NOT JUST FOR HARDWARE DEBUG: PROTOTYPE DEBUGGERS FOR SYSTEM VALIDATION AND OPTIMIZATION

Introduction

Simulation and emulation are great for finding and fixing hardware bugs. Software debuggers are great for finding and fixing general purpose software bugs. But traditional debugging tools often fall short in finding and fixing system-level bugs in complex SOC designs. In a series of short case studies, this paper explores how the use of embedded debug instrumentation in modern FPGA prototype debuggers can meet the emerging need for system-level debug.

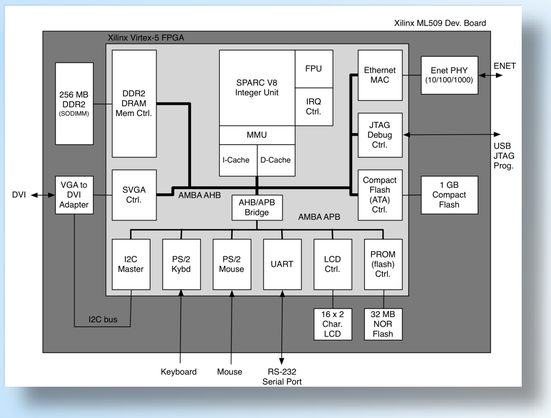
The key challenges in system-level debug and debug productivity fall broadly into three categories:

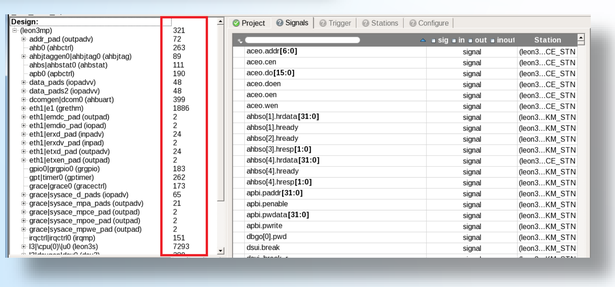
- Trace breadth
- Trace depth
- Turn-around time to debug the appropriate set of signals relevant to the problem

Trace Breadth

Trace breadth means being able to trace all the signals relevant to debugging a particular issue.

To debug system-level interactions, the need to trace hundreds or thousands of signals and registers across multiple clock domains is typical, especially during the initial phases of debug identification and isolation. The introduction of multiple clock domains derives a further requirement that the trace data must be correlated in time in order to comprehend true cause-and-effect relationships.

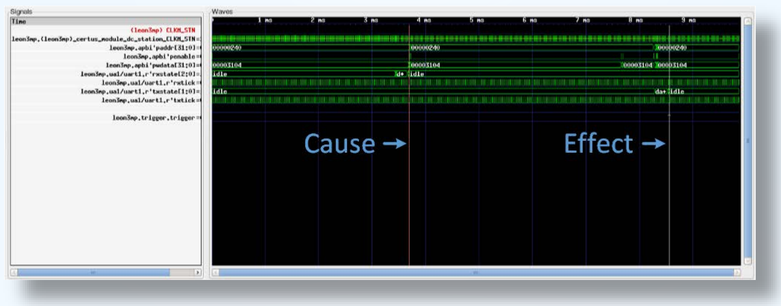




To avoid frequent re-compiles, instrumenting thousands of signals are required to debug even a simple SOC.

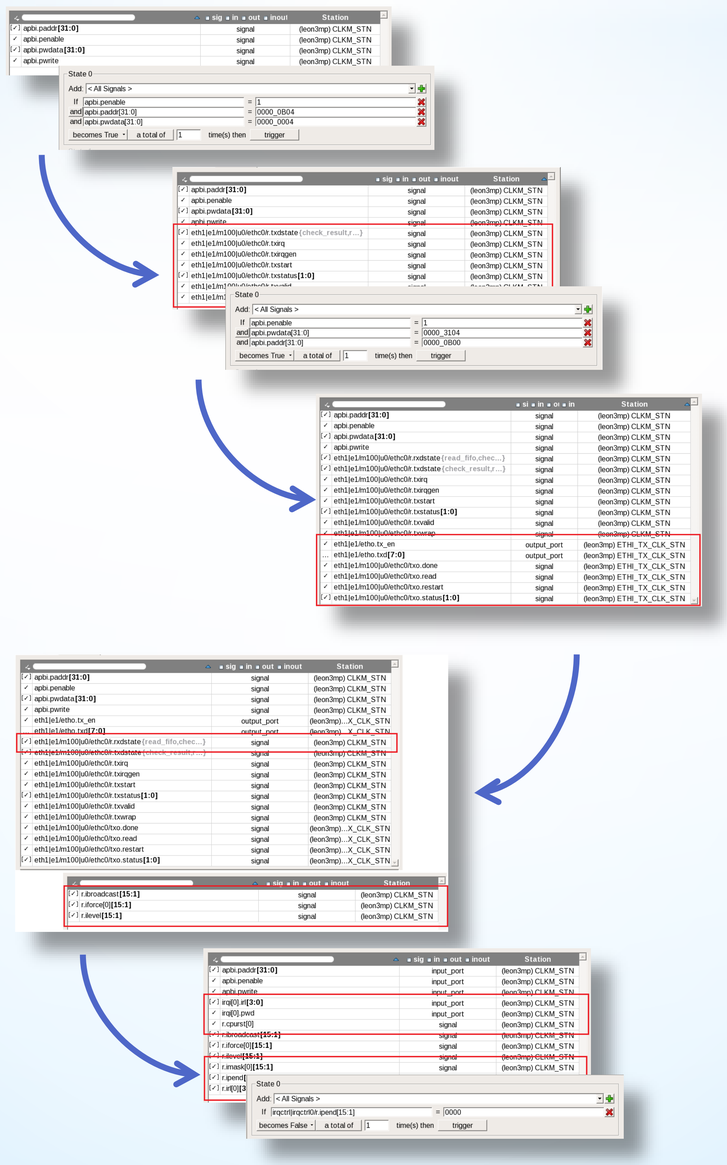
Trace Depth

Trace depth determines the ability to capture sufficient activity to recognize system-level cause-and-effect relationships. Since the effect of a bug may surface many transactions later from the cause, the ability to trace seconds of trace data can determine whether the validation engineer can debug an issue from one trace set or if multiple runs are required to capture multiple snapshots of trace data which the engineer must cobble together manually to root cause the issue.



Turn-Around Time

Turn-around time when debugging. Debugging is an iterative process of tracing back from an observable effect back to the root cause. Since the root cause is not yet known, all the signals relevant to debug are not known upfront. The signals the user wants to initially observe will rarely be sufficient to find the root cause of the bug. Yet, re-instrumenting can take hours, often over-night, if re-synthesis and place & route are required. Therefore the turn-around time from capturing one set of signals to a different set of signals if often the largest impact on time to root cause a bug.



It can take many iterations of signal selections and trigger settings to find the root cause of a bug.

Case Study 1 - Root Causing System-Level Bugs that Escape Simulation Detection

The first case study involves a team doing hardware/software co-design on a full system prototype. Software testing exposed two features behaving incorrectly. Both hardware and software designers searched for the cause of the issues. The hardware designers verified in simulation that properly configured, the hardware blocks work as expected. Meanwhile, the software designers, via register reads to the hardware, were able to confirm that the hardware blocks were configured correctly.

After employing an advanced silicon observability solution, the team was able to root cause the bug in a matter of hours. By directly observing the impact software instructions had on hardware operations the team was able to quickly recognize that, though software was setting the correct hardware configuration registers, the order of operation was incorrect and the hardware started processing data before it was fully configured.

Case Study 2 - Silicon Debug Improves Chip/System Quality

The second case study comes from the same team. Systems performance was far lower than expected, so poor that the system would be unviable in the market unless it was fixed. While chasing down the first bug the hardware designer noticed the hardware blocks were idle for far longer than expected.

Utilizing the ability to directly observe the timing relationship between software instructions and hardware operations identified a considerable number of areas for improvement in both software and hardware. They not only got their performance up to the target specification but continued making performance improvements to further improve the product.

Some of the performance improvements included:

- Reordering software instruction sequences to match hardware execution time.
- Creating a low-latency bus for frequently used hardware registers.
- Remove redundant software accesses to hardware registers.

Case Study 3 - Supplementing Proprietary ASIC Debug Capabilities

The final case study comes from a company which prototyped their ASICs before committing to silicon. This company employed a homegrown design for debug (DFD) solution on their ASIC.

This company experienced a bug that escaped to ASIC silicon. Fortunately, they found the bug before the customer, but they were still under tremendous pressure to issue a software fix before the customer encountered it. The ASIC had insufficient probe points and the probe points that existed could not be viewed in the combination required to debug the problem. The prototype team was called upon to recreate the bug in the prototype. Once the bug was recreated, two re-instrumentations and 2 days were all that was required to find the root cause of the bug. On the third day, a software fix was delivered and validated. The software fix was in the customer's hands on the fourth day.

Michael Sachtjen
Michael_Sachtjen@mentor.com
Joe Gaubatz
Joseph_Gaubatz@mentor.com
Mentor Graphics®