

Latch & Flip-Flop Modeling in AccuCell and AccuCore

Introduction

This application note explains AccuCell and AccuCore's automatic latch and flip-flop function identification process.

Overview

AccuCell and AccuCore automatically recognize the latch and flip-flop structures shown in Figure 1 through a strength-driven, state-based, Ordered Binary Decision Diagram (OBDD) algorithmic method, rather than simple pattern matching against internal templates with pre-defined functional descriptions. The algorithmic method is faster than matching-only methods. It also enables the extraction of the actual function present, across a wide variety of logic design styles and in the presence of RC parasitics.

Flip-flops are always initially broken down into latches unless part of a defined pattern file. However, by default, the tool will then attempt to form and compress master-slave latch pairs back into a flipflop during the classification phase. Structures classified as "flip-flops" will be characterized and modeled as edge-triggered devices.

Other variations of these latch and flip-flop structures may also be recognized, but such recognition is NOT guaranteed. Latches and/or flip-flops employing "pulsed" or "glitch" styles of clocking will NOT be automatically recognized. Edge-triggered flip-flops, not so classified, will require the use of either subcircuit preservation via the `KEEPSUBCKT` or `KEEPINST` commands, or use of topological pattern matching via the `FINDSUBCKT` command. Once pattern matched, an attempt will be made at classification.

Structures that fail automatic function extraction and classification will require either equation (.eqn) or vector table file (.tbl) driven information to be provided manually or from a library of defined solutions. For more information about this topic, see Simucad Application Note *Manual Latch and Flip-Flop Recognition in AccuCell and AccuCore*, the AccuCell and AccuCore documentation, or consult a Simucad AE.

Automatic Function Extraction

Automatic function extraction begins by identifying weak and strong static drivers through a fast duality check between pull-up and pull-down paths for each node. This ensures that the direction of the signal is correctly identified. A set of four subfunctions, represented by Ordered Binary Decision Diagrams (OBDDs), are extracted for each output of the partition/cell. The functions are expressed in terms of the primary inputs and State-Points identified by means of a netlist traversal.

`DEBUGBDDS` - 0 - Controls the printing of exhaustive information of output equations.

`PRINTEQNS` - 0 - Controls the printing of basic output equations.

State-Point Identification

Sequential-type logic circuits, latches, flip-flops, registers, and memory cells are examples of circuits with "state holding". f_0 (a discharge path "0"), f_1 (a charging path "1"), f_0' (no discharging path "NOT 0"), and f_1' (no charging path "NOT 1") fully describe the overall function.

"State holding" exists if (a) $(f_0 \neq 0 \text{ OR } f_1 \neq 0)$ AND $(f_0' \text{ AND } f_1')$ (i.e. floating); and (b) the holding period is much larger than the clock frequency (i.e. the effective node capacitance is greater than a user specified threshold). At this point, four final equations are created for each output and internal state, where the output is the name of the output pin. These are as follows:

`output.0` when output drives zero. (.0 in `PRINTEQNS` output)

`output.1` when output drives one. (.1 in `PRINTEQNS` output)

`output.z` when output is undriven. (.2 in `DEBUGBDDS` output)

`output.x` when output is unknown. (.3 in `DEBUGBDDS` output)

In the last step of the process, these four equations are analyzed to determine the classification.

Equations (.4 and .5 in `DEBUGBDDS` output) may be generated dealing with strong node identification as part of the directionality (and partitioning) process and final function determination under certain conditions.

Conclusion

Since an algorithmic approach, rather than user-defined patterns is used, functional extraction of a wide variety of cell types including complex flip-flops, latches and large footless and footed dominos is possible. As a result partitioning (AccuCore only) function extraction and vector generation are all interrelated in an iterative process controlled by a shared set of commands. Commands which appear in one flow section may have an additional role in other steps and thus may appear “out of sequence” in the log file.

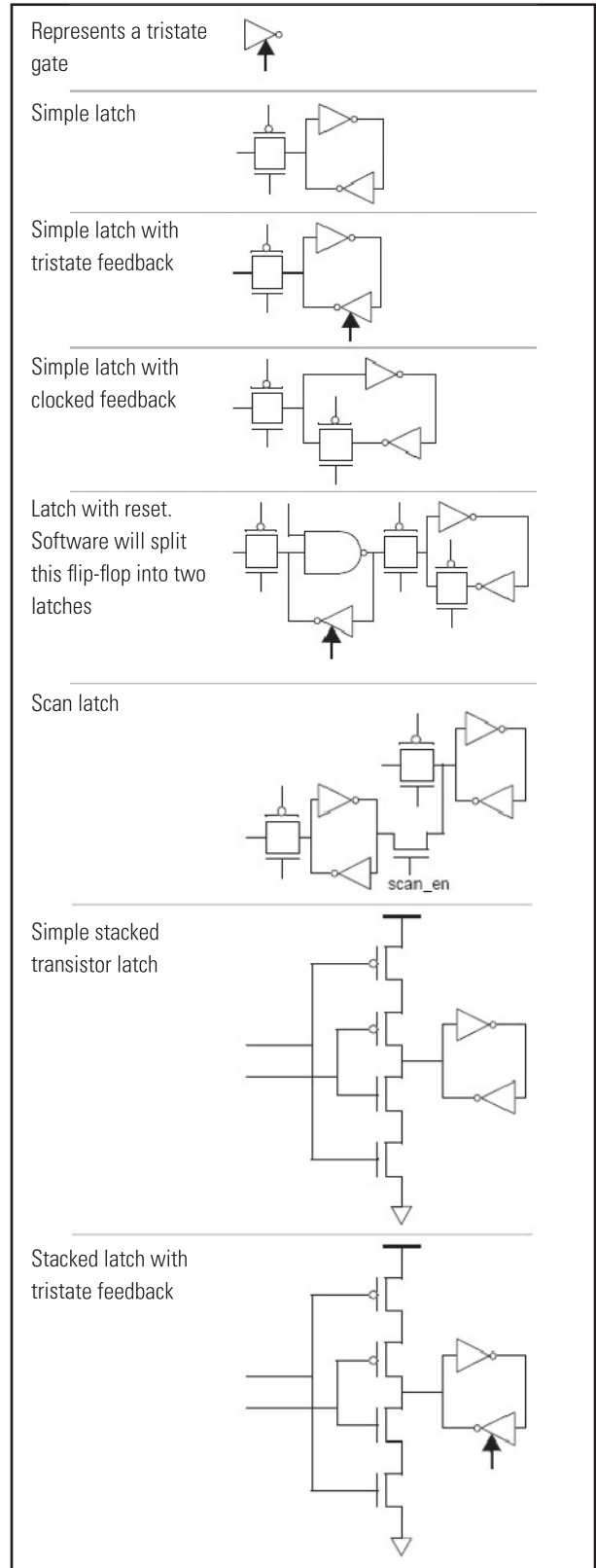


Figure 1. Supported structures