in the future. There are a number of possibilities for the evolution of "Xs" that developers of digital simulators might consider if designers feel they need such capabilities.

Static vs dynamic "Xs"

One of the problems with current digital-simulation technology is that "Xs" represent multiple conditions, from a steady-state, well-behaved unknown (which we might call a "static X"), to an uncontrolled oscillation of unknown frequency and unknown amplitude (which we might call a "dynamic X"). One solution is for both the designer and the simulator to be able to distinguish between these extremes. For example, we might use two symbols, "X" and "#X," to represent static and dynamic unknowns, respectively.

In this case, an uninitialized register element (**Fig 2a**) could generate well-behaved "X" values, but two gates driving incompatible values onto a common signal (**Fig 2c**) could result in a more pessimistic "#X" value. To illustrate a possible application of these two values, consider a modified, pseudo-HDL representation of a 2:1 multiplexer (**Fig A**).

Note that, unlike today's digital simulators, in which the model writer and the model user have to agree whether an unknown applied to the select input would cause an optimistic or pessimistic response at the output, the ability to differentiate between "X" and "#X" would allow the model to respond appropriately in both cases.

Inverse "NOT Xs"

Today's simulators do not consider the effect of inverting "Xs." For example, if an "X" is presented to the input of a simple inverting logic function, the resulting output from the function is also "X." Thus, another possibility for simulator developers is to introduce the concept of "NOT X," which we might represent as "~X." The concept of "~Xs" could greatly reduce pessimism and aid the simulator in clearing out uninitialized unknown values (**Fig B**).

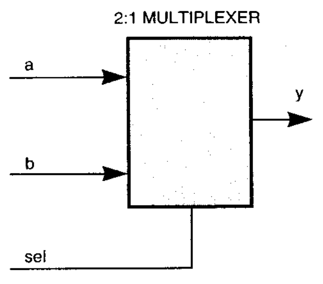
Obviously this is a contrived example, but it illustrates a point. Assume that the circuit has recently powered up and the registers contain unknown states. With today's simulators, you could apply clock pulses to clk1 infinitely without any useful effect. However, if the simulator inherently understood the concept of "~X" (as shown in **Fig B**), then the first positive edge on clk1 would cause the D2.q output to toggle from its original "X" to an "~X" value. The simulator could be made to recognize that one of these two values has to represent a lo 0, which would cause the D2 flip-flop to be placed in its s state.

"Xs" with ID numbers

As one final suggestion, consider giving each "X" a unique ID number. Every "X" in existing simulators is indistinguishable from its counterparts, which can result in undue pessimism (**Fig C**).

Note that, even in the relatively simple case of an XOR gate, the gate cannot recognize the fact that the "X" applied to both of its inputs comes from a common source. If the simulator could tell that both of these "Xs" were, in fact, the same, then the output from the gate could be assigned the less-pessimistic value of logic 0, which could aid in initializing other down-

FIGURE A



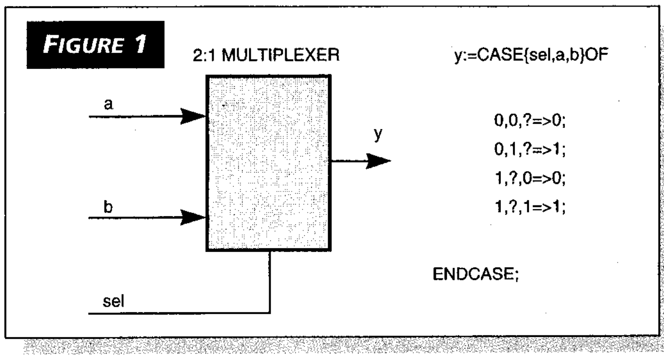
```
y:=CASE(sel)OF
    0=>a;
    1=>b;
    X=>a EQU b;
    #X=>#X;
ENDCASE;
```

This pseudocode HDL example for a 2:1 multiplexer uses both "X" and "#X" values.

formed from three inverters (b), and two tristate buffers driving the same node (c). In the case of (c), assume that both of the tristate buffers are enabled, and that one is attempting to drive a logic 0 value while the other is attempting to drive a logic 1.

In the case of (a), assume that power has recently been applied to the system and that the register was powered up with its "clear" input in the inactive state. Thus, the unknown "X" value on the register's output represents an uninitialized state. Additionally, if we assume that sufficient time has elapsed for the register to stabilize internally, then it's possible to say that this "X" represents a good, stable logic 0 or logic 1—even though we don't know which particular logic value it is.

FIGURE 1



```
y:=CASE{sel,a,b}OF

    0,0,?=>0;
    0,1,?=>1;
    1,?,0=>0;
    1,?,1=>1;

ENDCASE;
```

This pseudocode HDL for a 2:1 multiplexer uses "?" ("don't care") values as input assignments.

stream logic. The potential benefits increase dramatically if we consider a signal that diverges into multiple paths, where each path passes through several levels of logic—and then two or more of the paths reconverge at some point downstream. In fact, tracking "Xs" in this way would be similar in principle to the way in which today's dynamic timing analyzers resolve timing pessimism in circuits exhibiting reconvergent fanout, which [illegible] also known as common-mode ambiguity.

Another, but perhaps less obvious, application of "Xs" with IDs could be in pinpointing the origin of an "X." Assume that you are running your first-pass simulation, you are monitoring only the primary outputs from the circuit, and, at some stage during the simulation, you see some "Xs" in the output waveform display. The problem is that these "Xs" could have origi-

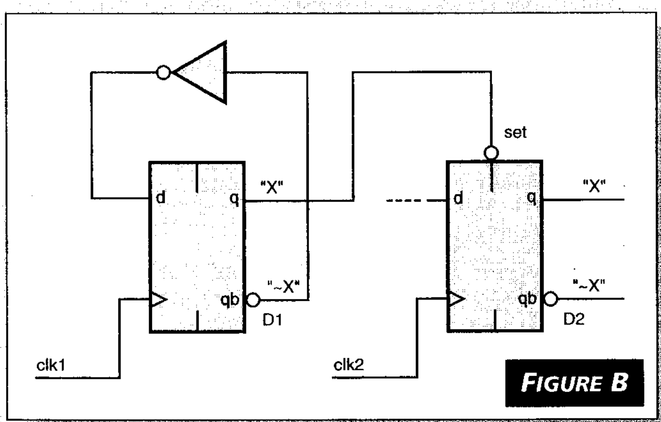
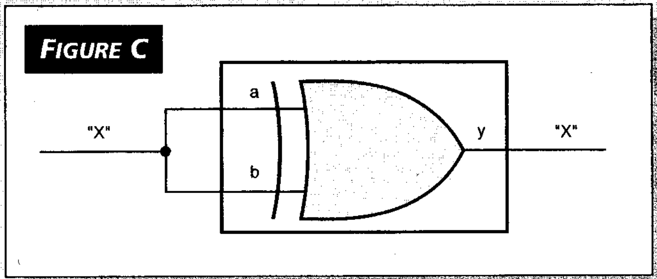


FIGURE B

If simulators could support "~Xs," their results would be less pessimistic.

nated deep in the bowels of the circuit, thousands of time-steps in the past. With today's simulators, your only recourse is to rerun the simulation and work your way back from the suspect [illegible]imary output, a process that resembles a salmon's struggle [illegible] swim upstream. However, if the "Xs" had unique IDs, it would be feasible to click your mouse on an "X" in the waveform display, and for the simulator to inform you that *this "X" originated at time 6854 nsec at gate G4569*. Thus, to pinpoint and isolate the problem quickly, you could immediately target the offending gate and monitor all of the signals in its vicinity.

FIGURE C



"Xs" are unduly pessimistic, because, in this case, the outputs should ideally be a 0.

Wouldn't it be nice if...

Each of the above suggestions ("X," "#X," "~X," and "Xs with IDs") could be useful in isolation, but a combination could dramatically improve the quality of digital simulations. One downside to all of this theory is that even today's simple "Xs" tend to impact simulation speed negatively, because they can procreate and propagate throughout a circuit at a frightening rate. The more sophisticated "Xs" discussed here, especially those with individual IDs, would certainly slow the simulator even further.

However, there are a number of ways to mitigate these detrimental effects. For example, one suggestion is for the simulator to only differentiate between "X," "~X," and "Xs with IDs" during the time when the circuit is undergoing its initialization sequence. Once the circuit has been initialized (at a user-specified time), the simulator could then revert to considering only "X" and "#X" values.

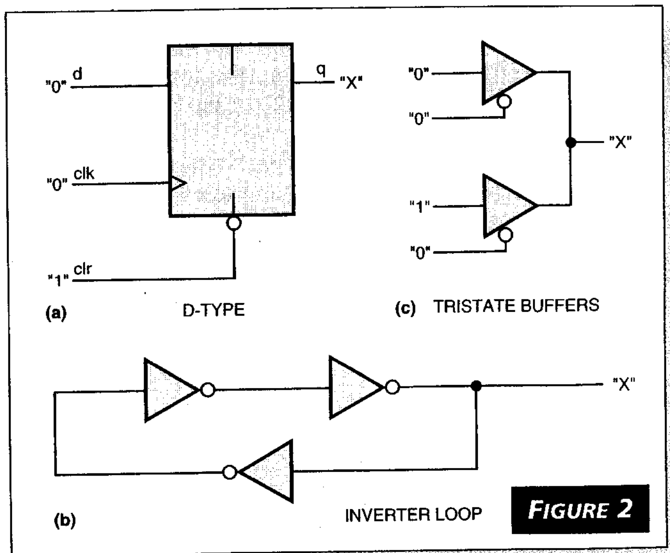
And, finally, I should point out that this excursion into the world of "Wouldn't it be nice if..." has only scratched the surface of what's possible. Ultimately, it's you, the designers in the trenches, who will determine how simulation tools evolve in the future.

DIGITAL SIMULATION

In the case of (b), the "X" generated by the inverter loop actually represents a controlled oscillation between good logic 0 and logic 1 values. Note that, in this context, "good" refers to the fact that the output isn't stuck at an intermediate voltage level, but that it is achieving real logic 0 and logic 1 thresholds.

Finally, in the case of (c), the output depends on the drive strengths of the two tristate buffers. If the buffers are of relatively equal strength, then the worst-case scenario is that "X" potentially represents an uncontrolled oscillation of unknown frequency and unknown amplitude.

In fact, some HDLs also support the concept of an uninitialized "U" value; for example, the VHDL "Standard Logic Value" set as defined in the IEEE 1164 standard (this nine-value set is sometimes unofficially referred to as MVL-9). Back in **Fig 2**, the uninitialized "U" value can be used to differentiate case **(a)** from cases **(b)** and **(c)**. However, today's digital simulators don't offer any way to differentiate between cases **(b)** and **(c)**.



Digital "X" values can represent a variety of different cases: for example, an uninitialized D-type register element (a), a loop constructed using inverters (b), or two tristate buffers attempting to drive opposing values (c).

Who makes the decisions around here?

If you're using digital simulation to verify a design, and if your company creates its own simulation models, then it's imperative for project leaders to decide and document what they want "X" values to represent—or, more precisely, how they want simulation models to deal with "Xs." Consider the case of a 2:1 multiplexer, which has logic 1 values presented to both of its data inputs and an "X" presented to its "select" input (**Fig 3**).

The decision to be made is what you want this model to generate as its output. Remember that the simulator (and, thus, the person writing the model) cannot differentiate between "Xs," and, therefore, that it's not possible to determine what any particular "X" represents. One option is to assume that "Xs" are generally well-behaved; that is, an "X" represents a stable logic 0 or logic 1 (as in **Fig 2a**) or well-defined oscillation (**Fig 2b**). If we assume that "Xs" are well-behaved, then we may take an optimistic attitude: The output from the multiplexer should be a logic 1 because, if both of the inputs are logic 1, it doesn't really matter which one is being selected. On the other hand, if we acknowledge that "Xs" may actually represent an uncontrolled oscillation of unknown frequency and amplitude, then we should really take the pessimistic approach and cause the output to drive an "X."

Whiners or thrill-seekers?

We can use the final decision to judge the project leaders' cowardliness or aggressiveness. Is this group a bunch of cringing whiners or reckless, irresponsible thrill-seekers looking for an adrenaline-rush on the company's budget? Actually, I'm not advocating any particular view of "Xs" but am reinforcing that someone, somewhere, has to make an informed decision. The absolute, worse-case scenario is for multiple modelers to create different models without any documented standard on how their models should view and handle "Xs." (If this were the case, then Murphy's Law dictates that different modelers are guaranteed to use different approaches.)

Ultimately, without a company standard, the end user doesn't know what an individual model may do. If a design-er uses two models with identical functions written by different modelers, the models could respond differently to the same stimulus, which is generally not considered an optimal situation. Even worse, consider the case where one model is a high-power drive equivalent of the other: By exchanging models to test the effects of using a different drive capability, the designer may completely change his simulation results, which can potentially consume endless man hours before it's tracked down to the simulation models.

Similarly, in addition to internally developed models, it's also necessary to define exactly what you expect when you're acquiring models from outside sources—especially if these models originate from multiple sources.

"Xs" and initialization

"X" values can perform a number of different (and, as we've seen, often incompatible) roles in digital simulation, but one very common role is that of indicating uninitialized elements. In this case, some users take the view that every memory element should power up containing "Xs," and if you can't clear them out as part of your initialization sequence, then shame on you.

But, in the real world, there's no such thing as an "X," and designers are endlessly inventive in making use of the fact that, in certain cases, it doesn't matter whether a particular element contains a logic 0 or a logic 1. For example, consider a number of D-type registers configured as a divide-by counter. In some cases, the designer simply may not care how the individual elements of the counter initialize. Although a purist would recommend using registers with clear inputs, the overhead of tracking an additional clear signal around the circuit may be unacceptable to the designer.

Similarly, in the case of RAM, generally one would expect and require them to power up with "Xs" in the simulation to indicate that they contain random logic 0 and logic 1 values. However, if the output from the RAM feeds into some other logic, such as a state machine that has already been initialized, then the "Xs" from the RAM may escape into this downstream logic and poison it.

Unfortunately, there is no all-embracing answer to satisfy every situation. Almost every digital simulator allows you to force a value onto a selected signal. For example, in the case of the inverter loop in **Fig 2b**, it's possible to force one of the signals forming the loop to a logic 0, hold that value for a sufficient amount of time for its effect to propagate around the loop, and then remove the forced value to leave the loop acting as an oscillator. However, be parsimonious in using this technique, because you've just introduced something into your simulation that does not reflect the circuit's real-world behavior. (It's not uncommon for problems to arise at a later date when the circuit is revised.) In particular, document exactly what you've done and ensure that you alert other team members whenever you use this methodology.

Another common technique is to use an internal simulator function to randomly coerce uninitialized "X" values into logic 0s and logic 1s. This strategy can be very useful, but, if the simulator supports it, you should restrain yourself to targeting only specific trouble spots. Also, at a minimum, ensure that you repeat the simulation with a variety of different, random seed values.

Perhaps the best advice to keep in mind is that hardware initialization techniques should be employed wherever possible, and that simulator tricks should be used sparingly—and with caution. It is not unheard of for a design to function in the simulation domain only to fail on the test bench because the designer used the simulator to force conditions that simply could not occur in the physical world.

"Xs" in mixed-signal environments

Last, but certainly not least, take particular care of "Xs" in mixed-signal designs, in which a digital simulator is interfaced to, and simulating concurrently with, an analog simulator. Obviously, "X" values are absolutely meaningless in the analog domain, so they have to be coerced into voltage levels that correspond to logic 0s and logic 1s as they are passed from the digital simulator to its analog counterpart.

The problem is that the analog portion of the circuit can act like an "X filter." For example, consider a digital portion of a circuit containing uninitialized "Xs" driving into an analog portion, which, in turn, feeds back into the digital portion. The "Xs" are removed as the signals move across the digital-to-analog boundary, but there's no way to restore them at the analog-to-digital interface. Thus, the downstream digital portion of the circuit "sees" an optimistic view of the world, which may lead portions of the circuit to appear to be initialized when, in fact, they are not.

Some systems only offer the option to coerce the "X" values into logic 0s (or logic 1s) as they are handed over to the analog simulator. In general, avoid this technique at all costs. Wherever possible, employ the technique of coercing the "Xs" to random logic 0s and logic 1s. Additionally, some systems combine an initial random assignment with the fact that every subsequent "X" on that signal will be alternately coerced to the opposite logic value of the one used previously. In all of these cases, plan on performing a number of simulations using a variety of random seed values.

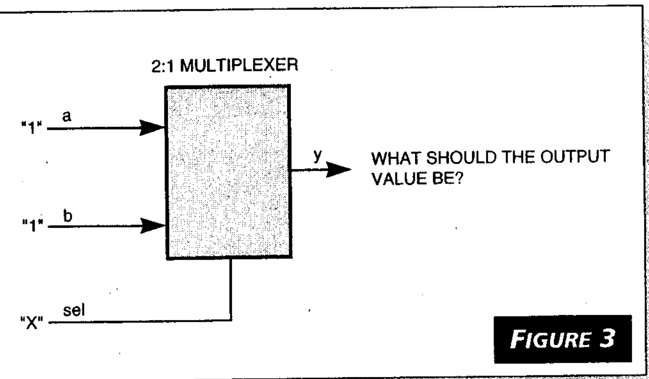


FIGURE 3

The way in which "Xs" are treated depends on the individual model writer; for example, consider the case of an "X" value applied to the select input of a 2:1 multiplexer.

The moral of the story

Treat "X" values with both respect (they can be extrem useful) and caution (they can potentially be the source diverse and subtle problems). It is not enough to say that "Xs" simply mean unknown, because we need to define what we mean by "unknown." Similarly, it's essential that everyone involved in creating and using a particular set of simulation models is in complete agreement as to how they expect the models to behave, both in the way the models generate "Xs" and the way in which they respond to them. EDN

Author's biography

Clive "Max" Maxfield is a member of the technical staff (MTS) at Intergraph Electronics (Huntsville, AL), where he specifies electronic-design-automation (EDA) products (phone (800) 837-4237). You can reach him by email at crmaxfie@ingr.com. Max is also the author of Bebop to the Boolean Boogie (An Unconventional Guide to Electronics). *Phone HighText Publications Inc (Solana Beach, CA) at (800) 247-6553.*