INTRODUCTION

Emulation systems always seem to be labeled as either FPGA-based or processor-based. While the origin of this categorization is unknown, we do know that the classification is not technically accurate because the term “processor-based” describes architecture and “FPGA-based” describes silicon implementation. Historically, FPGAs were used in emulators to implement a gate-level mapping of the design. As designs increasingly included behavioral code, low-power design techniques, and complex clocking, the gate-level mapping became very inefficient. For example, a behavioral model of a 32-bit by 32-bit multiplier takes only about 25% of the hardware resources of a gate-level implementation.

Rather than creating a gate-level model in an FPGA, why not create a “computing element” in the FPGA to directly process the higher-level construct? This is the approach taken by Cadence® acceleration/emulation technology. From an architectural viewpoint, both emulation product lines from Cadence, the Palladium® series and the Xtreme® series, are “processor-based.” These product lines are distinct from competing “gate-model-based” emulators, which use FPGAs to emulate the design. This paper will give an overview of the processor-based approaches used in Palladium and Xtreme systems; contrast these with the gate-model FPGA approach used by other emulation vendors; and show why the gate-model FPGA approach was declared history (even though some companies continue trying to use it). It will also show applications of simulation acceleration and emulation and how they can be used much earlier in the design cycle—by designers looking for faster simulation performance, and by verification specialists needing to construct a complete system-level verification plan and implementation with lower risk and better predictability.

INTRODUCTION TO FUNCTIONAL VERIFICATION METHODS

Functional verification is often performed using logic simulation and FPGA prototyping. There are advantages and disadvantages of each, and often both are used. Logic simulation is easy, accurate, flexible, and low cost. Unfortunately, it’s not fast enough for large designs and it’s much too slow to run application software on the hardware design. FPGA-based prototypes are fast and inexpensive, but the time required to implement a large design into several FPGAs can be very long and is error-prone. In fact, some projects are finding they cannot implement their design in FPGAs at all due to capacity and technology limitations inherent in FPGAs. Changes to fix design flaws also take a long time to implement and may require board wiring changes. Since FPGA prototypes have minimal debugging capability, probing large numbers of signals inside FPGAs is nearly impossible, and recompiling FPGAs to move probes also takes a long time. Clearly simulation should be used earlier in the verification process when bugs and fixes are frequent, and prototyping should be used later in the process when the design is more stable and more
speed is needed to get sufficient cycles to uncover the last few bugs. Prototyping is also popular for testing software. As design sizes increase, there is clearly a gap between the logic simulator and the hardware prototype. There is a need to run faster while maintaining the feel and debugging capability of a logic simulator. Is using another system based on a gate-level netlist in FPGAs the best way to fill this gap, or would a system with a simulation-like compile and debug flow be more appropriate?

**SIMULATION ACCELERATION**

Simulation acceleration can address the performance shortcomings of logic simulation. To use simulation acceleration, the design is mapped into a hardware engine to run much faster and the testbench (and any behavioral design code) continues to run on the simulator on the workstation. A high-bandwidth, low-latency channel connects the workstation to the hardware engine to exchange signal data at high speeds. Signal-based acceleration (SBA) refers to the exchange of signal values on every clock cycle. Transaction-based acceleration (TBA) is a methodology to minimize communication by sending more information at higher levels of abstraction, usually bus transactions or messages, to achieve increased performance. Embedded testbench or synthesizable testbench (STB) achieves the highest performance by eliminating communication entirely with the workstation at the expense of behavioral language constructs and workstation services such as file I/O. Normally, the testbench on the workstation becomes the bottleneck, but with a very efficient transaction-based testbench, the communication channel may become the bottleneck. Figure 1 shows the approximate performance that can be achieved with simulation acceleration.

**IN-CIRCUIT EMULATION**

In-circuit emulation improves SBA and TBA performance by substituting “live” stimulus for the verification environment. This stimulus can come from a target system (the product being developed) or from test equipment, as shown in Figure 2. In-circuit emulation provides a much easier environment to implement design changes and a comprehensive, efficient debugging capability. Some of these emulation benefits come at the expense of performance and cost when compared to FPGA prototypes. Later we’ll discuss the creation and bring-up of emulation environments. Depending on the project and the type of software being developed, both FPGA prototypes and emulation can deliver enough speed to execute software.

Some of the trade-offs among the various execution platforms that have been discussed so far are shown in Figure 3.
HISTORY OF EMULATION

To best understand the need for processor-based emulation systems, and the limitations of gate-level FPGA emulators, it's best to examine the past and see how the emulation market has evolved. Commercial emulation dates back to the 1980s when FPGAs reached sufficient capacity to enable the creation of the first emulation systems.

EMULATION TIMELINE

1980s  IBM develops simulation acceleration EVE family (Engineering Verification Engine)
1988  Quickturn invents gate-level FPGA-based emulation with RPM, 25Kgates
1992  Quickturn delivers Enterprise, 330Kgates
1993  Quickturn delivers Mars, 500Kgates
1993  Merger of Pie into Quickturn
1994  Quickturn delivers System Realizer, 6Mgates, 500KHz–1.2MHz, commercial FPGA
1995  Meta Systems in France delivers Sim Express, 1.4Mgates, 250KHz–1MHz, custom FPGA
1995  Mentor abandons emulation business, sells patents and technology to Quickturn
1995  Virtual Machine Works, nearest neighbor architecture with heavily multi-plexed interconnect
1996  Ikos acquires VMW
1996  Synopsys delivers Arkos system
1996  Mentor acquires Meta Systems
1997  Quickturn delivers CoBALt based on IBM hardware, 6M gates, 50-150KHz, processor-based
1997  Synopsys sells Arkos to Quickturn
1998  Quickturn delivers Mercury, 3.5Mgates, 500 - 900KHz, RTL and behavioral emulation (SimServer), FullVision, commercial FPGA
1998  Mentor Graphics attempts hostile takeover of Quickturn
1999  Cadence acquires Quickturn
2000  Quickturn delivers MercuryPlus, 20M gates, 300KHz–1MHz, custom FPGA
2000  Mentor Graphics delivers Celaro, 7M gates, 600KHz–1.2MHz, custom FPGA, 100% visibility
2001  Axis announces Xtreme emulator based on reconfigurable computing technology
2002  Cadence delivers Palladium, 128M gates, 250–750KHz, processor-based, FullVision
2003  Mentor Graphics delivers Celaro Pro, 12M gates, 600KHz–1MHz, custom FPGA
2003  Synopsys attempts—but Mentor Graphics acquires Ikos
2004  Verisity acquires Axis
2004  Cadence announces Palladium II, eighth generation of emulation technology
2004  Verisity/Axis introduces Xtreme Server, 50M gates, 250–500kHz, 12 users
2005  Cadence buys Verisity/Axis

GATE-LEVEL EMULATION USING FPGAs

FPGAs were an obvious choice for emulation, and all emulation products (except Synopsys’ Arkos system) used them initially. Since the largest ASICs being emulated were much larger than the largest FPGAs, multiple FPGAs were required along with solutions for design partitioning and interconnecting the FPGAs. Multiple partitioning algorithms had been available and well understood for some time, but interconnect was a new challenge. “Nearest Neighbor” provided a “rich” interconnect to FPGAs physically near each other and a “lean” interconnect for “distant” FPGAs. Connections could be made through FPGAs, but this used up valuable pins. As the number of FPGAs required for emulation increased into the hundreds, a more flexible and balanced interconnect was needed because too many FPGA resources were going unused—there was no way to interconnect them—and many I/O pins were wasted because they did not match the needs of a particular design. Quickturn pioneered the partial crossbar interconnect, which separated the interconnect problem from the partitioning problem. This solved the interconnect issue for a time by allowing any pair of FPGA pins to be connected based on the configuration requirements of the specific design via the programmability of the partial crossbar.

Virtual Machine Works (bought by Ikos, then Mentor Graphics) used FPGA pin multi-plexing to increase the apparent pin count on each FPGA package and termed this “Virtual Wires.” The large variability in FPGA timing wasted a large amount of each wire’s bandwidth. As the pin-multi-plexing factor increased in subsequent generations, emulation performance decreased to undesirable levels. FPGA-based emulators were further plagued by compilation failures. Based on heuristics, the partitioning software would carve up the design into blocks that it thought would fit into a single FPGA based on gate count and number of block I/Os. Sometimes a block wouldn’t fit and the software would have to re-partition or split a block and try again. Sometimes, after many hours or even a few days, the compile just wouldn’t succeed and the user would have to intervene manually.
Ultimately, gate-level FPGA-based emulation was doomed by four factors:

1) Limited I/O pins that restricted how the increased FPGA gate capacity could be used
2) FPGA place and route time (which can take hours)
3) The need for 100% signal visibility for debugging
4) The difficulty of controlling timing through FPGA logic

The first three of these are a result of the difference in market requirements for commercial FPGAs and FPGAs used in emulators. Commercial FPGA vendors are driven by the mass market for FPGAs; emulation is a tiny market by comparison. The market for commercial FPGAs is characterized by products with one or two large FPGAs and a relatively low pin-to-gate ratio. Additionally, since compilation is done infrequently, compile times of several hours is not a problem. Since debugging is done via simulation, minimal debug capability is built into the FPGA (compared to emulation's needs).

The characteristics that emulators require from FPGAs are markedly different. An emulator will have many hundreds of FPGAs. Consequently, emulators require an FPGA with very high pin-to-gate ratio. Since hundreds of FPGAs have to be compiled, a compile time of multiple hours each is intolerable. These differing requirements are summarized in Figure 4.

UNUSABLE FPGA CAPACITY

As with all silicon, FPGA gate capacity is increasing according to Moore's Law, doubling every 18-24 months, but packaging technology is not keeping pace and the number of pins is not increasing fast enough to be able to use all these gates. Also, high pin count packages are not usually needed for commercial FPGA applications, so this is not likely to change. The result, shown in Figure 5, is that there is no need for today's large FPGA to include more pins to better suit emulation.

Debug Visibility in Commercial FPGAs

Although there has been some progress to add more debugging features to FPGA, it is still a difficult task to provide a complete picture of what is happening inside an FPGA. In the past, users would need to route a few signals of interest to spare pins and probe them with a logic analyzer. Additionally, a scan chain of all flip-flops could be read out of the FPGA when emulation was stopped to view the current state only. If probing of an unforeseen signal was needed for debugging, a probe would be added and the design recompiled. Adding signals to the logic analyzer trigger conditions also frequently required design recompiles. Some gate-level FPGA-based emulators implemented a 100% visibility capability, but this consumed about 30% or more of the capacity of the system. The latest FPGAs now have provisions for compiling logic analyzer blocks into the FPGA, but in the context of emulation with 10s or 100s of FPGAs it is not practical to recover the data and correlate it with the original RTL source code within a reasonable time.

Timing Control in FPGAs

The time delay along a path through several FPGA cells (and perhaps chips) is highly variable and unpredictable. When designing a product with a few FPGAs, engineers will often spend several days or weeks getting the timing delays adjusted so the FPGAs run at the desired speed. Obviously, this is impossible for several hundred FPGAs in a gate-level FPGA emulator where compile time needs to be a couple of hours or less. As a result, emulators must be very conservative on timing and emulation speed is unpredictable from compile to compile.
CLOCK HANDLING IN GATE-LEVEL FPGA-BASED EMULATORS

Clock skew must be managed carefully in gate-level FPGA-based emulators (just as on silicon) to avoid creating timing errors. Various methods have been employed in commercial gate-level FPGA emulators to accomplish this. Some emulators have dedicated clock input pins, which are buffered and propagated to the FPGA clock pins. When gated clocks drive a large area of the design, clock path duplication has been used to maintain low clock skew to all the relevant FPGAs. Some emulators force inputs to be synchronized to a clock. Some emulators use a process called “timing re-synthesis” to create a pseudo-synchronous emulation model. None of these techniques is without drawbacks. In some cases, the user must manually enter delays to get the design to emulate correctly. In other cases, the ability to handle some circuit constructs—like multi-plexed clocking and asynchronous set and reset on flip-flops—is sacrificed.

PROCESSOR-BASED EMULATION AS USED IN PALLADIUM SYSTEMS

In the early 1990s, IBM pioneered a different emulation technology, which was an offshoot of earlier work they had done in hardware-based simulation engines. The hardware technology consisted of a massive array of Boolean processors able to share data with one another, running at very high speed. The software technology consisted of partitioning a design among the processors and scheduling individual Boolean operations in the correct time sequence and in an optimal way. Initially, performance could not match gate-level FPGA-based emulators, but compile times of less than an hour, and the elimination of timing problems that plagued gate-level FPGA-based systems, made the new technology appealing for many use models, especially simulation acceleration. Future generations of this technology eventually surpassed FPGA systems in emulation speed while retaining the huge advantage in compilation times. Advances in software technology extended application of processor-based emulation to handle asynchronous designs with any number of clocks.

Other extensions supported 100% “FullVision” of all signals in the design, InfiniTrace* visibility of all signals at any time from the beginning of the emulation run—and dynamic logic analyzer trigger events without recompilation. At the same time that the emulation speed of gate-level FPGA-based systems was decreasing, new generations of processor-based systems were not only increasing in performance, but were also scaling capacity to hundreds of millions of gates.

A = B + C \cdot D \cdot \bar{E}

Figure 6: Logic simulation—CPU does Boolean math on signals and registers

One type of simulator, a levelized compiled logic simulator, performs the Boolean equations one at a time in the correct order. (Time delays are not relevant for functional logic simulation.) If two ALUs were available, you can imagine breaking the design up into two independent logic chains and assigning each chain to an ALU, thus parallelizing the process and reducing the time required, perhaps to one half. Figure 6 shows how a processor-based emulator uses tens of thousands to hundreds of thousands of ALUs, which are efficiently scheduled to perform all the Boolean equations in the design in the correct sequence.
PROCESSOR-BASED
XTREME ARCHITECTURE

The second processor-based emulation product series offered by Cadence was acquired from VeriSil in 2005 and is called Xtreme. Although the Xtreme architecture uses FPGAs, they are used in a very different way compared to gate-level emulation. Xtreme implements a simulation algorithm based on reconfigurable computing (RCC). RCC configures the hardware structure to match the algorithm and selects the best resources for a particular task with maximum parallelism. For example, if a particular algorithm can take advantage of six arithmetic logic units with addition and subtraction as its only instruction, RCC will select the best hardware resource structure with those attributes for maximum efficiency.

RCC comes in two flavors: static and dynamic. Static RCC refers to the situation of having predetermined and fixed resources during execution. The resource allocation is performed at compile time when the algorithm is being analyzed. On the other hand, dynamic RCC refers to the situation that different algorithm requires different resources during execution. Depending on the exact location of execution, different resources are swapped in on a needed basis. For example, if during execution of a RCC program a different arithmetic logic unit is needed to efficiently run the algorithm, dynamic RCC will swap in the needed resource when running while static RCC will have loaded the predetermined resources before execution.

Xtreme applies RCC technology to accelerate functional verification by orders of magnitude while preserving the original debugging environment by using FPGAs to implement parallel logic simulation.

FUNCTIONAL SIMULATION USING RCC TECHNOLOGY

To fully take advantage of the merits of RCC technology, the best algorithms are those that can be massively parallelized with consideration of specified processors. Functional verification naturally falls into this category since evaluation of RTL and gate-level constructs can be accelerated with massively parallelized RCC processors.

Current design methodology involves describing the design in a language such as Verilog® or VHDL. The language is separated into three categories: behavioral, RTL, and gate. Behavioral constructs usually describe the system testbench and are most efficiently simulated on a microprocessor because they must be executed serially. RTL and gate-level constructs describe the design and can be compiled into RCC elements for parallel execution. Most behavioral constructs are extremely difficult to parallelize and the sequential nature of a microprocessor is the best resource to simulate them. In contrast, RTL and gate-level constructs are written for parallel execution. Each RTL or gate-level statement can be mapped into a computing device specifically designed to efficiently execute the instruction. The RCC architecture for functional verification achieves its high speed by having dedicated processors to form a massively parallel structure of computing elements specially configured for each design. A computing element is a small compact processor dedicated to perform one function. For example, Xtreme has custom computing elements to simulate Verilog “case” and “if” statements.

When executing, the RCC engine obtains instructions and data from the microprocessor, and sends the execution command to the SIMD controller, which sequences the evaluation and communication of all RCC computing elements. The controller’s next step is to collect all evaluation result from computing elements, pack them in a data stream format, and send the resulting data back to the microprocessor to continue simulation.

By mapping the design RTL constructs onto its custom interconnected computing elements, the RCC hardware is programmed for maximum performance execution for each design being verified. Using a proprietary systolic array interconnection architecture, communication between computing elements and between multiple devices is fast and efficient. The event-based simulation-like algorithm implemented by the RCC engine uses dynamic event signaling to communicate events between the computing elements. With RCC, many of the limitations of using commercial FPGAs are overcome. There is no one-to-one mapping of design wires to FPGA wires in the emulator; only event messages are sent between FPGAs over a high-speed shared bus. This eliminates the previously described interconnect problems in gate-level FPGA emulation systems.

So far we have covered the basic differences of the modern processor-based emulators vs. the gate-based FPGA emulators. Next, we will discuss one of the most difficult problems for gate-based emulators and FPGA prototypes: the ability to transfer a design and verification environment from simulation to hardware and back again in an easy way.

THE VERIFICATION GAP

In most of today’s design efforts, verification consists of disjointed environments such as logic simulation and hardware prototyping. Gate-based FPGA emulation is also another disjointed environment. In effect, the project team must create, operate, and maintain multiple distinct verification environments with differing characteristics. Within each environment there are often micro-environments. For
example, the use of IP blocks from multiple vendors may require several different simulation tools. Creating, operating, and maintaining a number of different verification environments has a number of disadvantages:

- Time intensive: Each environment requires a great deal of setup and configuration. Very little of this work is transferable between environments.
- Resource intensive: Separate sets of tools require tool specialists.
- Risk intensive: The biggest potential problem of multiple verification environments is the real possibility of incomplete verification. Moving from simulation to gate-based FPGA emulation or FPGA prototyping requires a “leap of faith” since successful simulation results do not guarantee the FPGA solution will work.

The best way to overcome the verification gap described is to avoid gate-based emulation altogether and use an emulation system that behaves much more like logic simulation. Understandably, some projects require FPGA prototypes, but using them only for situations that truly require them also helps overcome the verification gap. Situations such as in-system testing at a customer or partner location, or pure software debugging tasks where no hardware visibility is needed, are examples of using prototypes in a way that does not try to make them into verification tools. The next section discusses the details of how to overcome the verification gap and how projects can logically move out of simulation into emulation by introducing new variables one at a time and avoid the “leap of faith” mentality.

IN-CIRCUIT VERIFICATION METHODOLOGY

The transition from logic simulation to in-circuit emulation involves many interdependent variables. Attempting to introduce all variables at once and move from the simulation domain to the emulation domain makes it difficult to determine the cause of problems. The process of compiling the design and mapping it into the emulation platform as well as building a target board introduces many new variables. An error in any part of the setup can cause the emulation to function incorrectly. Some of the areas included in crossing from simulation to in-circuit emulation include:

- Functioning emulation database
- Working target board
- Clock configuration of target board and emulator
- Physical interface between target board and emulator

The best way to minimize in-circuit emulation bring-up time is by utilizing the different In-Circuit verification modes to logically introduce only one or two variables at a time instead of introducing everything simultaneously.

Initial verification work on this design involves using logic simulation to verify each of the major blocks is working correctly. A bus functional model for the CPU bus can be used to test the memory control logic and register interfaces, and a bus functional model for the PCI Express interface can be used to verify I/O bus operation. Block-level verification is done using a software simulator and block-level verification environment. During this stage, lint tools are used to check syntax and synthesizability, and code coverage tools are used to confirm the block is sufficiently tested. Other tasks may include protocol compliance checking, protocol coverage checking, directed and random testing.

After each block has been tested independently, the parts are brought together to form a full-chip simulation. Software for the CPU is loaded into the Verilog memory models and executed. A testbench is used to generate transactions on the PCI Express bus interface. Assuming the design is in the one to five million-gate range, the simulation environment will probably run about 100 cycles/sec. This is just fast enough to see some of the initialization software run and configure the system, but not fast enough to simulate a meaningful quantity of data traffic. To increase performance of the simulation environment, simulation acceleration mode is used. The design under test and memories are mapped into hardware, and the PCI Express verification environment can continue to run in software simulation. This acceleration typically provides a 10–100x performance improvement compared to software simulation. Performance in the range of 1,000 to 10,000 cycles/sec is achieved. At this performance level, it is possible to simulate much more data being processed by the design.

**Figure 7: Example design**
Using simulation acceleration mode not only provides faster performance sooner in the project, but also serves to verify the mapping of the design into the emulation platform. When moving to in-circuit emulation, this is one of the variables that can cause problems. For gate-based FPGA emulators, the transition from event-based simulation to an FPGA netlist can be difficult. Even with processor-based technology there are still possibilities of mismatches when moving from simulation into emulation. Common causes are race conditions and timing or delay dependent behavior. Verilog does not have well-defined event-scheduling semantics and it is common for code to work on one IEEE-compliant simulator and not another because of event-timing dependencies.

**IN-CIRCUIT SIMULATION**

Part of the design verification plan requires in-circuit emulation to provide additional testing of the PCI Express interface. In-circuit operation can provide many more combinations of bus transactions and error conditions that are too tedious to create using a testbench. The necessary target boards and test equipment are assembled in the lab for in-circuit emulation. Initial bring-up is done using In-Circuit Simulation (ICS). ICS uses the emulator as a pass-through connection to the target system. ICS also eliminates an important variable from the bring-up: the emulation database. Remember, the majority of the database has already been verified. Using ICS allows design changes to be recompiled very quickly since no mapping to hardware is being done. Basic functions, such as reset sequencing and system initialization, can be closely monitored and easily debugged with ICS. You can single step through each clock of the simulation. A diagram of ICS is shown in Figure 8.

![Figure 8: In-circuit simulation](image)

**IN-CIRCUIT ACCELERATION**

After the interface between the target board and the emulator has been validated using ICS, the next step is in-circuit acceleration. ICA offers all of the performance benefits of simulation acceleration with the added feature of a target system. With ICA, behavioral Verilog models and C models can continue to run in the software simulator. Performance levels are equivalent to simulation acceleration. A diagram of ICA is shown in Figure 9.

![Figure 9: In-circuit acceleration](image)

**IN-CIRCUIT EMULATION**

When all desired test sequences are verified using ICA, the next step is in-circuit emulation. ICE gives the highest runtime performance. When it is time to enable ICE, nearly every variable of the transition from simulation to emulation has already been verified. The design under test mapping to hardware has been verified during simulation acceleration and ICA. The target system interface has been debugged with ICS and tested again with ICA. The last step is to remove all testbench code and enter ICE mode. With ICE, the emulator now takes control of the simulation event scheduling. A diagram of ICE is shown in Figure 10. Even though performance levels may be similar to gate-level FPGA emulation methods, In-circuit verification provides simulation-like debugging. All of these features make running emulation feel just like running a software simulator. Notice Figure 10 shows a dotted line to the workstation. Even in emulation mode it is possible to stop the emulation using Ctrl-C just like a software simulator. ICE allows additional software to be tested that was not possible with simulation or even acceleration performance levels. Real-time operating systems (RTOS) and other application software can be run at speeds acceptable to software engineers.

![Figure 10: Diagram of ICE](image)
An in-circuit verification methodology has been presented to unify the simulation and emulation domains for applications that use a target system. In-circuit verification provides a smooth transition from software simulation to emulation with consistent features and debugging. In-circuit verification results in faster bring-up time for In-Circuit Emulation and higher performance earlier in the design cycle.

**SUMMARY**

This paper presented background information on how the emulation industry started by using FPGAs for gate-level emulation and has evolved to provide processor-based emulation systems that are easier to set up, run at comparable speeds to gate-level FPGA models for designs of average complexity (processor-based systems run faster for high-capacity designs over 20M gates), and provide simulation-like debug features. While acknowledging the need for FPGA prototypes for specific validation tasks, it's clear the future of high-performance verification will be based on processor-based simulation acceleration and emulation. Engineers evaluating gate-level FPGA emulation systems should take a hard look at the actual benefits provided and focus on verification quality and return on investment vs. an exclusive focus on low cost.

As an example, a European design team that has relied on gate-level FPGA technology as a verification tool in the past recently switched to processor-based emulation. The project was a relatively small design, about 1M gates, and would seem to be a good fit for FPGA solutions. Some key conclusions were highlighted following the first successful project:

- **Faster turnaround time compared to FPGAs,** specifically the ability to go from RTL to emulation in about 10 minutes.
- **Ability to do HW/SW co-debug with software and true RTL debugging to help understand relationships between hardware and software in a single environment and find bugs more quickly.**

Although the FPGA solution may have been viewed as “good enough” by some, the benefits of emulation were clear. More bugs were found and fixed in a shorter time and the completed design achieved higher performance and higher quality than what would have been achievable without emulation. Cadence has extensive emulation experience, and the Palladium and Xtreme series products meet the challenges of verification.

![Figure 10: In-circuit emulation](image)