The Good, the Bad, and the Ugly of Silicon Debug

Doug Josephson
Intel Corporation
3400 E Harmony Rd, MS HRM1-1
Fort Collins, CO 80528
+1 (970) 207-8546
ddj@fc.intel.com

ABSTRACT

Silicon debug begins with the arrival of design prototypes and can continue well after a product has gone into production. It is perhaps the most exciting and challenging stage of the integrated circuit development process. This paper gives an overview of silicon debug, and describes tools and methods used during the debug process.

Categories and Subject Descriptors
B.7.3 [**Hardware**]: Reliability and Testing

General Terms: Design, verification.

Keywords
Debug, design for test and debug, characterization, validation.

1. INTRODUCTION

The purpose of debug is to identify and resolve any problems (or *bugs*) that exist in an integrated circuit to ensure that it operates correctly for customers over the specification range. The debug process can easily make the difference between success and failure of a design.

The term "bug" has been used for over a century to describe design problems. There are three categories of bugs: functional, electrical, and manufacturing/yield. Functional bugs, also called logic bugs, are where the logic of a design is improperly implemented. Electrical bugs are where the functional behavior of the circuit is correct, but it fails to operate correctly in some part of the specified operating region (i.e. at some voltage, temperature or frequency). Manufacturing/yield bugs occur when a circuit is sensitive to some variable in the manufacturing process and changes its operation as a result of manufacturing variations. For example, an analog circuit's operation may be sensitive to the gate length used for a transistor and may fail due to manufacturing variations between different dice.

The goal of debug is to find all bugs and eliminate them through design changes or other means before selling the device to a customer. This paper briefly describes the process, design features, tools and methods used to accomplish this goal. Additional background on these topics can be found in [1][2].

2. THE DEBUG PROCESS

Debug begins with the execution of a detailed plan to characterize and validate both the functional and electrical operation of a design. This can be a daunting prospect with a complex design. For example, exhaustively validating a 64-bit wide execution unit in a microprocessor functionally would require 2^{128} vectors be applied to the circuit being tested. As if this were not impractical enough, consider adding the requirements that operation needs to be checked over the entire voltage, temperature, frequency and process variation expected for the design, and that such a unit may be only one small piece of an entire design. It is impractical to exhaustively validate a design, and thus tradeoffs must be made during characterization and validation without compromising customer quality.

The entire debug process consists of three main efforts: *characterization, triage and debug*. Characterization is the process of collecting data on many devices using as much validation content as possible. The validation content may include focused testing,. where specific tests are written to exercise the device, or random testing, where random tests are generated to exercise the device. Characterization is normally run on manufacturing testers using patterns which are primarily used to test the device for defects, as well as in systems where the device will eventually be used. Functional characterization is usually run at the specified nominal operating point of the device, whereas electrical characterization explores the entire operation region of the device over voltage, temperature, frequency and using devices with variations in processing. A common method used during such characterization is the *shmoo* [2][3], where voltage and frequency are varied and plotted against each other in a graph to indicate where the device operates correctly and where it does not. Shmoo plots can be very important in understanding electrical failure mechanisms and offer clues about what sort of failure may be occurring during a specific test.

Once characterization data is taken, the process of triage begins. This is the analysis of the data to determine dominant modes of failure for further investigation. This can involve analysis of shmoo shapes, state information from the device at the completion of the test, etc. The goal of triage is to "bucket" failures into major common categories to allow the debug process to begin.

With identified categories of failures to investigate, the process of debug can begin. This usually involves one or more debug engineers methodically investigating the failure to understand it in detail, using a bevy of features designed into the device to support debugging it as well as tools external to the device.

There are five major steps in debugging a particular failure:

1) Control the failure
2) Isolate the failure
3) Root cause the failure
4) Expand the failure
5) Fix the failure

Controlling the failure involves determining sensitivities of the failure via experimentation. For a functional bug, this might involve trying slightly different operations or vectors than the failing one to see if they pass or not. For an electrical failure, it involves determining sensitivity to environmental parameters like voltage, frequency, temperature and process, in addition to experimenting with different data patterns and vector sequences to see how they affect the failure. Shmoos are often performed at this time to learn more about the failure.

Once sensitivities of the failure are understood and a solid failure is obtained, work begins on isolation of the failure. This involves determining when the failure occurs (temporal isolation) and where in the design the failure occurs (spatial isolation). Usually a combination of designed-in debug features as well as external tools are used during this stage, as discussed below. These first two steps typically dominate the effort expended during the debug process for a bug.

3. DESIGN FOR DEBUG

One of the key components to enabling the debug process to go smoothly is the implementation of features in the design to aid debug. This is known as *design for debug*, or DFD. In addition, there are a number of tools and methods which are used in concert with these design features once silicon has arrived to accelerate the debug process.

3.1 Pre-silicon Design for Debug

Over many years and through hard-won experience, numerous DFD features have been developed and added to designs to aid in debugging them [1][2]. Often these features also serve a purpose in helping to execute manufacturing tests on the device as well.

It is essential during the debugging process to be able to pinpoint both temporally (when) and spatially (where) a failure is occurring. This is enabled through DFD features which capture and access the internal state of the device and manipulate its internal behavior in a controlled manner.

Enabling capture of such information requires the addition of triggering circuitry to detect particular conditions during debug. This may be as simple as a counter that begins counting clock cycles after reset of the device and compares to a programmed cycle number of interest, or as complex as a programmable state machine similar to an external logic analyzer that can evaluate and compare many different signals inside the device against programmable criteria. An example might be to match against a type of instruction in a microprocessor, wait 7 cycles, and then compare an address to see if it matches a specific pattern. Once such a triggering condition is matched, the trigger circuitry can fire a signal to cause various debug events to occur within the device. Examples of such events include capturing data or manipulating an internal device clock to support debug.

Normally, access to the internal state of a design is limited due to the relatively small number of inputs and outputs of a device to the internal state contained within it. Fortunately, most complex chips implement a feature known as *scan* which allows visibility into the internal state elements of the design. While primarily intended to support manufacturing testing, scan can also be used to examine the internal state of a device during debugging. Scan is typically implemented by connecting all of the state elements in a design (e.g. latches and flip flops) together in a serial shift register, in addition to their normal parallel connections to logic. This allows data contained within the storage elements to be serially shifted out by clocking them with a special serial shift clock. Once shifted out, the data can be examined for clues as to where (spatially) a failure might have occurred within the device.

Scan can be classified into two types: destructive and non-destructive. Destructive scan typically requires the internal clock(s) of a device to stop to enable the serial shift clock to be used to scan the data out of the device. Such clock stops can be handled by the trigger circuitry described previously. Non-destructive scan consists of additional state elements added solely for debug purposes which have no functional purpose in the design. These “extra” elements can monitor signals in the device and capture them via the trigger circuitry describe previously as well. Once data is captured for a clock cycle of interest, the data can be scanned out while the device continues to operate (since these latches serve no functional purpose, they can be scanned while the clocks continue to run). This can be particularly advantageous in debugging failures in the system the device is used in, as it may be time consuming to set up a failure over and over to enable many cycles of data to be captured.

Another feature typically added to designs is the ability to capture information over many cycles into a buffer or cache. Often memory circuits internal to a design can be partially or completely taken over to capture information from within a device as it is generated over a number of cycles. Various signals can be multiplexed into such storage to allow a wide range of information to be captured. Again, the trigger circuitry described previously is vital to starting and stopping such captures. The disadvantage of this approach as compared to scan is that usually only a relatively limited set of signals can be observed without high design cost. However, this feature can be useful in rapidly narrowing the search to a particular region of time or location on the chip to allow more thorough (but much more time-consuming) examination via scan techniques.

The ability to manipulate the clock(s) used within a design is also a very important aid to debug. With the ability to move edges of clocks within the device (again, using internal triggering circuits), it is possible to exacerbate timing problems on signals and thus identify temporally when a failure is occurring. This is known as finding the *critical clock* of the failure. For example, on a tester, each individual clock cycle inside the device can be manipulated individually while running a tester vector over and over to identify the failing clock cycle. Once this is known, data can be captured via scan during that exact cycle to find where on the device the failure is occurring. Clock manipulation coupled with scan techniques can be a powerful automated method for locating failing circuits quickly within a design.

Two other interesting DFD features are process variation monitors and power supply droop monitors. Process variation monitors are identical circuits which are placed throughout a design which can be used to evaluate localized process effects. Typically such circuits are ring oscillators which are configured to be sensitive to various electrical process parameters (e.g. gate length, width, leakage, etc.) The frequency of oscillation is thus dependent upon individual processing parameters. By making measurements of the frequency of these oscillators, it is possible to evaluate variation in processing across a given die being debugged. This can offer valuable clues during the debug of electrical failures. Similarly, power droop detection circuits may be distributed in a similar fashion around a die to allow evaluation of power supply disturbances. This can also be helpful in spatially locating places within a die where circuits might be failing.

Once a failure has been identified, the ability to "work around" the failure becomes important to allow other failures to be found and fixed without having to actually fix the design. By implementing various methods of configuring the device, it may be possible to avoid the failing behavior. For example, in the event that a problem is encountered in a parity generation circuit, configuring parity errors to be ignored can allow forward progress to be made. Similarly, to avoid a functional bug, it may be possible to configure the triggering circuitry described previously to detect when a failure is about to occur and avoid it through alternative means (e.g. executing a substitute code sequence in a microprocessor).

An additional method of working around electrical failures which can be very helpful is to allow for the localized adjustment of clock edges to circuits [4]. By slightly moving edges of clocks to individual circuits, it is possible to avoid timing problems in the circuits. This can be particularly important when trying to increase the speed of operation for a device. Such movements can be controlled through scan configuration of the device or through more permanent means such as fusing.

One final useful feature is the ability to inject artificial error conditions or anomalous events into the hardware of the device to see how it responds. An example would be an error injection circuit in a cache which can mimic the effect of a cosmic ray event upsetting a memory cell. The ability to inject errors randomly can allow for testing of hardware correction circuits as well as software to recover from errors. Accelerating such testing is helpful as it is often one of the last areas of a design to be validated. Similar opportunities exist to affect electrical operation of the device – e.g. through randomly disturbing clocks by adding jitter or moving edges, or causing localized power disturbances on-chip to evaluate how the device responds.

3.2 Post-silicon Tools and Methods

Armed with spatial and temporal information about a particular failure to isolate it, several different tools can be used to confirm the root cause of the failure. This is usually essential to validate that the failure is completely understood and sometimes to also check out a possible fix before actually implementing it within the design. The ability to check out a fix before committing it to silicon is critically important, as revising silicon can be very expensive and time-consuming, which can affect overall product schedule adversely.

There are a number of tools which can be used to probe actual signals on a device to verify the root cause of a failure. One of the oldest is actual mechanical probing, where a very small mechanical probe is used to measure a signal on the device. This is a very time-consuming and error-prone activity due to the incredibly small dimensions involved in today's devices.

As a result, more sophisticated optical probing methods have become the standard for evaluating signals. These approaches all make use of the fact that most chips today are "flip-chip" packages, where the device's inputs and outputs are connected to a substrate from the front of the die. This allows the "back side" of the device to be probed (after appropriate processing) using optical methods. All these methods take advantage of the fact that silicon is transparent to infrared light, and they are either active or passive.

Two active methods are laser voltage probing (LVP) [5] and laser assisted device alteration (LADA) [6]. LVP is a method by which relative timing can be established between a group of signals. An infrared laser beam is applied through the backside of the devices and illuminates diffusions of transistors. By observing the reflected laser light from the diffusion of the transistor, it is possible to determine when voltage transitions occur on a given node. This tool can provide oscilloscope-like traces of individual signals relative to each other which is obviously extremely useful in debugging timing-related failures. LADA is an extension of this technique where a laser beam is rastered across the back of the device in a given area to change the behavior characteristics of individual transistors. This can be very helpful in spatially identifying where a failing circuit might be.

Passive techniques include emission microscopy (EMMI) [7] and time resolved emission (TRE), also known as picosecond imaging circuit analysis (PICA) [8][9]. Both techniques make use of the fact that CMOS transistors in saturation emit infrared photons. Through use of a photodetector, it is possible to assemble photon counts over an integration period to locate devices which are in saturation (usually devices which are either switching or are defective). TRE/PICA are a dynamic version of EMMI (EMMI is mostly used for failure analysis) which enables time resolution of when signals switch with picosecond accuracy. Since this method is passive, it does not affect the operation of the circuit in any way which can be an advantage compared to LVP.

Through use of these optical probing techniques, it is possible to get very accurate timing information regarding switching of internal signals of a device. This is useful to allow debug of either functional or electrical failures through the reconstruction of signal waveforms on a device for further comparison to simulation and data collected through scan and clock manipulation. Usually with all of this data it is possible to get to a definitive root cause for the failure in question.

Once root cause has been determined, a focused ion beam (FIB) edit [10] can be performed to either try to fix the failure or perturb the circuit to ensure that root cause is fully understood (i.e. either make the failure worse or better). A FIB edit is a form of silicon microsurgery where an ion beam is used to dig through a die to allow metal connections to be cut or made to individual transistors or to trim existing transistors. A FIB edit can be tremendously useful in correcting a functional or electrical bug and allowing it to be validated prior to fixing it. For example, a functional bug can often be corrected by rewiring signals and using "spare" gates which are scattered throughout a design for enabling such FIB edits. An electrical bug such as a speedpath from an underdriven buffer can be corrected by wiring in another buffer in parallel to help drive the slow signal. A FIB edit is an essential method to validate root cause and possibly avoid a bad/misunderstood fix before it is committed to silicon. FIB edits can also be useful for experimenting with suspected circuits when not much is understood about the failure (these are known as speculative FIBs and are similar to "exploratory surgery"). Sometimes a set of edits can expose sensitivities in the design which may not otherwise be obvious.

Once the root cause of the failure has been established, the final step in the debug process is to expand the failure. This has two components – the first is to ensure that the worst case conditions exist for future testing for the failure, and the second is to ensure there are no other similar types of failures in the design that might be the result of inadequacies in design methodology.

After completing the debug process, a fix may or may not be developed depending upon the severity of the problem and its impact on the product. If a fix is made, it is usually desirable to verify it via a FIB edit if possible to ensure the fix is robust. With verification of the fix, either in silicon via a FIB edit or through confirmation using design tools, the masks are changed and re-released to correct the failure, along with any other failures. Depending on the product type and the complexity of the design, it may take anywhere from one to ten or more design revisions, and 6 months to 2 years of work to complete the debug process to enable shipment of the product.

Case studies of actual bugs will be presented during the conference session to illustrate the use of the debug features, tools and methods described above. In addition, interesting examples of debug case studies can be found in [1][2][11][12][13].

4. REFERENCES

[1] Josephson, D., Gottlieb, B. *Advances in Electronic Testing – Challenges and Methodologies: Chapter 3 (Silicon Debug)*, pp. 77-108, Gizopoulos, D. (Editor), Springer, 2005, ISBN 0-387-29408-2.

[2] Josephson, D. "The Manic Depression of Microprocessor Debug", *Proc. IEEE International Test Conf.*, pp. 657-663, 2002.

[3] Baker, K., Beers, J. V. "Shmoo Plotting: The Black Art of IC Testing", *IEEE Design and Test of Computers*, Vol. 14, No. 3, pp. 90-97, July/September 1997.

[4] Naffziger, S., et al. "The Implementation of a 2-core, Multi-threaded Itanium™ Family Processor", *IEEE Journal of Solid State Circuits*, Volume 41, Issue 1, pp. 201-204, January 2006.

[5] Paniccia, M., et al. "Novel Optical Probing Techniques for Flip Chip Packaged Microprocessors", *Proc. IEEE International Test Conf.*, pp. 740-747, 1998.

[6] Eiles, T., et al. "Critical Timing Analysis in Microprocessors Using Near-IR Laser Assisted Device Alteration (LADA)", *Proc. IEEE International Test Conf.*, pp. 264-273, 2003.

[7] Bruce, M., Bruce, V. "ABCs of Emission Microscopy", *Electronic Device Failure Analysis*, pp. 13-20, Volume 5, Issue 3, August 2003.

[8] Tsang, J., et al. "Picosecond imaging circuit analysis", *IBM Journal of Research and Development*, Vol. 44, No. 4, 2000, pp. 583-603.

[9] Knebel, D., et. al. "Diagnosis and Characterization of Timing-Related Defects by Time-Dependent Light Emission", *Proc. IEEE International Test Conf*, pp. 733-739, 1998.

[10] Livengood, R., Medeiros, D. "Design for (physical) debug for silicon microsurgery and probing of flip-chip packaged integrated circuits", *Proc. IEEE International Test Conf.*, pp. 877-882, 1999.

[11] Huott, W., et al., "The Attack of the 'Holey Shmoos': A Case Study of Advanced DFD and Picosecond Imaging Analysis (PICA)", *Proc. IEEE International Test Conf.*, pp. 883-891, 1999.

[12] Josephson, D. et al., "Debug Methodology for the McKinley Processor", *Proc. IEEE International Test Conf.*, pp. 458-459, 2001.

[13] Pyron, C., et al., "Silicon Symptoms to Solutions: Applying Design-for-debug Techniques", *Proc. IEEE International Test Conf.*, pp. 666-672, 2002.