Delay-Safe False Paths

Engineers enter false paths on a design for a variety of reasons. A path may be specified false because it is between two asynchronous clock domains, or because the path originates from a static register whose value never changes, or because the path is a timing don't care. In this paper we focus on paths that are specified false because the engineer believes the path is impossible to sensitize. Engineers specify such paths as false because they believe the combination of logic values required to sensitize the path is not possible based on the functionality of their design.

Static sensitization is an approach to false-path verification that establishes if the required combination of values is possible to sensitize a path. If the combination is impossible a false-path definition is considered good. If it is possible then the false-path definition is considered bad. However, it is important to consider not just the functionality of the chip but circuit delay when determining if a path is sensitizable.

Dynamic sensitization refers to the fact that while in any given clock cycle, the static state of a design may be such that a path cannot be sensitized, the dynamic nature of the design (once timing delays are accounted for) may temporarily allow the design to enter the state necessary to sensitize a path. Dynamic sensitization allows glitches to propagate through paths that are statically false. It is important that false-path definitions established using static sensitization are safe under dynamic sensitization, i.e. that static sensitization does not incorrectly mask real timing problems by underestimating the true delay on a design.

Static and Dynamic Sensitization are Unsuitable for False Path Verification and Generation

Consider the circuit in Figure 1. Assume that net delays are zero and gate delays are 1ns. Assume the following two false-path definitions are specified:

```
(1) set_false_path -from A -through D -to Z
(2) set_false_path -from B -through D -to Z
```

If these false paths are verified considering just static sensitization then both false-path definitions will be considered good. The propagation condition for the first false-path definition is `(B & !C & E)`. If `B` is high then `E` must be low, so `(B & !C & E)` is always false. So, based on static sensitization, false-path definition `(1)` is good. Similarly, the propagation condition for false-path definition `(2)` is `(A & !C & E)`. If `A` is high then `E` must be low, so `(A & !C & E)` is always false. So, based on static sensitization false-path definition `(2)` is good. Once both false-path definitions are applied the critical delay on the design is 2ns, because both 3ns paths have been classified as false paths.

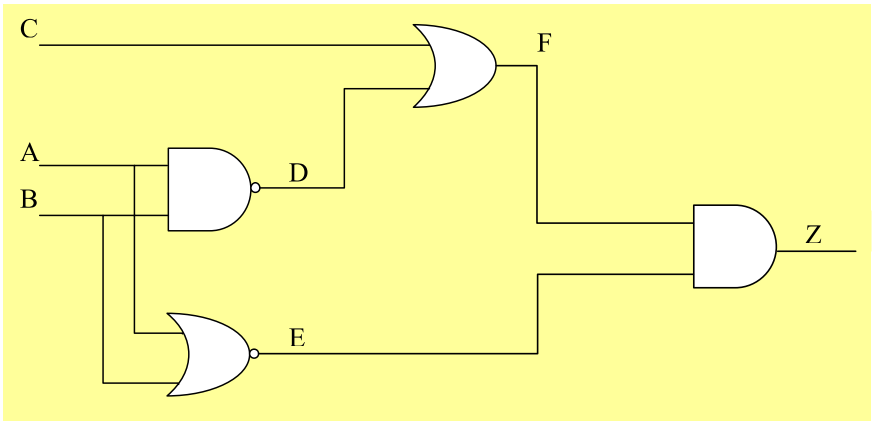

Figure 1: Example design.

Now, consider the dynamic nature of the design. Assume all inputs (`A`, `B`, `C`) transition from high to low at 0ns as shown in Figure 2. Nets `D` and `E` will both go high at 1ns. `F` will go high at 2ns and `Z` will go high at 3ns. So, the real critical delay to `Z` is 3 ns – not 2 ns and false-path definitions `(1)` and `(2)` are actually incorrect – they both apply to dynamically sensitizable paths that are, in fact, the critical paths to `Z`. Applying these false-path definitions is a recipe for disaster because it could easily result in timing failure.

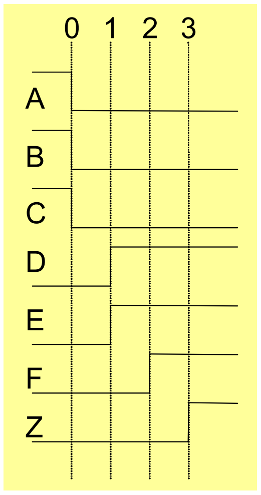

Figure 2: Stimulus applied to example design.

So, clearly static sensitization cannot be used to verify if a false-path definition is good. It can easily say a false-path definition is good when actually the path is both sensitizable and is the critical path to a timing endpoint. Dynamic sensitization appears to overcome this issue because it takes circuit delay into account. The problem with dynamic sensitization, however, is that the results it reports change with circuit delay. For example, assume that the nor-gate in Figure 1 has a delay of 3ns and all the other gates have a delay of 1ns. With this delay assignment, dynamic sensitization would consider false paths `(1)` and `(2)` correct. Consider you are performing dynamic sensitization using worst-case maximum delay values. Based on a 3ns delay, dynamic sensitization would conclude that paths `(1)` and `(2)` are false, but if when you manufacture the chip, the nor-gate sometimes has a delay of less than 3ns (1ns, for example) then the paths are not false and applying a false-path definition that has been generated or verified using dynamic sensitization will result in silicon failure.

The delay on a design undergoes significant change as it is taken through the implementation flow. A false-path verification approach based on dynamic sensitization could easily say that a false-path is good after logic synthesis, bad after placement, good again after routing and finally bad after considering SI effects on delay. It is impossible to implement a chip if you have to keep adding and deleting false-path definitions as implementation progresses. Ideally, you need a stable constraint file at the start of chip-implementation whose contents have been verified good and whose contents stay good as chip-implementation progresses. So, what is required is a false-path verification approach that is able to detect those false-path definitions that are good and impervious to changing circuit delay.

Theory

A *controlling side-input* is an input to a gate that needs to have a specific value for the transition on another input to the gate to propagate to the output. To propagate a transition through a gate it is only necessary for a controlling side-input to have the required value at the time the transition is propagating through the gate. So, in the example in Figure 1, for the `A->D` transition at time `0`, `B` needs to be high at time `0`, for the `D->F` transition at time `1`, `C` needs to be low at time `1`, and for the `F->Z` transition at time `2`, `E` needs to be high at time `2`. It is not necessary for `B` and `E` to be high and for `C` to be low for the entire duration of the clock cycle as assumed by static sensitization.

Until the controlling side-inputs reach their final stable value there may be a transitory period when the required combination of controlling side-input values is achievable to propagate a path. Once the controlling side-inputs reach their final stable value it is impossible to propagate a transition along a path that is statically false. The key point here is that for a static false path to be dynamically sensitizable under some input stimulus there **must** be a controlling side-input to a gate that transitions at the same time or after the time at which the input to the gate along the path transitions. This is the "*requirement for the dynamic sensitization of static false paths*". For example, the path `A->D->F->Z` in Figure 1 that is statically false is dynamically sensitizable for the stimulus shown in Figure 2 only because the controlling side-input `B` transitions at 0ns (at the same time at which the input `A` transitions). If, for the stimulus shown in Figure 2, `B` had transitioned before `A` then the path `A->D->F->Z` would not be dynamically sensitizable (`B` with a value of `0` would block the `A->D` transition).

Delay-Safe False Path Verification and Generation from FishTail

The false-path sensitization approach taken by FishTail's exception generation and verification products (Focus and Confirm) ensures that only false-path definitions that are **safe under dynamic sensitization** are both generated and verified to be good. We will refer to the false paths that Focus generates or that Confirm verifies as correct as delay-safe false paths. We are able to **guarantee** that even under dynamic sensitization delay-safe false paths will not mask real timing problems on a chip and that **delay-safe false paths will never cause the delay to a timing endpoint to be underestimated**. Bottom line - delay-safe false paths are impervious to circuit delay and are always safe to apply.

Delay-safe false paths result from control flow on a design. For each gate in a design we partition the inputs to the gate into data and control. A couple of examples are shown in Figure 3. Control inputs to a gate (`sel` in Figure 3a) impact the flow of information through the data inputs of the gate (a and b in Figure 3a). So, for example, the transition from a->y in Figure 3a requires sel to be high and the transition from b->y requires sel to be low. Control and data inputs adhere to the following rules:

1) Only control inputs to a gate are controlling side-inputs – so only control inputs impact the propagation of transitions through a gate.

2) Data inputs are never controlling side-inputs. So, data inputs have no impact on the propagation of transitions through a gate.

3) Transitions on control inputs propagate unconditionally to the output of a gate. So, nothing blocks the transition on a control input.

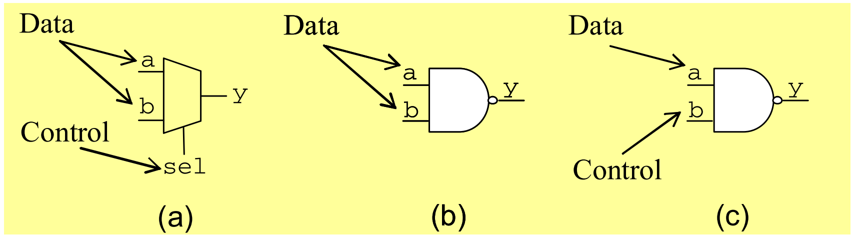

Figure 3: Example control and data markings.

Figure 3b shows one possible control and data marking for an and gate. Both inputs to the and gate are of type data and because there is no control identified, the transition from the inputs of the and gate to the output is always possible and unconditional. Figure 3c shows another possible control and data marking for an and gate where the a input is data and the b input is control. With this control marking the b input unconditionally transitions to y, while the a input only transitions to y when b is high.

For the design in Figure 1, under no control/data marking that conforms to rules (1) through (3) above, are the false-path definitions (1) and (2) both considered good. For example, if none of the inputs to any of the gates is marked as control, then there are no controlling side-inputs and all paths on the design are true. So, Focus would not generate any false paths on the design and both false-path definitions (1) and (2) would be flagged as incorrect by Confirm.

Now, consider the design in Figure 4 to see why delay-safe false paths are impervious to changes in circuit delay. Assume all net delays are zero and gate delays are 1ns. Focus would generate and Confirm would verify the following false-path definition as good:

```
set_false_path -from B -to Z
```

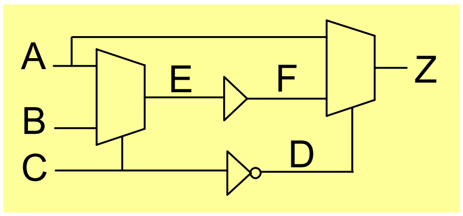

Figure 4: Example design.

The `B->E` transition in Figure 4 requires `C` to be low and the `F->Z` transition requires `D` to be low. So the propagation condition for the false path is `(!C & !D)`. If `C` is low then `D` must be high and so the condition `(!C & !D)` is always false. The path is statically false. Without the false-path definition the critical delay to `Z` is 3ns (`B->E->F->Z` and `C->E->F->Z`). Even with the false-path definition applied the critical delay to `Z` remains 3ns (`C->E->F->Z`).

Why is the false-path definition safe under dynamic sensitization? Consider the stimulus shown in Figure 5 where all inputs (`A`, `B`, `C`) transition from low to high at 0ns. `D` goes low at 1ns, `E` goes high at 1 ns, `F` goes high at 2 ns and `Z` goes high at 3ns. The path `B->E->F->Z` is dynamically sensitizable with a delay of 3ns, but because there is no false-path definition from `C` to `Z` the delay to `Z` is not under-estimated (`C->E->F->Z` is a true path). So the false-path definition does not mask a real timing problem on the design and is safe under dynamic sensitization.

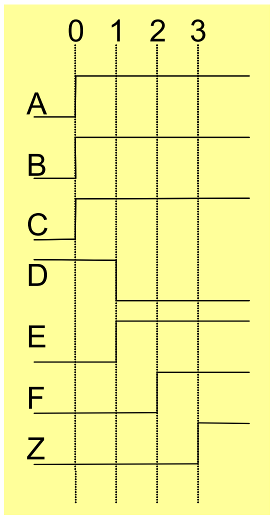

Figure 5: Stimulus applied to example design.

The reason delay-safe false paths are impervious to circuit delay is the following. According to the "requirement for the dynamic sensitization of a static false path", there must be a controlling side-input at a gate that transitions at the same time or later than the transition on the input to the gate along the path. With our control-based approach to false-path generation and verification, the controlling side-input must be a control input (all controlling side-inputs are control inputs). Control inputs always unconditionally transition to the output (nothing blocks the transition from a control input to the output of a gate). So, when a delay-safe false path is dynamically sensitizable, there will always exist a path through a control input that is a true path with delay greater than or equal to the dynamically sensitizable false path. For this reason delay-safe false paths never mask a real timing problem on the chip and are always safe to apply in the implementation flow.

The false paths generated by Focus and those verified as good by Confirm are delay-safe false paths that are legitimate to apply throughout the implementation flow regardless of circuit delays.