Reducing Time to Insight in Digital System Integration

Digital design teams are facing exponentially growing complexities and need processes and tools that reduce the time needed to gain insight into difficult system integration problems. This article describes modern digital systems in terms of the problems they create in the system integration phase. The debug cycle is described with special emphasis on the "insight loop," the most time-consuming phase of system integration. A case study from an HP workstation design effort is used to illustrate the principles. A new digital analysis tool, the HP 16505A prototype analyzer, is introduced as a means of solving these vexing problems more quickly by reducing time to insight.

by Patrick J. Byrne

The digital revolution is upon us in every form. Computer performance doubles every 18 months. Networks of high-performance servers are replacing mainframes at a dizzying pace. Personal communication systems are pervasive, from remote sales tools to medical information systems to networked workgroup tools.

What is behind this revolution? The answer is hardworking teams of engineers focused relentlessly on reducing time to market with the latest silicon and embedded systems. These teams of engineers are taking the latest silicon technology in ASICs (application-specific integrated circuits) and memory, exploiting them while adding value in the form of special application-focused architectures and implementations, and bringing the end products to market at the right time, price, and performance. These efforts are often made up of highly integrated subteams of four to eight engineers—hardware development specialists with expertise in ASIC design, printed circuit board system design, hardware architectural trade-offs, and the latest CAE (computer-aided engineering) tool methodologies. These teams specialize in designing high-performance, efficient, embedded software systems using the latest software design tools and languages. It is not unusual for design teams to use C++ and to have a carefully crafted multilayer software system with de facto APIs (application programming interfaces) between layers. These abstractions are used to manage the growing complexities of real-time embedded systems. These teams have to keep the customer in clear view as they make complex trade-offs between software and hardware implementation, between time to market and feature sets, between material cost (hence customer price) and component sourcing risks.

Concurrent Design and the Integration Phase

Figs. 1 and 2 illustrate the nature of the modern digital design process. Fig. 1 shows how the many competencies are brought together by the design team into the end product. The modern digital design process is highly concurrent. To make progress, each engineer makes assumptions about the other engineers' work. Sometimes these assumptions are well-documented, but often they are not because of the relentless schedule pressures. The risk accumulates throughout the design process because the tools used by different members of the design team do not explicitly articulate the interdependencies between subsystems. These design interdependencies are often found in the integration phase, when the whole design comes together for the first time. This is the well-known finger-pointing phase. "It's a hardware problem," says the software engineer. "It's a software problem," says the hardware engineer. The team members then sort out the problems from their respective points of view, engaging in the iterative and unstructured debug process.

Fig. 2 shows this development process in terms of cycle times. Twenty to thirty percent of the entire development process time is used in this unstructured integration phase. This phase of the design process is unstructured because there are so few well-designed tools and processes for engineering teams to use to work through the key integration tasks. In the integration phase, hardware meets hardware in the form of ASICs interacting with the circuit board. Interhardware design and modeling flaws are found here. Hardware meets software in the form of driver code running ASIC subsystems. There are also problems from application programs causing software integration problems. Application code provides an incredibly rich and complex stimulation system to the hardware.

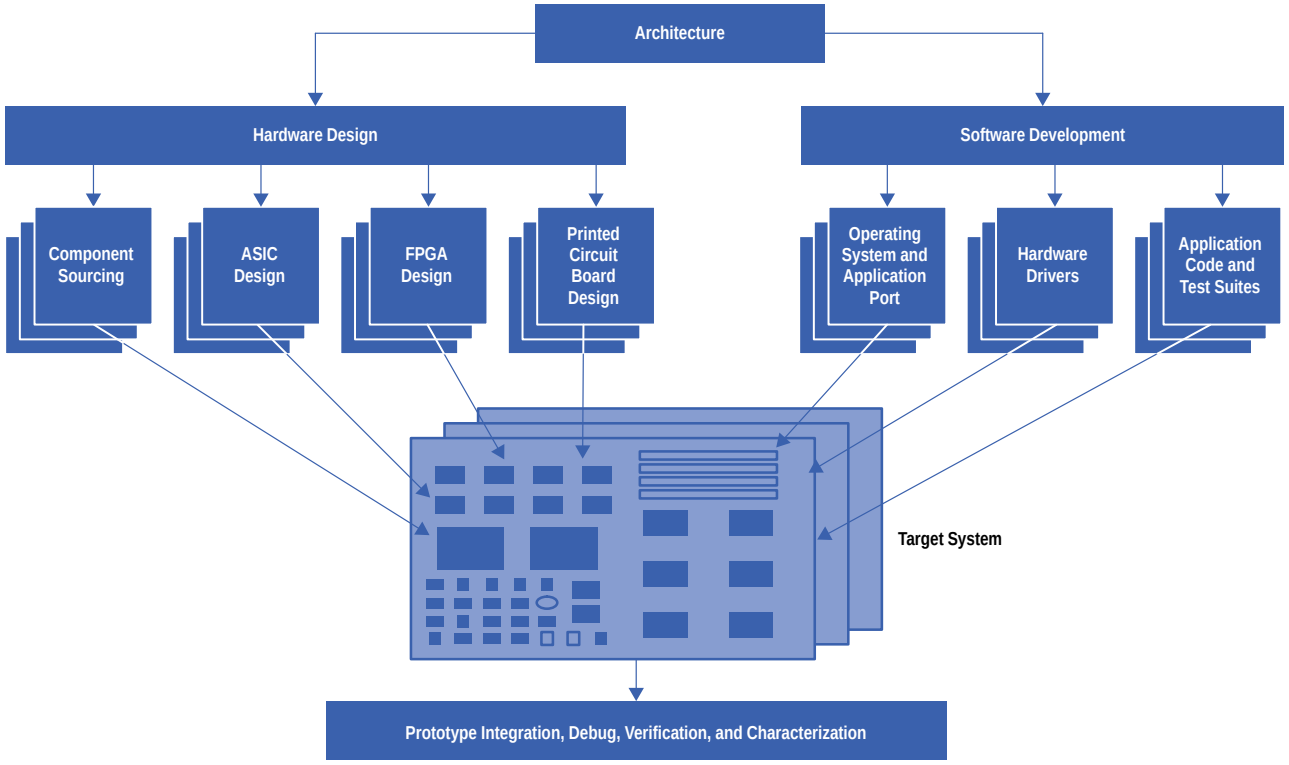

Fig. 1. *Digital system design process.*

Hardware and software modeling tools (simulators, compilers, logic synthesis, etc.) falter in their ability to model all these complex combinations, sequences, and data and code interdependencies. Fig. 3 shows the simulation execution speed of a typical 40-MHz system. Most digital circuit simulators execute at 10 to 30 cycles per second, leading to total system simulation times of days and sometimes weeks. This is too long for designers who are used to the iterative design and debug process. This shows that simulators are just too slow to expose real-world problems within the mental time constants designers need.

Hard Problems in the Real World

This rich and complex environment is the real-world environment indicated in Fig. 2. The modeled world in Fig. 2 is where designers work in isolation to predict real-world behavior. But it is very hard to predict every corner case to which the actual applications will take complex combinations of hardware and software. The real-world environment is the domain of the "hard problems," also called the "interesting cases" by some engineers. The software team has done their best to model the

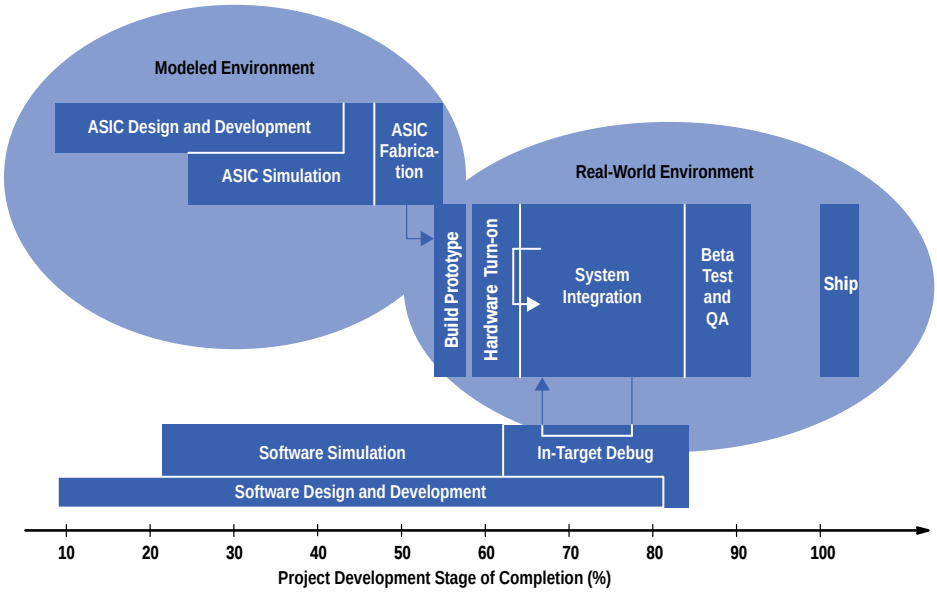

Fig. 2. *Stages in the development process for ASIC-based designs.*

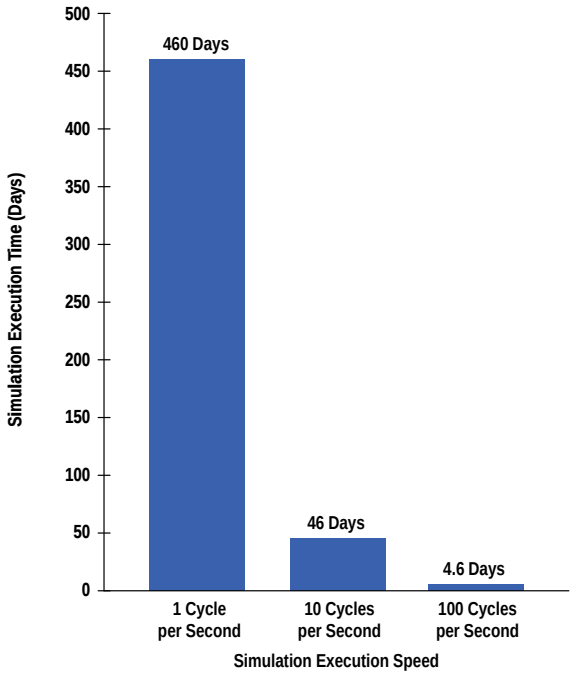

Fig. 3. *Time required to simulate one second of operation in a 40-MHz system.*

hardware. They have read the hardware documentation and written their driver and application code consistent with these models. The hardware engineers have done their best to model the software. Often, hardware engineers will standardize hardware interfaces specifically to avoid unmodeled software behavior. However, when the team puts the system together, they find unmodeled or undermodeled cases—the interesting cases. They find that their documentation doesn't cover some sequence of behavior by which the application code runs the ASIC into a state machine or switching condition that causes a glitch that delays response, causing the software to time out at the fourth layer of the protocol. It is very common to have the problems in the real world be separated, symptom from root cause, both temporally and spatially. For example, an ASIC on a peripheral bus has a state machine sequence that causes an error condition, but the errored packet is not sent back to the CPU until it fits into the priority stack. Or even more simply, the bad data is written to memory a long time before it is read and found to be faulty. The problem is detected later in time and at a different point in the system from where and when it was caused. Multiple bus architectures, such as Peripheral Component Interconnect (PCI), using intelligent queueing techniques, make these kinds of latencies commonplace.

Another common real-world problem is the infrequent problem that only happens at odd, nonrepeatable times because complex hardware and software systems have deep memory capacities and latencies in registers and state machines in addition to complex logical extent. The hard problems are those that the system stimulates through complex interactions between big application systems and deeply sequenced complex hardware systems. Odd pairings of hardware and software states cause infrequent behavior. Every engineer knows the problem of solving an infrequent system crash.

Another common real-world problem is the hidden dependency. This is caused by underspecified subsystems working together at blinding speeds. It is just not possible to specify every hardware or software component's ultimate use. Environmental conditions like temperature or application loading stress the system to expose these hidden dependencies. The case study described later in this article will show how simple and "perfectly specified" TTL parts can be the root cause of a very nasty operating system crash. This is typical of the hard problems.

Complexity: Growing Exponentially

The root causes of these kinds of vexing problems are hard to anticipate and hard to find. They delay projects and defy schedule prediction for managers. The reason they are so common is the exponentially growing complexity of today's systems. Silicon allows systems with 50,000 to 100,000 gates to be combined with 32-bit CPU architectures with out-of-order execution and large on-chip caches, controlled by 3M-to-10M-byte embedded software systems. These software systems are designed carefully with multiple layers of functionality to promote reuse and maintainability. These software layers, combined with poor code execution visibility because of caches and queues, can create indirection and poor observability during the integration phase.

Fig. 4 illustrates the exponentially growing complexities of today's subsystems. ASIC integration allows entire subsystems to be placed beyond the eyes of probes. Modern CPUs have 10 million transistors and multiple execution units. Embedded systems grow as well. All these technologies are brought to bear at varying levels of risk to be competitive today. Teams must master not only the individual technologies but their inevitable integration. The complexity simply overwhelms. It is

very difficult for engineering teams to predict all the interdependencies between such complex subsystems. Few engineers on a team understand enough of the details of each technology to predict the conditions that will cause a failure during integration. Engineers have to know their design tools: what they do right and what they do wrong. They have to know the system functionality and how it reflects down to the details of their design and the design of the next engineer. They have to know how to unravel the odd sequence of events that causes the infrequent system failures. Most hard problems must be solved by teams of engineers because the complexities of the interactions cross engineering disciplines.

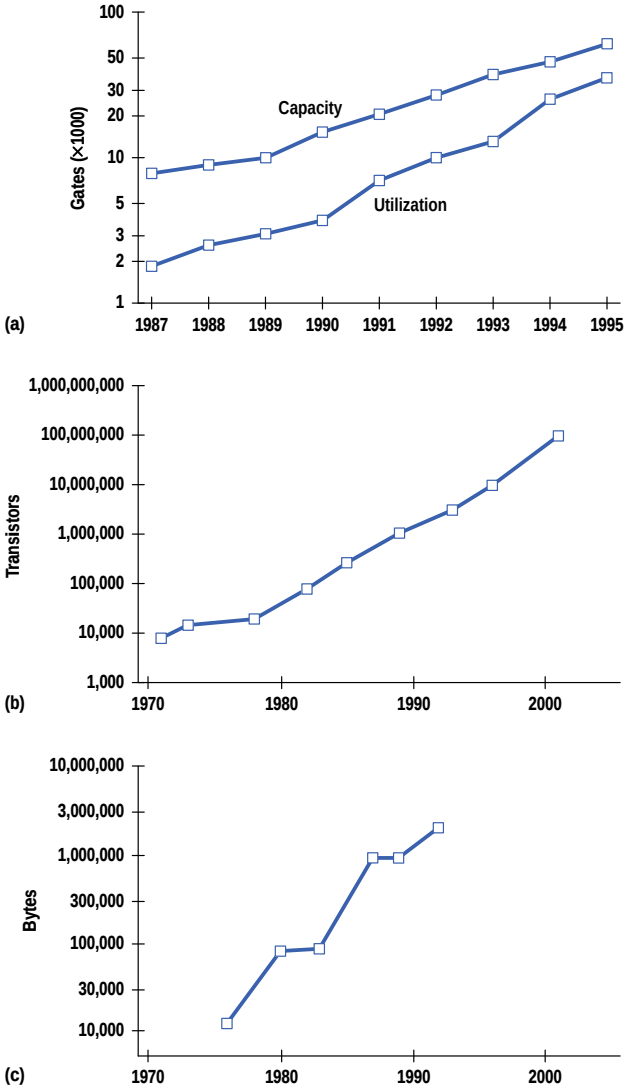

Fig. 4. *Increasing capability and complexity in digital system design. (a) In ASIC and FPGA hardware value added. (b) In microprocessors. (c) In software value added.*

Iterative Debug Loop

The process used by engineers to solve these complex integration problems that have eluded their design tools is illustrated in Fig. 5. The engineers start in the upper right corner, identifying the problem first by clarifying the symptoms.

The next step in the process, isolating problems correctly, is often tricky. Rumors and "old problems—old solutions" confuse. It is very common to jump to conclusions based on symptoms because the engineer once saw a problem like this before and solved it in a certain way. The correct approach is to perform what doctors call a differential diagnosis —describing in detail and with accuracy what has changed from the correctly working condition that could cause a problem, thereby isolating the unique operational sequence. This takes discipline and careful analysis.

The next phase is to discover the hardware or software dependencies that consistently describe the unique configurations and sequences to describe the problem correctly. Sometimes a binary search technique is used: code is sliced apart and patched back together to stimulate the hardware to the unique and small code segment that reliably repeats the problem. Experience and intuition are very important here because hard problems seem to have classic root causes but a variety of

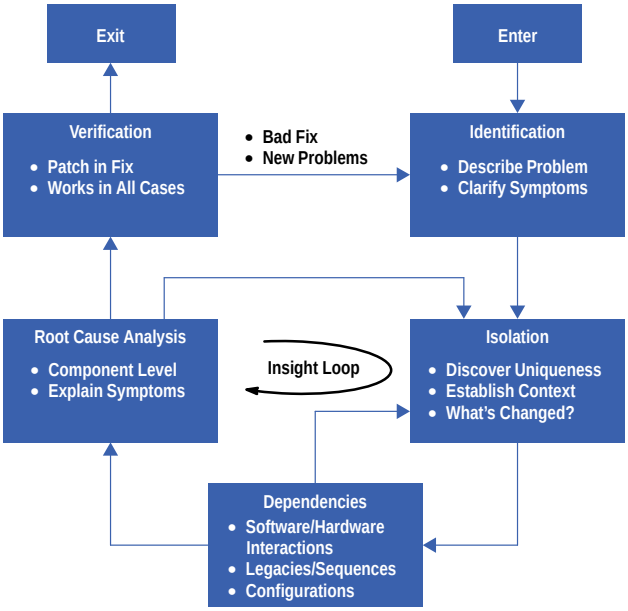

Fig. 5. *Iterative debug loop, showing the insight loop.*

symptoms, discovered over and over by each generation of engineers. Yet the process must be data-driven so that the team is not led to false conclusions.

It is often the case that only one in twenty engineers can correctly isolate the dependencies and sequences. This is because few people are chartered with the task and have the experience to understand the total picture of how a system works in enough pervasive detail to lead structured detective work. The next phase in the debug cycle is finding the true root cause. Infrequent problems, large unobservable ASIC systems, complex software systems, and time pressures make this task difficult. But it is very important to find the true root cause or a simple fix will unravel when some hidden dependency is invoked. A new solution causes another problem. A small change in temperature, voltage, or application loading will cause the fix to stress to the breaking point. Margins will be broken. This is the "insight loop": the structured detective work needed to synthesize the true root cause from confusing and overwhelming data sets caused by vast complexities. The ability to reach the true root cause rapidly and explain the symptoms, dependencies, legacies, and sensitivities is an art form and is deeply insight-based.

The root cause phase leads to the verification phase, in which the suspected fix is patched in and tested to see if it works in all cases. Often design members will go back to their design tools (simulators, etc.) to verify a fix because design tools provide a much richer method for controlling stimulation variables than the real world. It is hard to control a train wreck! The design team works hard to recreate exactly the parameters that caused the problem, and then they look for other related problems and symptoms. Unfortunately, a patched fix often breaks the system in a new way, causing new problems to occur. Or worse, the root cause was wrong and the fix doesn't work. Then the team starts over, having "wasted" time solving the wrong problem or only partially solving the right problem.

Required Tools and Processes

The time it takes to solve these complex problems correctly is highly variable and highly sequential. Therefore, it is very hard to predict or reduce this time systematically. The variability comes from the inner loop, called the insight loop. This is where the engineers labor over the vast complexities and try to connect detailed hardware and software component-level understanding with rich and complex sequences and interactions. The engineering team must traverse from operating system to switching sequences to solve the problems. They need tools that traverse these levels of design hierarchy and the attendant complexity in a way that preserves context but reveals details. The engineer must retain detailed understanding but in the context that describes the problem, always performing the deconvolutions that slice the problem into the subcontext. This is the great challenge of the insight loop.

The other reason for the long integration time is that problems are uncovered sequentially. One problem is solved and then another is uncovered. An operating system must boot before a critical performance bottleneck can be discovered. A clock distribution problem must be fixed before a bus timing problem is tackled, and so on.

To be competitive, design teams must use the latest technologies to get to the price/performance or sheer performance they need. The underlying technology pace causes the complexity, which causes the long integration phase. The design teams that master the integration phase are the first to market, since few companies have a sustainable advantage in underlying technologies. The winning design teams are able to master this phase consistently over several product generations while bringing unique architectures and implementations to important customer needs. Saving 20% of a design cycle may not seem

like much, but doing this four times in a row places the team fully one generation of products ahead of its competition, a crushing advantage in today's electronics markets. This is illustrated in Fig. 6. With product life cycles in the 6-to-18-month time frame, victors and vanquished are determined in just a few short years using this logic. The complexity is the blessing and the curse, the enabler and the nemesis for competitive market entry.

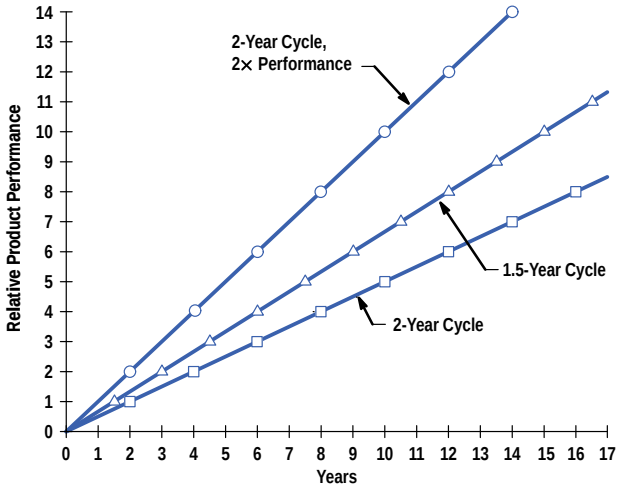

Fig. 6. *The cumulative effect of reducing design cycle times from two years (bottom) to 1.5 years (middle) is equivalent to increasing the performance of each product generation, as exemplified by the top curve.*

Design teams need processes and tools to manage this complexity during the integration phase so that they can, with confidence, take one step more in the use of modern technologies without going too far. Reducing the time to insight in the insight loop is how teams can master complexity rather than have it curse their design cycles with long and unpredictable times. The masters of the integration phase will win over the long term. These are the winning design teams. The tools and processes they need simultaneously provide insight into problem dependencies, reducing the time required to isolate the context, and solve multiple problems at once. Therefore, these tools must retain context while revealing details, and must not hide one problem with the symptoms of another. Without the right tools and processes, teams will suffer through multiple insight loops in sequence.

The Hard Problems: Summary

To summarize the nature of the hard problems and the debug process so that they remain in focus during the case study to follow, the hard problems are listed here in order of common occurrence and degree of pain to the digital design team:

- Hardware-software integration. Different tools used by different engineers. Very complex interactions. Data dependent.
- Displaced locality. Symptom is separated from root cause in location and time.
- Infrequent. Hard to repeat because of deep legacies in hardware and software subsystems.
- Hidden dependencies. Underspecified components find the corner cases.

The tools and processes needed in the insight loop provide the following:

- Design hierarchy traversal. Details are shown in the unique context that causes them.
- Correct isolation. The true and unique context of a problem is found using sequential exploratory logic. Symptoms are explained.
- Correct root cause analysis. The component-level source of the problem is found.
- Verification of proposed fixes, making it possible to move on to the next problem with confidence.
- Accuracy. One problem is not hidden with the symptoms of another.

HP Workstation Case Study

Following is a case study from an HP development to illustrate the above principles. Hewlett-Packard is a heavy user of advanced technologies and tools to bring high-performance products to market. The product developments are typically organized in integrated design teams that effectively design complete products. Therefore, HP is a good place to look for the challenges and best practices associated with the hard problems of complex systems. The case study presented here is described in significant detail in **reference 1**. Here it will be used to illustrate the nature of the hard problems and the iterative debug process (the insight loop) and the need to reduce the time to insight.

The case study is from the HP workstation group. The target system is one of the HP 9000 Series 700 high-performance HP-UX* workstations. The block diagram is shown in Fig. 7. It has a multiple-bus architecture with several custom ASICs,

and was designed specifically for high performance at a low cost. During the final product qualification phase, an operating system crash was found by a test engineer who was verifying various hardware and software configurations. The problem was infrequent and hard to duplicate. The system just crashed at odd times. The test engineer called together a team of engineers with the ability to traverse the design hierarchy with acumen.

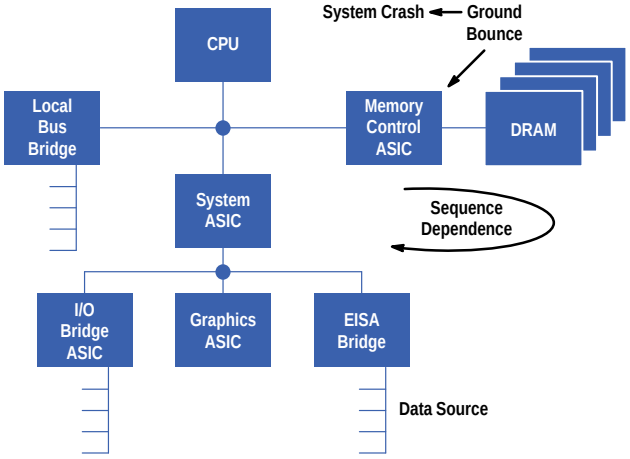

Fig. 7. *Block diagram of an HP workstation showing the location of the ground bounce causing the problem described in the case study.*

The first step was to identify the problem correctly. The team found that the operating system crashed only when the system was performing DMA (direct memory access) transfers from a SCSI drive on the EISA bus. The next step was duplicating and isolating the problem. It could have happened anywhere from the data source on the EISA card to the DMA cycle termination at the CPU bus. To isolate the problem, the data transfers had to be reduced in size and made repeatable. This is where the software team got involved, since they could assist in slicing the data sets into pieces that might recreate the problem. A binary search ensued, cutting and trying different data sequences until repeatability was achieved with a minimum failing sequence (MFS). Even this arduous trial-and-error process yielded only 90% repeatability. The 10% nonrecurrence was ultimately tracked to a DRAM refresh cycle that changed the timing of the sequence that caused the failure (all symptoms must be explained to ensure that a true root cause has been found. Often, teams will neglect inconvenient facts.) The team was now deeply in the insight loop, trying to find the sequences and context that would isolate the problem, leading them to the root cause.

The next step was to probe all data transfers from the EISA card into and out of main memory. For this task, they needed a logic analyzer and multiple, time-correlated, probing points. They looked through many large (1M-byte) data sets to find the failing sequences. Comparing data across the bus bridges is time-consuming, since the latencies are variable as a result of the intelligent queuing techniques employed in the bus bridge ASICs. Fig. 8 shows how the four bus transfers were observed to isolate the failing transfer. Finally, it was found that the MFS was isolated to memory writes and reads across the bus transceivers in the main memory system. The suspect part became a TTL bus transceiver, a common part “well-specified” by multiple vendors. The root cause of the problem was found with a high-speed oscilloscope time-correlated to the logic analyzer trigger looking at the error detecting logic in the CPU, as shown in Fig. 9.

The root cause was ground bounce inside the TTL part. A particular sequence of events placed the TTL part in a situation where a write enable to memory was asserted and directly followed by the simultaneous switching data transfer of the entire byte (Fig. 10). While this seems normal and is indeed within specifications, the closeness of the switching events, combined with the initial state conditions and the weak ASIC drive circuit nuances and memory system capacitive parasitics, caused a 2-to-3V ground bounce inside the TTL part. This caused the part to open up both bidirectional buffers, which caused temporary oscillation, which caused data corruption. This corrupt data, when read back to the CPU many cycles later (several microseconds in a 60-MHz design), caused the operating system to crash.

In the end, all symptoms were explained. Redesigned TTL parts were used and the memory system layout and the ASIC buffer design were improved. The breadth of experience and measurements needed to solve the problem was significant and is a true illustration of the insight loop. Vast data dependencies and odd sequences had to be explored and sliced through to find the MFS. Multiple points in the system had to be probed in time correlation. Data transfers had to be compared across bus bridges. In the end, the entire design hierarchy had to be traversed, from the operating system crash to the ground bounce triggered by data dependent signal integrity. To solve the problem, the entire design team had to be involved: the board designer, the ASIC designers, the software engineers, and the component engineers.

The Prototype Analyzer

The HP Model 16505A prototype analyzer, shown in Fig. 11, is a prototyping tool designed specifically to reduce the time to insight into the common and complex problems plaguing complex digital systems. The prototype analyzer architecture is

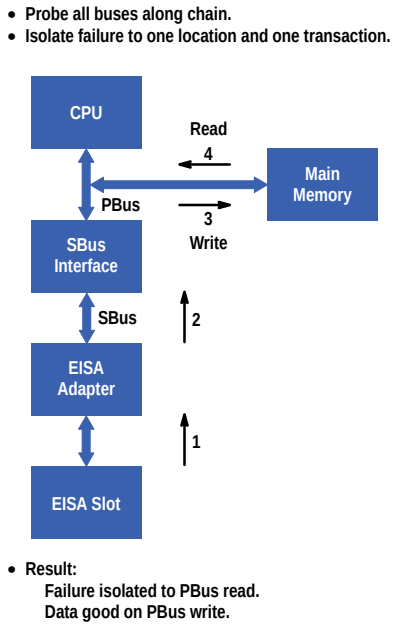

Fig. 8. *Four bus transfers had to be observed to isolate the failing transfer.*

described in ***Article 2***. It is based on a *measurement server architecture*, described in ***Article 3***. It is further based on the key concepts of smart data exploration, through a logically rich postcapture filtering capability, as illustrated in the case study and in ***Article 2***. The key software technology that allows the product to present multiple time-correlated views across the design and measurement hierarchy is called *normalized data* and is discussed in detail in ***Article 4***.

The key concepts of the prototype analyzer product are as follows:

- Insight into problem sources comes from the time sequence dependencies and is facilitated by sequential ordered exploration tools (filters). Therefore, time correlation, using global time markers, across measurements and across measurement domains is critical. Use of color and flexible user-definable data organization is required to aid the deep insights required to solve the hard problems.
- Most important problems are rare in occurrence. Therefore, the tool must quickly and intelligently filter out and eliminate large data sets that don't relate to the problem at hand. Furthermore, since the hard problems are rare, the tool must be able to keep data around to postprocess in every conceivable way to extract the maximum use from the valuable captured data (the cause is in there someplace!). Avoiding the rerun penalty with its associated time delay keeps problem-solving times short, the way engineers need them.
- Hard problems are solved by design teams, not by single members only. Therefore, the measurement server architecture needs to reside on the network and allow offline analysis by multiple members. Fig. 12 shows a common configuration of the prototype analyzer on a network, accessible to every design member. The easy

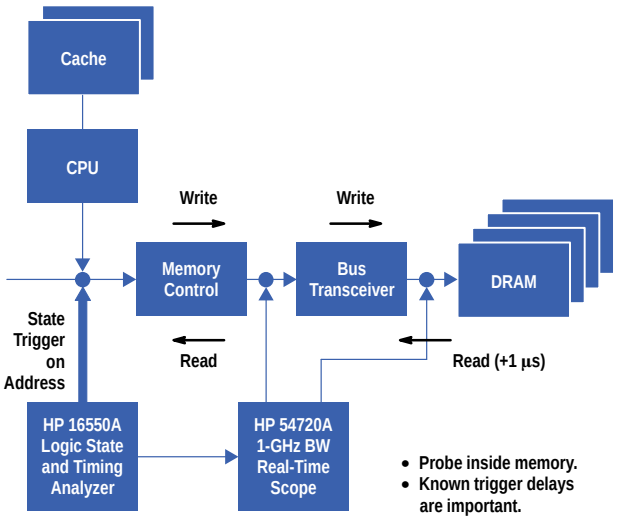

Fig. 9. *The root cause of the problem was found with a high-speed oscilloscope time-correlated to the logic analyzer trigger looking at the error detecting logic in the CPU.*

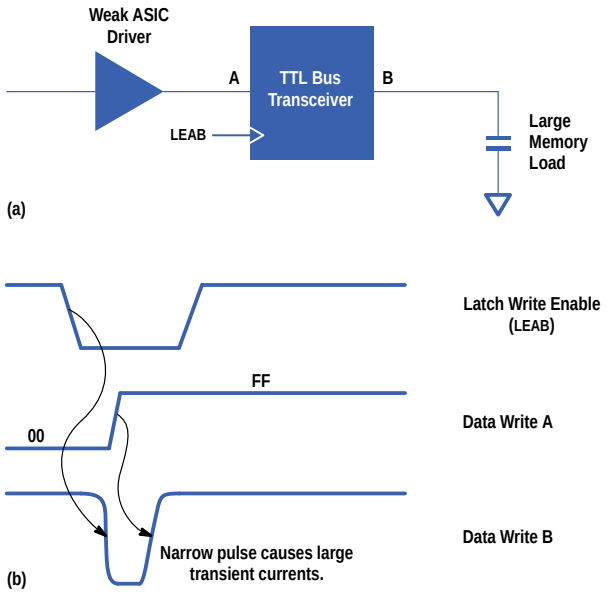

Fig. 10. *The root cause was ground bounce inside the TTL bus transceiver.*

communication of information across a local area network using standard file and server protocols facilitates team solving processes. Furthermore, the tool needs to provide team-member-centric views appropriate to each person and problem. For this reason, the HP 16505A prototype analyzer provides source-to-signals correlation, allowing correlated views from C source code to analog signals, all based on time-correlated measurements. The 16500B logic analysis system supports all these measurement modalities.

- Designers have vast intellectual property vaults in their design databases. These form the basic raw material for an engineer's analysis needs. They must be viewable and comparable to the measured data to accelerate time to insight. A file in, file out capability facilitates the easy movement of the critical data sets to where they can be manipulated with the best postprocessing tools available. Engineers have their favorite tools and the prototype analyzer accesses them all through the X11 Window interface.
- Large data sets are difficult to use to find the elusive root cause of a problem even if the problem has been captured in the data set. This is because the user is overwhelmed by the magnitude of the data and has to wallow around in it. Overview tools are needed that transform the low-level digital data and software code to a higher level where users can see the data that lies outside the norm. The human eye easily distinguishes such

Fig. 11. *HP 16505A prototype analyzer.*

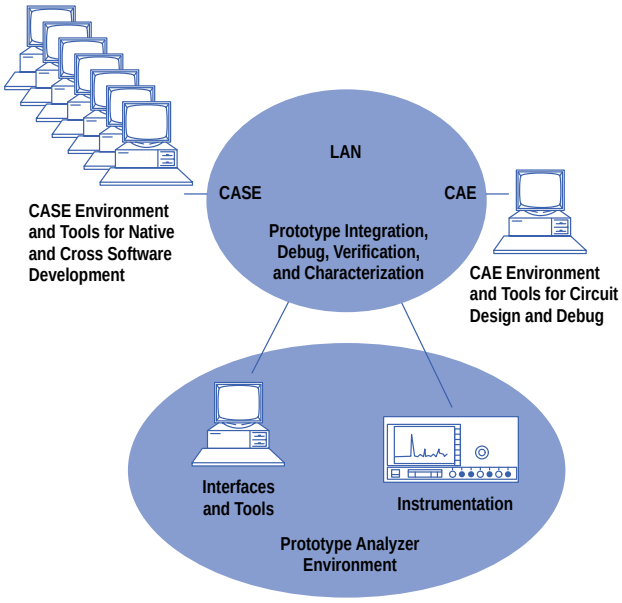

Fig. 12. *Team profile and design environment for digital system design.*

signals. The user wants to see changes in the data rather than just the raw data. This higher level is called the *modulation domain* and is a specialty of the prototype analyzer with its ability to create new labels from combinations of raw incoming data. An example of this is shown in Fig. 13. The analyzer is used to extract setup times on a bus from the raw incoming data and clock signals. A new label is created from the difference between the data and the clock. This label can then be displayed as it changes over large time constants using the chart tool. A user can easily view a million setup times over several milliseconds. Dramatic changes of setup time at periodic intervals tell the engineer that the system timing margins are being stressed on software time constants, perhaps as a result of a DRAM refresh or the servicing of a periodic interrupt. This display of the modulation domain is unique to the prototype analyzer among digital analysis tools and illustrates the power of postprocessing large data sets in overview displays to guide an engineer towards the elusive integration problem.

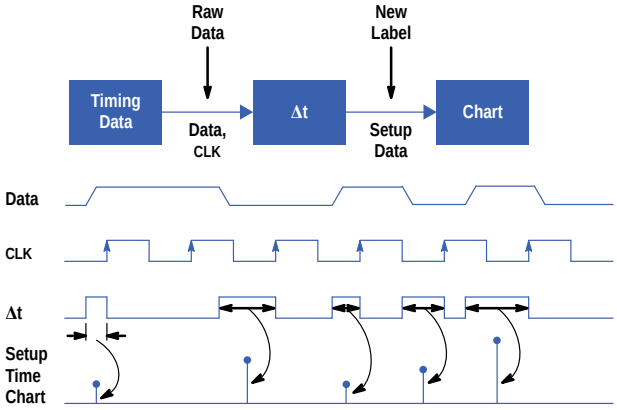

Fig. 13. *In the modulation domain, changes in the data, rather than just raw data, can be observed. In this example, the prototype analyzer extracts setup times on a bus from the raw incoming data and clock signals and then displays changes in setup time using the chart tool.*

Prototype Analyzer Solution

Now we revisit the HP workstation case study to see how the prototype analyzer can solve this problem much faster. The problem in the target system only occurred when simultaneous switching occurred on the data lines with write enable followed quickly by data write changes (refer again to Fig. 10). Fig. 14 shows how the prototype analyzer can be used with postprocessing filters to isolate the case much faster. The prototype analyzer is able to filter out states that follow specific sequences. In this case, Filter 1 in Fig. 14 could be set to view only the data for cases in which all states changed from ones to zeros (FF to 00) or vice versa. The design team suspected simultaneous switching noise in their design and this technique would have quickly given them the opportunity to explore this possible root cause without rerunning the target system. Any logic analyzer can find the first case of FF followed by 00, but only the prototype analyzer can scan an entire

one-megasample record, find *all* the cases where this is true, and *display them all* with the original data, time correlated with other views, such as analog data. Showing all the data, rather then just triggering on the first occurrence of this transition, allows the design team to find the one in a million times that this transition causes the failure, rather than having to keep retriggering the logic analyzer, hoping to find this case. The next step is to filter again using a timing filter (Filter 2 in Fig. 14). In this case, the design team is testing the theory that close timing between LEAB and data switching on data line A causes a narrow pulse, which, when driving the high capacitive load of the memory system, causes high ground currents. Only the prototype analyzer can find *all* of the occurrences of these conditions *and* only those that meet the Filter 1 criteria. This consecutive filtering allows engineers to build isolation techniques into their analysis to test their theories, rather than having to recapture the data and incur the retrigger and recapture time penalties. Using the oscilloscope built into the HP 16500B logic analyzer, they can capture with time correlation the lines that are being corrupted by the simultaneous switching noise and ground bounce. The time correlation allows engineers to retain the state and switching context that created the failing conditions. The waveform and lister tools merge the analog and code contexts to form a new measurement domain.

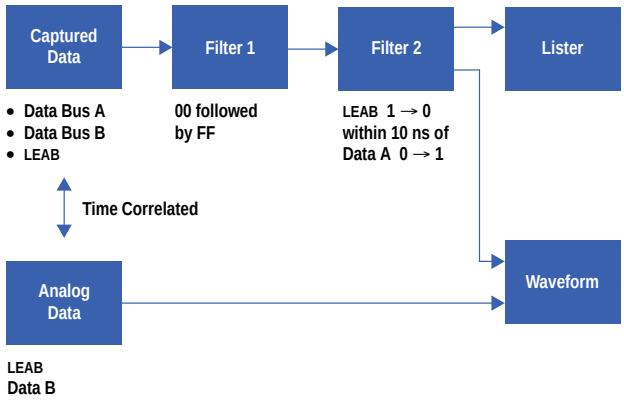

Fig. 14. *Solving the case study problem with the prototype analyzer.*

In summary, the prototype analyzer can be used to isolate the root cause of this problem much more quickly because it has sequential postprocessing filters, allowing analysis of entire data records at once. These kinds of problems are the cause of many delayed developments in advanced digital systems. The prototype analyzer tool is a way to solve them more quickly, minimizing time to insight for the design team.

Conclusion

Fig. 15 shows the use model for the prototype analyzer, redrawing Fig. 1 to show how the risk in the concurrent engineering process can be managed with a single digital analysis tool platform to unravel the complexities of modern digital systems and deliver to design teams the tools and processes needed to master the integration phase to competitive advantage.

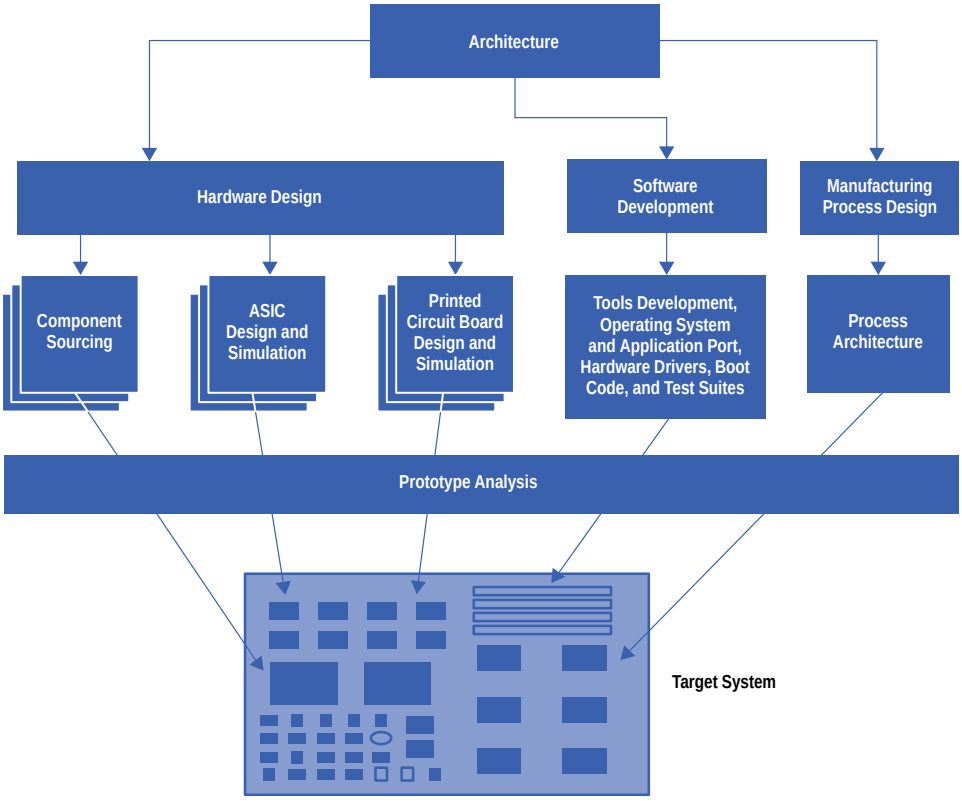

Fig. 15. *Updated system design process using the prototype analyzer.*

Acknowledgments

Pursuing the hard problems and delivering superior time to insight to digital design teams was the work of many people at the Colorado Springs Division. Greg Peters, Steve Shepard, Jeff Haeffele, Chuck Small, Jeff Roeca, and James Kahkoska were the leaders. James is the creator of the prototype analyzer concept. His unique ability to understand customer needs and translate these insights into compelling products and architectures was the magic behind this product. Fred Rampey, Dave Richey, and Tom Saponas sponsored the market and product initiative. Frank Lettang and Rob Horning of the Computer Organization were the case study agents and their easy access was critical to deep understanding.

Reference

1. *Hewlett-Packard 1993 High-Speed Digital Symposium Digest of Technical Papers*, pp. 10-1 to 10-20.

▶ Go To Next Article

▶ Go Table of Contents

▶ Go To HP Journal Home Page