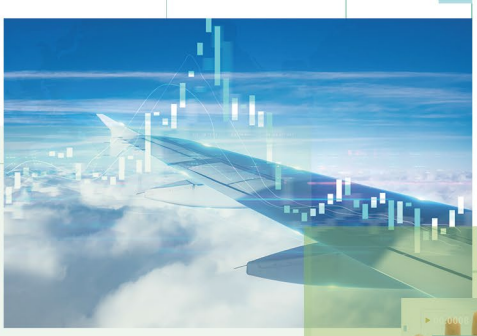
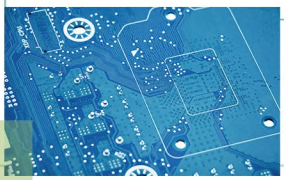




PROTOTYPICAL II

The Practice of FPGA-Based Prototyping for SoC Design

Daniel Nenni & Steve Walters

A SEMIWIKI PROJECT

Published by SemiWiki LLC Danville, CA

First printing: July 2021

Printed in the United States of America

Contents

Part I – Evolution of Design Verficiation Techniques

The Art of the "Start" 4

A Few Thousand Transistors 6

Microprocessors and ASICs 9

The Birth of Programmable Logic 11

Pre-Silicon Becomes a Thing 15

Positioning: The Battle for Your Mind 19

First Pentium Emulation 20

Enabling Exploration and Integration 25

Part II - FPGA Prototyping for Different Design Stages

Design Exploration 30

IP Development 35

Hardware Verification 45

System Validation 57

Software Development 64

Compatibility Testing 71

Part I – Evolution of Design Verification Techniques

The Art of the "Start"

The semiconductor industry revolves around the "start." Chip design starts lead to more EDA tool purchases, more wafer starts, and eventually to more product shipments. Product roadmaps develop to extend shipments by integrating new features, improving performance, reducing power, and reducing area – higher levels of functional integration and what is referred to as "improved PPA." Successful products lead to additional capital expenditures, stimulating more chip designs and more wafer starts. If all goes well, and there are many things that can go wrong between the MRD and the market, this cycle continues. And in keeping with good capitalist intentions, this frenetic cycle drives increased design complexity and design productivity to feed the global appetite for economic growth.

Chip designs have mutated from relatively simple to vastly complex and expensive, and the silicon technology to fabricate chips has advanced through rapid innovation from silicon feature sizes measured in tens of microns – to feature sizes measured in nanometers. Once visualized as ones and zeroes in a table, functions now must comprehend the execution of powerful operating systems, application software, massive amounts of data, and heretofore incomprehensible minuscule latencies.

Continued semiconductor industry growth depends on delivering ever more complex chip designs, co-verified with specialized system software – in less time with relatively fewer mistakes. New chip wafer fabs now cost billions of dollars, with production capacities in the 10's of thousands of wafers per month – in May of 2019, TSMC announced that it would build a new wafer fab in Arizona. The total project spending for the planned new 5-nm wafer fab, including capital expenditures, is expected to be approximately $12B from 2021 to 2029, and the fab is expected to have the capacity to produce 20,000 wafers per month. [1] One malevolent block of logic within a chip design can cause very

expensive wafers to become scrap. If a flaw manages to escape, only showing itself at a critical moment in the hands of a customer, it can set off a public relations storm calling into question a firm's hard-earned reputation as a chip supplier.

Chip design verification is like quality: it asymptotically approaches perfection but never quite achieves 100%. It may be expressed as a high percentage less than 100%, but close enough to 100%, to relegate fault escapes to the category of "outlier" – hopefully of minimal consequence. Only through real-world use in the hands of lots of customers will every combination of stimuli be applied to every chip pin, and every response be known. So, chip designers do their best to use the latest cocktail of verification techniques and tools, and EDA companies continually innovate new verification tools, design flows, and pre-verified silicon IP, in a valiant effort to achieve the elusive goal of achieving chip design verification perfection.

The stakes are very high today for advanced silicon nodes where mask sets can cost tens of millions of dollars, and delays in chip project schedules that slip new product roll-out schedules can cost millions of dollars more in marketing costs. With the stakes so high for large, sophisticated chips, no prudent leader would dare neglect investing in semiconductor process quality. Foundries such as GlobalFoundries, Intel, Powerchip, Samsung, SMIC, TSMC, UMC, and others have designed their entire businesses around producing high-quality silicon in volume at competitive costs for their customers.

So, chip design teams struggle to contain verification costs and adhere to schedules. The 2020 Wilson Report found that only about 32 percent of today's chip design projects can achieve first silicon success, and 68 percent of IC/ASIC projects were behind schedule. [2] A prevailing attitude is that the composite best efforts of skilled designers using advanced EDA design tools should result in a good outcome. Reusing

known-good blocks, from a previous design or from a reliable IP source, is a long-standing engineering best practice for reducing risk and speeding up the design cycle. Any team that has experienced a chip design "stop" or "delay" knows the agony of uncertainty and fear that accompanies these experiences. Many stories exist of an insidious error slipping through design verification undetected and putting a chip design, a job, and sometimes an entire company, at risk. The price of hardware and software verification escapes can dwarf all other product investments, and ultimately diminish a hard-earned industry leadership reputation.

Enter FPGA-based prototyping for chip design verification. A robust verification plan employs proven tests for IP blocks, and tests the fully integrated design running actual software (co-verification) – which is beyond the reach of software simulation tools alone. Hardware emulation tools are highly capable, and faster than software simulation, but highly expensive and often out of reach for many design teams. FPGA-based prototyping tools are scalable, cost-effective for almost any design, offer capable debug visibility, and are well suited to hardware-software co-verification.

In this book, we look at the history of FPGA-based prototyping and the leading providers – S2C, Synopsys, Cadence, and Mentor. Initially, we will look at how the need for co-verification evolved with chip complexity, where FPGAs got their start in verification, and why ASIC design benefits from prototyping technology.

A Few Thousand Transistors

One transistor came to life at Bell Labs in 1947. Solid-state electronics held great promise, with transistors rapidly improving and soon outperforming vacuum tubes in size, cost, performance, power

consumption, reliability, and footprint. However, there were still packaging limitations in circuit design, with metal cans, and circuit boards and wires, and discrete passive components such as resistors and capacitors. [3]

In 1958, Jack Kilby of TI demonstrated a simple phase-shift oscillator with one bipolar transistor and roughly hewn resistors and capacitors on one slice of germanium, with flying wire connections on the chip. By 1960, Fairchild teams led by Robert Noyce had a monolithic integrated four-transistor flip-flop running in silicon, a more stable and mass-producible material and process. [4], [5]

Standard small-scale integration (SSI) parts appeared in 1963, with Sylvania's SUHL family debuting as the first productized TTL family (Transistor-Transistor Logic). TI followed with the military-grade 5400 Series and the commercial-grade 7400 Series, setting off a parade of second-sourcing vendors. In rough terms, these SSI parts used tens of transistors providing a handful of logic gates. [6]

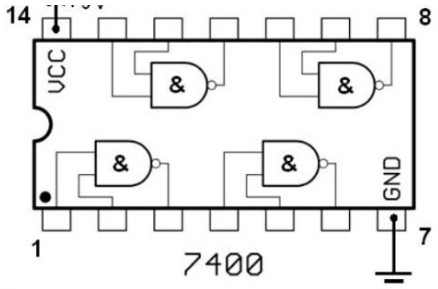


Figure 1: 7400 SSI Logic Device – Quad 2-Input NAND Gates

Medium-scale integration (MSI) first appeared with the 4-bit shift register – a part that Irwin Jacobs of Qualcomm fame proclaimed in a 1970 conference as "where it's at" for digital design. MSI parts with hundreds of transistors extended the productized logic families with a range of functions, but were still simple to use. Where SSI parts offered several individual gates in a single package with common power and

ground, MSI parts usually grouped gates into a single functional logic block operating on multiple bits of incoming data. Pin counts and package sizes remained small. SSI and MSI parts are the electronic equivalents of hand-chiseled statues. Producing a mask was labor-intensive, with layouts carefully planned and checked by engineers. Vendors heavily parameterized parts across variables of voltage, temperature, rise and fall time, propagation delay, and more. Each chip was a small block of IP, taken as golden, assembled into a system using wire wrapping or stitching for prototypes or short runs, and printed circuits for finished products in higher volumes. Everything about an SSI or MSI design was readily visible just by probing with an oscilloscope or logic analyzer at the package pins, and problems were usually somewhere in the wires in between.

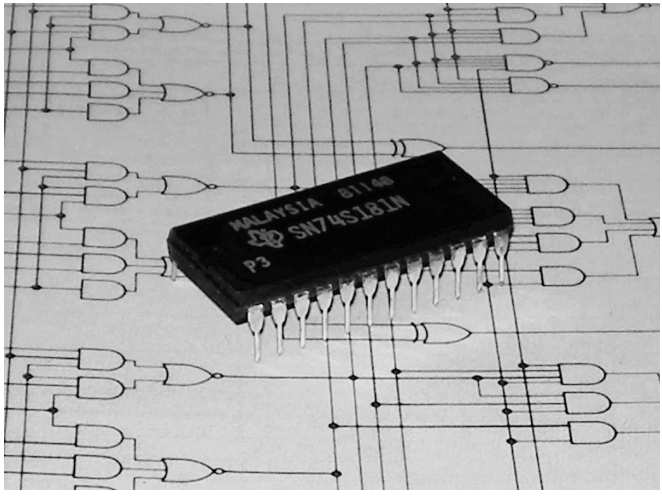


Figure 2: Texas Instruments SN74S181N 4-bit ALU with 63 logic gates

That changed drastically when large-scale integration (LSI) parts emerged. The early 1970s saw chips for digital watches, calculators, and the first integrated computer memories, each with a few thousand transistors. LSI parts were analogous to Mount Rushmore – carved from the monolith in labor-intensive steps. Parts were harder to verify post-layout, and more expensive to fabricate. Packaging changed as chips had

significantly more I/O pins. Second-sourcing became less common as vendors protected their high-value IP. Using LSI chips changed as well. The good news was more functions were integrated. The bad news was board-level test visibility declined, with designers having to trust the datasheet because the inner workings of a chip were mostly impenetrable. Chip errata become commonplace; instead of fixing the chip layout immediately, vendors spent energy on diagnosing issues and determining workarounds, waiting to gather enough fixes to justify a chip re-spin.

Microprocessors and ASICs

Entire "processors" were implemented by connecting LSI, MSI, and SSI chips on printed circuit boards. A prime example was a Linkabit design in 1971 for a Viterbi decoder – 360 TTL chips on 12 boards, in a single 4.5U rackmount enclosure replacing a couple of cabinets of earlier equipment. Assembly language programming took shape, with simple instruction sets. This was exactly the transformation Jacobs had been talking about, but his firm and many others were looking beyond to bigger chips that consolidated functions. [7]

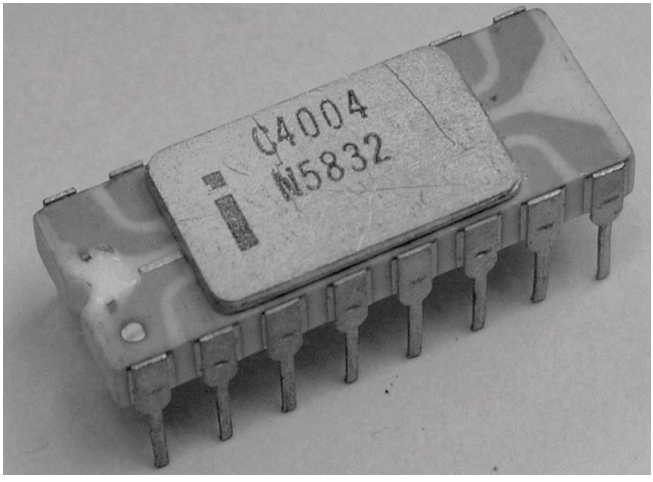


Figure 3: Intel 4004 microprocessor

Intel moved to the lead in LSI with offerings in DRAM, EPROM, and a new type of chip in November 1971: the microprocessor. Its first part sprang from a custom product for a Japanese calculator vendor. The 4004 4-bit microprocessor debuted under the MCS-4 banner, including RAM and ROM and a shift register tuned for the 4-bit multiplexed bus. With 2300 transistors fabricated in 10-micron silicon and running up to 740 MHz, the 4004 had 16 internal registers and offered 46 instructions. [8]

Feverish competition ensued as a slew of vendors created new 8-, 16-, and 32-bit microprocessor architectures during the late 1970s and early 1980s from chip companies including Intel, Zilog, Motorola, and Signetics. Even with lengthy schedules and meticulous design checking, very few of these complex chips worked the first time. Design and fab costs continued escalating as transistor counts moved into the tens of thousands and beyond. Most of these microprocessor vendors had large fabrication facilities, and proprietary design flows tuned to their engineering standards and fabrication process. A sea of changes was occurring in VLSI (Very Large-Scale Integration), with several technological advances fanning the flames of semiconductor company competition.

The first use of ASICs was as glue logic for improved integration of other discrete chips, or as companion chipsets to microprocessors, customized to a specific application. A growing roster of ASIC vendors eventually including AT&T, Fujitsu, IBM, LSI Logic, Plessey, Toshiba, TI, and VLSI Technology were working to harvest the economic benefits of the best abstracted design flow with chip design tools, IP libraries, and fab qualification. For the first time, design teams at a customer could create parts using a “standard cell library”, and get the design produced as a custom ASIC at moderate risk and reasonable lead times of a few months.

The average 32-bit microprocessor trended toward being bloated, with more transistors to execute complex instruction sets (CISC) with routine, and not-so-routine, operations and specialized addressing modes. Researchers focused on the flow of CISC instructions, deciding that instruction sets could be optimized for simplicity and performance, and came up with the idea of a Reduced Instruction Set Computer, or RISC. ASICs and RISC architectures then lead to new microprocessor companies, including MIPS Computer Systems, Sun Microsystems, and others producing new processor chip products to compete with Intel's CISC architecture processor chip products.

The Birth of Programmable Logic

Early programmable logic devices (PLDs) were emerging in the mid-1970s as an adaptation of early programmable read-only memory (PROM) technology. Read-only memories (ROMs) were a common staple of electronic system design – they were also commonly referred to as "non-volatile memory devices" (NVMs) because these memories would retain their contents upon the loss of power. ROMs would be programmed with information that might only be used at system startup and did not change during system operation, such as "boot ROM operations", which enabled systems to automatically restart after every power cycle by implementing a minimum set of startup operations that would automatically execute after each incidence of power loss when power was eventually restored. Early bipolar (CMOS semiconductor technology was still in its infancy at the time) programable read-only memory devices (PROMs) emerged with memory densities from 256-bits (32 words by 8 bits) to 4K-bits (512 words by 8 Bits). PROM devices would be manufactured with all of the memory content in the same "state" (ONE or ZERO) and the user would "program" the PROMs with an application-specific content by applying prescribed voltages to the device pins in a prescribed sequence affecting selected wires (sometimes

called “fuses”) that would change the logic state of selected memory bits. The bipolar PROM programming process is an irreversible, “one-time” process – once the PROM was programmed, it could only be reprogrammed by changing any previously unprogrammed bits, but the previously programmed bits could not be changed again. In some cases, users would organize the PROM into “banks” of memory (where a bank was selectable with a most-significant memory address bit), and if the memory image needed to be changed, a different memory bank would be programmed with the new memory image. Since PROM programming was a physical process involving a wide variety of PROM devices from various semiconductor companies with a wide variety of PROM programming procedures – another product market was born to commercialize PROM programming products – eventually lead by companies like Data I/O.

Imagine, if you will, the early bipolar PROM semiconductor technology required to effectively “melt” the semiconductor metal interconnects between transistors to permanently change the logical state of each selected memory bit – to produce devices that could be mass-produced, subsequently programmed in the field by users, and then be expected to operate reliably over time in production electronic systems. The affected metal layer interconnect needed to be “opened”, or removed, after the semiconductor manufacturing process, including packaging considerations that would tolerate the electrical currents necessary to only melt the selected interconnect, without any remaining metal fingers that might not render the electrical connection completely open, and no remnants of the violent microscopic physical event required to open the electrical connection would be “splashed” onto neighboring circuits and adversely affect their electrical integrity.

Why all this fuss about building a field programmable logic device? Well, I don’t believe that the pioneers of early programmable logic could possibly have imagined that the programmable logic market would

explode the way it has into today's FPGA market – almost a half-century later. At Signetics, a pioneer of programmable logic devices, management would need to be "sold" on the idea that existing Signetics' MSI and SSI logic device customers would rush to buy high volumes of these new, more highly integrated logic devices. At the time, if you were building almost any commercial digital system, you would buy boatloads of the MSI and SSI logic devices, mount them on printed circuit boards, and either hard-wire the interconnect with board-level metal traces, or use "wire-wrap" techniques to connect the socketed logic devices. It was not uncommon to spend long hours of logic system debug time with a wire-wrap gun, and wire-wrap wire removal tool, to bring-up new digital systems. The only options at the time for a higher density semiconductor implementation of a complex digital system were the newly emerging microprocessors that could implement the required logical functionality in software, or, if the expected end-product production volume was high enough to justify the cost, custom silicon ASICs.

Signetics and Intersil were two of the first semiconductor companies to offer programmable logic devices by adopting their PROM programming technology to logic devices. Signetics, who was a market leader in bipolar PROMs, as well as a leading supplier of MSI and SSI logic devices (7400 and 74S series), began offering bipolar programmable logic devices in a 28-pin plastic package adapted from their line of MSI and SSI logic devices.

Signetics called these devices fuse Programmable Logic Arrays, or FPLAs. FPLAs were simple at first: field programmable by the user, 16 inputs variables, 48 programmable product terms (or P-terms), and eight outputs functions (16 x 48 x 8). Signetics named this product the "PLS100", and claimed it was fully supported by "industry standard (JEDEC compatible) PLDCAD tools for designing the logic function to be implemented, including Signetics' SNAP, Data I/O Corporation's

ABEL, and Logical Devices Inc.'s CUPL design software packages." [9]

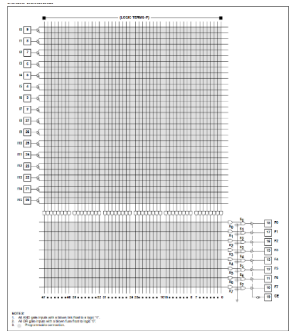


Figure 4: Signetics PLS100 Programmable Array Logic Diagram

Another breakthrough was near. Altera took an idea from the research halls of GE, combining the elements of EPROM memory (Erasable PROM) with CMOS floating logic gates, and added synthesis software in 1984. A logic design for the Altera EP300 could be created on a PC in a week or so using schematic capture, state machine, or logic table entries. Parts could be "burned" (programmed) and easily erased with ultraviolet light – and then reprogrammed as needed, in a matter of hours. Customers with conventional digital logic schematic entry skills had access to relatively high levels of customization with very low turnaround time.

Figure 5: Altera EP300 programmable logic device

A different technology appeared on November 1, 1985, with the thundering headline, "Xilinx Develops New Class of ASIC". [10] The XC2064 logic cell array was RAM-based, loading its configuration at boot time. Soon to be labeled by the media as a field programmable gate array or FPGA, these first parts featured 1200 gates, offering more scalability and higher performance. Logic could be simulated on a PC, and in-circuit emulation aided in functional verification.

Pre-Silicon Becomes a Thing

With programmable logic in its infancy, VLSI designs were still territory for ASICs. Even "moderate risk" using ASIC technology was still a significant risk. The SPARC I processor took four re-spins to get right. In contrast, the ARM1 processor at Acorn Computers powered up and ran on its first arrival from VLSI Technology in April 1985 – a minor miracle that shocked its creators and still stirs amazement.

From Calma and Applicon came the first EDA tools from pioneers Daisy, Mentor, and Valid which were adapted from Calma and Applicon circuit board design – to ASIC design tasks. [11] Rather than capturing a design and "tossing" it into silicon and hoping for good results, more emphasis was being placed on design verification to confirm correct operation prior to committing to silicon. EDA workstations were relatively fast, but the simulation of a VLSI design was still a tedious and slow process – requiring a great deal of skill to create a testbench that would provide a comprehensive set of stimuli and expected responses.

In many cases, ASIC simulation was cheaper than a failed piece of silicon, which meant more dollars and several more months waiting for a silicon fix. [12] Nonetheless, some designers resigned themselves to accept that it was not possible to "get it right" with one spin of silicon, and literally planned for the inevitable re-spin (or two) and set a goal to

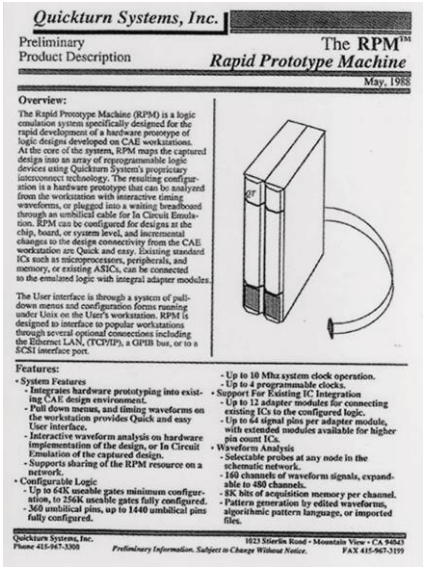
"minimize" the number of re-spins to 2 or 3 rather than plan to eliminate re-spins.

Major chip design innovation was happening at Intel. Thanks to the success in PC markets, its microprocessor families evolved rapidly. The design of the first mainstream PC processor, the 8088 released in 1979, involved painstaking human translation of logic gate symbols into transistors. For the 80286 debuting in 1982, an RTL (register transfer level) model drove high-level design and timing analysis, but manual translation into transistor structures was still necessary. The 80386 launched in 1985, was a 32-bit extension of the 80286 architecture requiring 275,000 transistors, [13] and saw a wider use of RTL synthesis and a move toward CMOS standard cells, with only several specific logic blocks which were hand-optimized.

If Intel was to continue its winning streak, it needed a breakthrough in design productivity. They knew that they would not be able to hire enough chip designers to keep up with the anticipated growth in processor design complexity. Development processes had to change to require fewer engineers and shorten the design cycle time for increasingly more complex parts. In 1986, Intel made a $250M investment for its next microprocessor design, including a proprietary system of EDA tools and practices. To enable fully automatic synthesis of layout from RTL, teams created iHDL, built logic synthesis tools from code developed at the University of California, Berkeley, and formalized and extended its standard cell libraries. The result was the 80486 in 1989, breaking the one-micron barrier with a staggering 1.18M transistors. [14]

Even with these investments in tools and processes, Intel knew that software-based simulation was too slow. RTL simulations were chewing up more than 80% of Intel's EDA computing resources, and the verification effort was growing non-linearly with processor size. Intel

was focused on reducing the verification effort and made it a priority to find a better way. A new verification solution would come from an unexpected source: a small startup company – Quickturn Design Systems. Quickturn was founded in May 1988 by Mike D'Amour and Steve Sample from Daisy Systems, and they envisioned a new type of verification platform targeting chip designers. Their first FPGA prototyping product was called the Rapid Prototype Machine ("RPM") using an array of Xilinx XC3090 FPGAs. Quickturn's early software could take an ASIC netlist of tens of thousands of gates, partition it into a large array of FPGAs, and model chip design functionality that could be run much faster than other verification methods available at the time. [15]



Quickturn Systems, Inc.

Preliminary
Product Description

The **RPM**™
Rapid Prototype Machine

May, 1988

Overview:

The Rapid Prototype Machine (RPM) is a logic emulation system specifically designed for the rapid development of a hardware prototype of logic designs developed on CAE workstations. At the core of the system, RPM maps the captured design into an array of reprogrammable logic devices using Quickturn System's proprietary interconnect technology. The resulting configuration is a hardware prototype that can be analyzed from the workstation with interactive timing waveforms, or plugged into a waiting breadboard through an umbilical cable for In Circuit Emulation. RPM can be configured for designs at the chip, board, or system level, and incremental changes to the design connectivity from the CAE workstation are Quick and easy. Existing standard ICs such as microprocessors, peripherals, and memory, or existing ASICs, can be connected to the emulated logic with integral adapter modules.

The User interface is through a system of pull-down menus and configuration forms running under Unix on the User's workstation. RPM is designed to interface to popular workstations through several optional connections including the Ethernet LAN, (TCP/IP), a GPIB bus, or to a SCSI interface port.

Features:

- System Features
  - Integraies hardware prototyping into existing CAE design environment.
  - Pull down menus, and timing waveforms on the workstation provides Quick and easy User interface.
  - Interactive waveform analysis on hardware implementation of the design, or In Circuit Emulation of the captured design.
  - Supports sharing of the RPM resource on a network.
- Configurable Logic
  - Up to 64K useable gates minimum configuration, to 256K usenble gates fully configured.
  - 360 umbilical pins, up to 1440 umbilical pins fully configured.
  - Up to 10 Mhz system clock operation.
  - Up to 4 programmable clocks.
- Support For Existing IC Integration
  - Up to 12 adapter modules for connecting existing ICs to the configured logic.
  - Up to 64 signal pins per adapter module, with extended modules available for higher pin count ICs.
- Waveform Analysis
  - Selectable probes at any node in the schematic network.
  - 160 channels of waveform signals, expandable to 480 channels.
  - 8K bits of acquisition memory per channel.
  - Pattern generation by edited waveforms, algorithmic pattern language, or imported files.

Quickturn Systems, Inc.
Phone 415-967-3300

Preliminary Information. Subject to Change Without Notice.

1023 Stierlin Road • Mountain View • CA 94043
FAX 415-967-3199

Figure 6: Quickturn Systems RPM datasheet

In the beginning, Quickturn was struggling with what to call their new technology – what we call "Emulation" today. When Quickturn first talked with potential customers about Emulation for ASIC design verification, it was a time when software simulation, hardware accelerated simulation, and FPGA prototyping were all commonly used for design verification in varying combinations – and hardware

Emulation was unknown and different from the other available verification approaches. Traditional software simulators used a "timing wheel" approach to model the propagation of signals through the design logic with each "tick" of the design clock. The propagation of signals through each and every logical element of the design would be evaluated for each clock cycle, and then the evaluation would be repeated for each subsequent "tick" of the design clock – like a "wheel" that completes one turn with every clock cycle using the propagated signals from the previous clock cycle to stimulate every logical element of the design on the next clock cycle.

Simulation accelerators were also timing wheel-based engines using specialized hardware, and were faster than software simulators running on servers, but accelerators were still orders of magnitude slower than running the design on an FPGA hardware implementation. At the time, FPGA prototyping was mostly relegated to small designs that could fit into a single FPGA, and was a primitive, highly manual process, not considered for large designs that required multiple FPGAs connected together into a single large "sea of gates". Further, because FPGA suppliers considered FPGA design tools to be an enabler to selling FPGAs chips, pricing for these tools was inherently depressed – the FPGA suppliers basically gave the tools away as a necessary selling cost for selling the FPGAs. FPGA suppliers were in the business of selling FPGAs, not design tools, and consequently, the economic incentive for EDA companies to invest in innovation was impaired.

In early discussions with potential customers, Quickturn customers would ask: "Do you mean simulation acceleration?" The answer would be, "no, it's much faster." Then the customer would ask: "Do you mean FPGA prototyping?" Again, the answer would be "no," because FPGA prototyping at the time referred to a highly manual process that lacked serious automation for implementing the prototype. What Quickturn was developing was different from simulation, simulation acceleration,

or the FPGA prototyping of the day, so Quickturn searched for a new "category name" that was not "simulation", and was not "FPGA prototyping". To clearly differentiate this new verification method, Quickturn would eventually choose the name "Emulation", and "position" a new "market category" in the minds of customers – something different yet similar to the already familiar verification methods of the day.

Positioning: The Battle for Your Mind

The Al Ries' and Jack Trout's book "Positioning: The Battle for Your Mind," [16] was an inspiration to Quickturn at the time, and they rallied behind several of the book's concepts to position Emulation in the hierarchy of existing verification tools:

> "'It's better to be first than it is to be better' is by far the most powerful positioning idea."
>
> "'If you can't be first in a category, then set up a new category you can be first in' is the second most powerful positioning idea."
>
> "The mind has no room for what's new and different unless it's related to the old."

The Ries and Trout positioning concepts were, in part, what led Quickturn to choose the name "Emulation". Quickturn's Emulation became the "first" in a new "chip design verification category", and Quickturn began positioning Emulation as it "compared to" simulation, and FPGA prototyping – faster than simulation, and easier to deploy than FPGA prototyping. Later, after Quickturn realized how difficult Emulation actually was to set up and use when compared to simulation, and after they had already established the Emulation category, they doubled down and coined the phrase: "Damned hard, but well worth the effort" – then they worked diligently to create compelling and plausible ROI scenarios to defend the cost and effort of deploying Emulation, with

an emphasis on lost revenue attributable to delayed time-to-market (TTM).

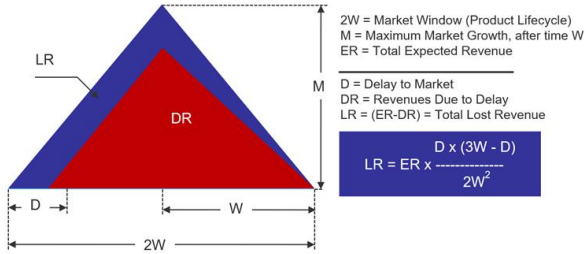


Figure 7: Lost Revenue Due to Delayed Time-To-Market

First Pentium Emulation

Getting back to the story about Intel's "discovery" of Quickturn and their new Emulation products – Intel was in the early stages of designing the Pentium P5 processor, and the P5 project managers were desperate to find new innovative ways to reduce the P5 verification effort. One of the engineers assigned to the P5 verification team in the late 1980's, Azam Barkatullah, took this challenge to heart and on a sunny California day he happened to be driving by Quickturn's headquarters in Mountain View, California, when he saw a Quickturn sign that caused him to stop his car and go inside. Quickturn was still a young EDA company struggling to establish itself in the EDA market. Azam met with Quickturn's management who explained how their new verification technology might be applied to the P5's daunting verification challenge and reduce the TTM. Armed with RPM Data Sheets and a vision for "In-Circuit Emulation" for more accurate system-level verification, Azam returned to Intel to report the discovery to his management. Intel quickly realized that if they could model the P5 in FPGA hardware, and plug a hardware Emulation of the P5 into a real PC prior to silicon, they could run real software on the Emulated P5 design – and they might be able to substantially reduce the time it would take to get through a

rigorous verification process faster and with fewer resources. Intel's Pentium CPU Validation Goals would eventually become:[17]

> Ensure backward compatibility with previous generations of processors.
>
> Run real-world operating systems and applications pre-silicon to validate the design.

Pentium project management arranged to meet with Quickturn management to negotiate a deal to acquire enough RPMs for P5 verification, but at the time they didn't know how many RPMs it would take. A fully configured RPM at the time was expected to support about 150,000 "equivalent ASIC gates", but the P5 had not been designed as an ASIC – it was a full custom design, and the concept of equivalent ASIC gates simply did not apply. Without any better options, Intel pushed forward and eventually a business deal was struck on a "project basis", where Intel would pay a base project fee with milestone-based bonuses, and Intel would get "as many RPMs as it took" to Emulated the P5. Under the terms of the deal, Intel would not need to purchase more RPMs than they needed at the start of the project, and they wouldn't need to negotiate the price of each additional RPM if they underestimated the number of RPMs they needed at the start of the project. Quickturn also knew that considerable marketing value would be derived from successfully Emulating the P5 processor running software while plugged into a real PC –prior to silicon.

A large part of Quickturn's early differentiation story was that an Emulated design could be connected directly to a real target-system, and this would become a very compelling justification for adding Emulation to a complex chip verification flow to reduce TTM. Quickturn would claim that the only sure way to verify that a chip design would work in a target-system prior to silicon was to connect the chip design to a real target system with what was called "In-Circuit Emulation" (ICE). ICE

became a compelling value proposition for pre-silicon chip Emulation. ICE was accomplished by connecting the Emulated chip design's I/O signals to one end of a flat multi-pin cable (or several cables), and the other end of the cable would be "plugged into" a real target system where the chip would eventually be plugged when the chip was available in silicon. Of course, the Emulation would run slower than the real silicon, but it was still orders of magnitude faster than simulation-based verification, and it would run fast enough to sustain target-system operation running real target-system software – prior to silicon tape out. ICE would also provide the verification benefit that the target system would, by its nature, subject the emulated chip design to real-world stimulus, which is the ultimate determiner of functionally correct chip operation.

Azam's four-man emulation team mounted a heroic effort to get the P5 Emulation up and running on Quickturn RPMs with the expectation that it would significantly reduce the P5 verification effort and accelerate TTM (also referred to by Intel as "time-to-money"). Azam's team worked 18-hour days converting the custom P5 design into synthesizable HDL, partitioning the design into RPM-sized blocks, assigning the blocks to RPMs, mapping the RPM interconnect to cables between the RPMs, physically connecting all of the interconnect cables between the RPMs, and connecting the cables for ICE to the target system.

Intel used a combination of Quickturn design tools and Intel design tools to develop a design flow for transforming the P5 HDL description into FPGA bit files that could be loaded into the RPMs with cables connecting the RPMs. The P5 design was created in Intel's proprietary HDL, which at the time was not synthesizable. For silicon production, Intel would hand-translate its proprietary HDL into a transistor layout, but Intel's proprietary HDL could not be used directly for RPM Emulation – it needed to first be manually translated into a synthesizable HDL. Azam's team then partitioned the P5 design into

blocks of logic that would fit into the RPMs, paying special attention to the required RPM interconnecting cables. The team knew the Emulated design implementation would be different from the P5 silicon design, but they worked diligently to build the best possible adaptation of the P5 design for Emulation that would be logically correct even if the timing had to be manipulated to meet the constraints of the RPMs. For example, some P5 embedded memories were implemented with "hardware adapter modules" that used actual memory chips outside the FPGAs, connected to the RPM backplane and then to the Emulated design inside of the RPMs. The P5 design was organized into blocks and the blocks were grouped together to create larger blocks with gate counts that were expected to fit into a single RPM – it took 7 RPMs for integer operations, 4 for caches, and 3 for floating point. Then the interconnect signals between the RPM design blocks were assigned to flat ribbon cables that would be connected between RPM's. It would eventually take 14 RPMs arranged in a U-shape, with half of the RPMs sitting on top of the lab tables, and the other half of the RPMs on the floor under the lab tables. Unwieldy as it appeared, the U-shaped RPM arrangement proved to be the optimal scheme for connecting the many cables that would be required between the 14 RPMs. Try to imagine 14 RPMs, stacked two high on the P5 lab tables arranged in a U-shape with a "pile" of dozens of flat ribbon cables crisscrossing the lab floor between the RPMs – later we'll reflect on this image to imagine an Emulation of Intel's next Pentium processor, the P6.

When the P5 design needed to be changed during the verification/debug process, and it was certain that it would need to be changed, the affected Emulation blocks would need to be recompiled (HDL to FPGA programming bit files) and, if the design change affected the RPM cables, the Emulation cable connections would need to be changed. That is, cables needed to be carefully disconnected from one RPM and reconnected to another RPM. Changing cables between RPMs made P5

Emulation changes slow and error-prone. Eventually, as the P5 design stabilized during the debug process, fewer changes were required to the Emulation. The cables would also limit the Emulation speed because of the signal propagation time across the cables between RPMs. The final P5 Emulation speed was eventually arrived at by "trial and error" – timing analysis was rudimentary at the time – and the Emulation clocks were simply turned down to about 300KHz until the functionality was correct.

It took Azam's team 4 to 5 months to get the P5 Emulation to a state where they could boot the DOS operating system and run software applications on a real target PC. In November of 1991, Albert Yu, an Intel VP at the time, was attending a forum for PC companies and software developers, and he "dialed into" (remember dial-up modems?) Azam's Emulation lab and ran a Lotus 123 spreadsheet application on the P5 Emulation from a remote terminal. The forum attendees were astonished that a P5 model was already working. It was said that Compaq Computer Corporation was planning to switch to a RISC-based PC at the time, but six months after the Albert Yu presentation at the November forum, Compaq scrubbed their plans to switch processors. The P5 Emulation was considered a success. The P5 Emulation team identified at least one critical design bug prior to tape-out, the first tape out of the P5 was functional (although not at speed), Albert was able to impress a crowd of product developers and potential customers with a demonstration of early P5 design maturity that may have saved at least one considerable business deal for Intel, the P5 Emulation was credited with shaving "a few months off the P5 production ramp" leading to a timely release of the Pentium processor in March of 1993. [18] And, Quickturn had a legend-worthy Emulation success story that they would use to encourage countless other prospective Emulation customers to invest in Emulation.

The P5 story was such a clear success that any reasonable observer

would have assumed that Intel would have gone on to repeat the P5 success on the P6 project. Well – Intel did engage Quickturn again to Emulate the P6, but the P6 story had a very different ending than the P5 story. Earlier in this section, the reader was asked to imagine 14 RPMs, stacked two high on the P5 lab tables arranged in a U-shape with a "pile" of dozens of flat ribbon cables crisscrossing the lab floor between the RPMs. When Intel moved to Emulate the P6 design, they realized that the Emulation technology of the day was not scaling at the same pace as their processor complexity. The P6 design was about four times the gate complexity of the P5 design. The P6 Emulation required more than 40 RPMs, and dozens more interconnect cables between the RPMs! The P6 Emulation build took much longer, design changes were much harder, and it was not credited with the pre-silicon discovery of any critical bugs. Intel discontinued Emulating processor designs after the P6.

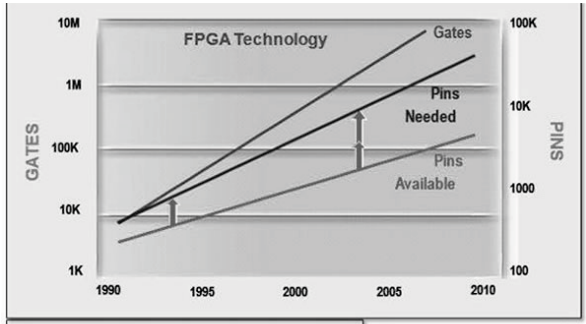
Enabling Exploration and Integration

The Quickturn RPM, implemented with standard Xilinx FPGAs, was the first commercial hardware emulator and became an essential ASICs prototyping tool for well-funded companies. Following Quickturn's success with Emulation, more advanced hardware emulators and FPGA-based prototyping platforms were developed – and they took divergent paths for different use cases.

Hardware emulators evolved to be highly automated, and only affordable for large chip projects and broader application for multi-user, multi-design projects. A user need not know the details of the logic implementation, or how internal design interconnects are organized on the hardware. A netlist for an ASIC is loaded by the Emulation software, chopped into arbitrary partitions, and the partitions are spread out across the hardware – implemented with tens or hundreds of devices. These partitions are subject to a relationship known as Rent's Rule,

describing an empirically determined ratio of logic gates to interconnect pins. As general purpose FPGA logic capacities increased, FPGA pin-count growth did not keep up and retain interconnect limitations got worse for arbitrary partitions of a design, requiring even more FPGAs to accommodate large netlists. Eventually, emulator providers moved from FPGAs to ASIC-based designs. The price of tossing more hardware at the problem is steep, however – today's high-performance hardware emulators can cost over $1M.

FPGA prototypes are more design specific, and often configured and tuned for a specific project. Assuming adequate logic capacity and interconnect pins, a design can be synthesized for a single FPGA device, or perhaps partitioned across a handful of devices with optimized interconnect between the devices. Rent's Rule becomes more manageable for a design of moderate size. This is the basic premise of FPGA-based prototyping, which gives the appearance of becoming more and more attractive as FPGA logic capacities improve. [19]



Year	FPGA Family	Max. Capacity Gate Equivalent	Max I/Os	Capacity per I/O
2003	Virtex-II Pro	1,230,000	1040	1,183
2004	Virtex-4	2,490,000	960	2,594
2006	Virtex-5	3,320,000	1200	2,767
2009	Virtex-6	9,105,000	1200	7,588
2011	Virtex-7	14,000,000	1200	11,667

Figure 8: FPGA gates versus pin count, courtesy Cadence

What really makes the case for FPGA-based prototyping is not a change

in FPGAs, however, but changes in system design practices and objectives. The type of design starts typically in the industry evolved dramatically, looking less often like an enormous Intel microprocessor. System-on-chips, microcontrollers, application-specific standard products (ASSPs), and other designs take advantage of a growing field of IP for customized implementations.

Reuse and integration are now paramount. Stand-alone verification of individual IP blocks is cost-effective using FPGA-based prototyping. Third-party IP, existing internally designed IP blocks, and new internal development can then be combined, with partitioning and test artifacts reused to aid in the process.

Design exploration is feasible, especially for software teams that can afford to provide FPGA-based prototyping platforms to lots of developers. What-if scenarios run at the IP-block level can explore software tradeoffs or minor hardware architectural tradeoffs, not just functional fixes. These results can be rolled up quickly to the full-up design, perhaps resulting in a critical product pre-silicon enhancement.

More FPGA-based prototyping platforms are integrating actual I/O hardware, usually with a mezzanine-based approach, instead of emulating I/O with a rate-adapter of some type. This is an important factor for complex interface and protocol verification. It can also be a deciding factor in safety-critical system evaluation, where validation using actual hardware is essential.

At the high-end, FPGA-based prototyping is scaling up. Platform-aware synthesis is improving partitioning across multiple FPGAs, allowing larger ASIC designs to be tackled. Cloud-based technology is connecting platforms and developers via the internet. Debug visibility is increasing, with approaches including deep-trace capture and automatic probe insertion. Integration with host-based simulation and graphical analysis

tools is also improving steadily.

The inescapable conclusion is, if a chip project is to “start”, it had better finish with robust silicon quickly. New applications, particularly the Internet of Things, may reverse a trend of declining ASIC design starts over the last decade. Design starts are likely to be smaller and more frequent, with highly specialized parts targeting niche markets. Advanced requirements in power management, wireless connectivity, and security are calling for more intense verification efforts.

FPGA-based prototyping, as we shall see shortly, is rising to these challenges for a new era of chip design.

Part II - FPGA Prototyping for Different Design Stages

Design Exploration

It is common wisdom today that early discovery of design problems during product development reduces the cost to fix the problems – especially with complex SoCs. It is also common wisdom that the cost to fix problems increases by 10x or more after silicon implementation and increases another 10x when the product gets into the hands of customers. These cost factor increases are most certainly understated for high-volume or high-priced products. The challenge of early problem discovery is amplified by rapidly increasing SoC design complexity and cost, high software content, and the declining TTM tolerated by competitive markets. Consequently, the pressure on electronic system designers to get it right early has never been higher – and can only get more intense. These considerations led to the definition of the term *Electronic System Level* design, or *ESL Design,* by Gartner Dataquest in 2001. [20] The term was defined as "the utilization of appropriate abstractions in order to increase comprehension about a system, and to enhance the probability of a successful implementation of functionality in a cost-effective manner." [21]

SoC-based designers are highly incentivized to model key system capabilities to verify key system parameter targets such as performance, power, silicon area, and functionality – early in the development process and at a high level of abstraction. These design abstractions are implementation-independent, consider system-level dependencies, can focus on isolated critical aspects of the design, are easy to refine (low effort), and the added verification complexities of implementation details are deferred until later in the development process. Many of the key system parameters are locked in by early architecture decisions and cannot be changed significantly after a certain point in the implementation process. As the design matures, additional levels of implementation detail are added to the process, solidifying parts of the design, and increasing the “weight” of the design (effort needed to

change) as it converges (hopefully) to the final product. During this design maturation process, critical sections of the design may be modeled using familiar design verification tools that would not normally be expected to come into play until later in the development process – such as FPGA prototyping. Algorithm performance, video imaging QOR, and network throughput are examples of where FPGA prototype modeling is finding increasing use in ESL.

Thus, the importance of design exploration during early product definition is considered by some to be the most critical stage of any development project – in which optimizations of the architecture are the least costly to realize in time and effort. *ESL Design* allows system architects to play "what-if" games with system partitioning and quickly evaluate different implementation alternatives – which parts should be implemented in hardware and which parts should be implemented in software. According to a 2018 article in Semiconductor Engineering;[22] "The architecture space was, is, and always will be, a relatively small number of users that use every tool in their arsenal, with Excel being probably the most used." The article goes on to assert, "The classic architecture analysis dilemma remains an issue: decisions must be made as early as possible to be effective; and to make effective architecture decisions, architects would like the accuracy of models that are only available once the implementation is decided." The article continues; "As a result, there is a clear bifurcation in the architecture space with very abstract models that are used pre-implementation on the one hand (using languages like The Mathworks M or graphical definitions like in National Instruments LabView) and cycle-accurate representations in RTL or SystemC on the other."

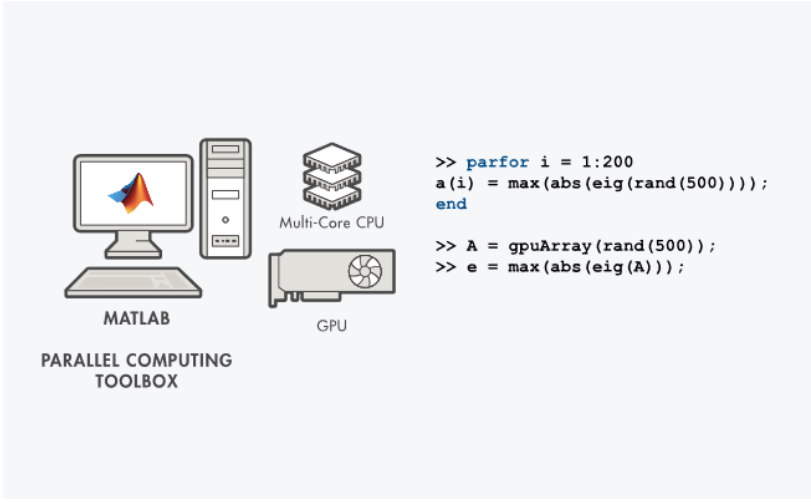


Figure 9: MatLab Parallel Computing Toolbox

So it is that classic hardware verification tools such as FPGA prototyping, together with transaction-level testing, are increasingly finding their way into the Design Exploration space for architecture optimization decisions. Again, from the Semiconductor Engineering article about ESL Design: "The products in this space simply bridge the classic verification space into the system level—one could argue by 'brute force'. You want to boot an operating system like Android or Linux, and you need the hardware implementation detail, hence cannot use abstraction? Here, use my emulator or FPGA-based prototype that runs in the MHz or 10's of MHz range, respectively, compared to the Hz or low KHz range in host-based RTL simulation that would take weeks to boot the OS. This area is somewhat of a gray space because the primary use of emulation is hardware verification, and the primary use of prototyping is software development—but the lines are blurry as emulation extends into software, especially with virtual-platform-emulation hybrids, and prototyping extends into hardware verification for regressions once the RTL gets more stable."

To facilitate early *Design Exploration*, ProtoBridge from S2C provides a high bandwidth data channel between software models running on a

host PC and the FPGA prototyping hardware. ProtoBridge consists of C-API for users, software drivers for the host PC, PCIe-based connectivity hardware between the host PC and the Prodigy Logic System, and a PCIe-to-AXI bridge to interface with the user design blocks.

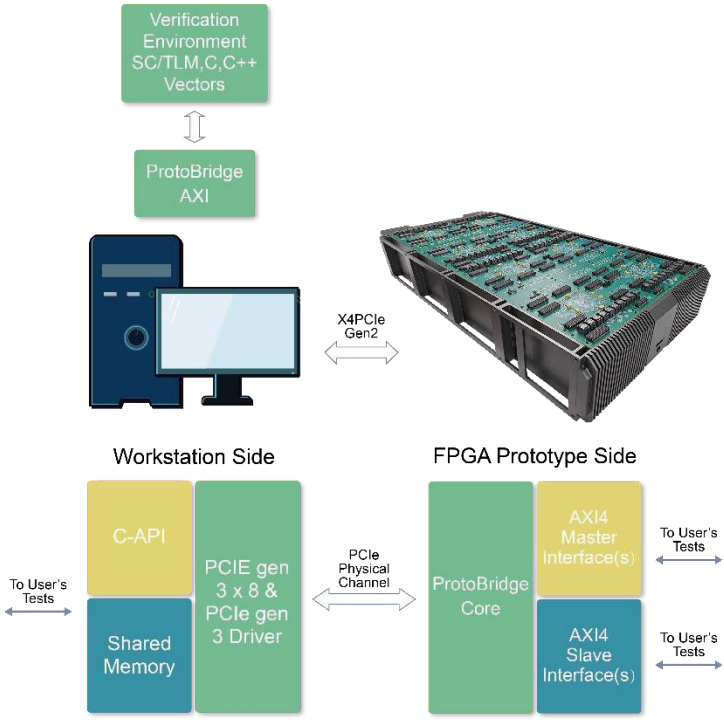


Figure 10: S2C ProtoBridge solution

In August of 2020, S2C announced a collaboration with Mirabilis Design to deliver a hybrid SoC architecture exploration solution that reuses available RTL-based blocks to accelerate model construction and complex simulations. Mirabilis Design's VisualSim interfaces with S2C's FPGA Prototyping solution, Prodigy Logic System, to model a functional block of the design in which the FPGA prototype acts as a sub-model and provides accurate simulation responses for architecture exploration. The collaboration enables the RTL behavior modeled through FPGA prototyping, to be easily integrated into an ESL model to create a virtual

platform. The model can be simulated to gather metrics on response times, throughput, power consumption, and correctness of data values.

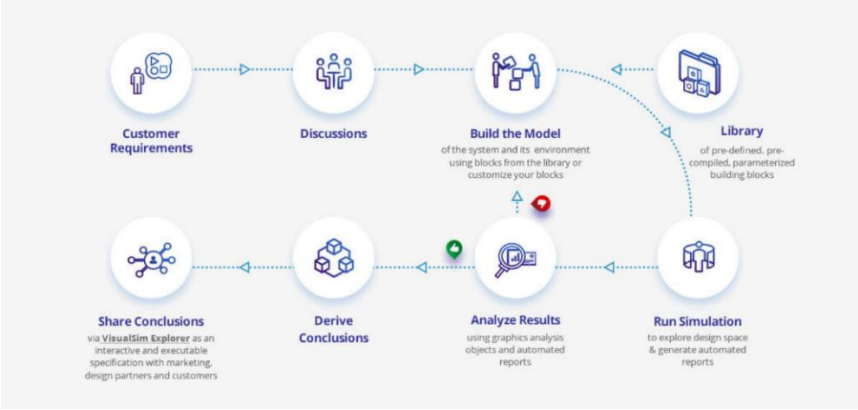


Figure 11: VisualSim Modeling, Simulation, Exploration and Collaboration

IP Development

Brief History

Semiconductor Intellectual Property (IP) may be defined as "a reusable unit of logic, cell, or integrated circuit layout design that is the intellectual property of one party. IP cores can be licensed to another party or owned and used by a single party." [23] While IP licensing became a common practice in the 1990's, the development and the use of IP by companies developing ASICs and SoCs for its own internal use, or as a paid service to other companies, or by semiconductor foundries to enable customer wafer production, has been around much longer.

The conceptual benefits from IP reuse were identified early by a few prescient technologists at large electronic companies (HP, Sun Microsystems, etc.) well before the commercial IP market enjoyed the vibrance that it does today. Internal company standardization of certain IP blocks was seen as an opportunity for improved ASIC development efficiency across multiple internal ASIC projects developing its own versions of essentially the same functionality – at the same time with little or no cooperation between the projects. As an added benefit, standardization and reuse would assure compatibility between all the company's different implementations of the same IP functionality (incompatibility between different internal development efforts of the same IP functionality in the same company actually occurred!). As an interesting historical note, and as is often the case when large companies try to standardize design practices, there were inefficiencies associated with early IP standardization efforts which tried to impose an IP reuse *standard interface* on all internally developed IP (to assure easy reuse by all internal projects) before the IP was allowed to be added to the company's IP library. It was a well-intended and conceptually valid constraint, but the internal IP user community resisted adoption because of the area and performance penalties associated with the

company-imposed IP reuse *standard interface* itself.

One EE Times article attempted to simplify the essence of Semiconductor IP with a few rhetorical questions and answers:

> "Question: What is the central issue facing the system-on-chip chip design community? Answer: Design costs.
>
> Q: How to cut costs? A: Design reuse.
>
> Q: What is the challenge of design reuse? A: Intellectual-property verification." [24]

We will return to the topic of IP verification later. However, this nod to IP verification reflects the criticality of well-verified IP to unlocking the potential value that Semiconductor IP can bring to SoC developers.

IP Market

According to a report by MarketsandMarkets, the global Semiconductor IP market is estimated to be $5.6 billion in 2020 and projected to reach $7.3 billion by 2025. [25] The report cited the "Key factors fueling the growth of this market include the advancement in multicore technology for the consumer electronics industry, increasing demand for modern SoC (system on chip) designs, mitigation of the continuously rising chip design cost and expenditure, growing adoption of connected devices for daily use, increasing demand for electronics from healthcare industry due to COVID-19, and increasing demand for teleconference instruments amid the COVID-19 pandemic."

Another report by Verified Market Research projects similar numbers for the Semiconductor IP Market size, pegged at $5.3 billion in 2019, growing to $7.4 billion by 2027. "Various factors that are driving the growth of the Semiconductor IP Market are growing production of mobile devices and use of electronic devices such as smartphones and tablets. Also, risen (sic) demand of modern system on chip (SoC) design

and reduction in design and manufacturing cost is boosting demand for Semiconductor IP Market." [26]

Figure 12: Global Semiconductor IP Market, 2020-2027

A third report published by Polaris Market Research is a little more aggressive and anticipates the Semiconductor IP market will be $9.3 billion by 2026. [27] "Rising demand for modern System on Chip (SoC) design, and growing need to reduce manufacturing and design cost boost the Semiconductor Intellectual Property market growth. Advancement in multicore technology for consumer electronics further supports the growth of the Semiconductor Intellectual Property market." [28]

Semiconductor IP is generally categorized by functionality: processor (ARM, RISC-V, Arc, Tensilica, etc.), foundation IP (memory, standard cells, etc.), on-chip interconnect IP (AMBA, NoC, etc.), standards-based interface IP (PCIe, USB, DDR, HDMI, MIPI, etc.), video CODEC IP, and analog and mixed-signal IP (PLL, ADC, DAC, PHY, etc.).

Today's commercial IP market is dominated by a few large EDA companies, which offer a rich selection of pre-tested, off-the-shelf IP. According to a June 2019 *Design & Reuse* article, the top 6 Semiconductor IP suppliers were expected to dominate the commercial IP market in 2019 (by revenue); Arm Holdings (40.8%), Synopsys.

(18.2%), Cadence (5.9%), SST/Microchip (2.9%), Imagination Technologies (2.6%), and Ceva (2.2%) – with these 6 IP suppliers expected to account for over 70% of the 2019 commercial IP market. [29]

Rank	Company	2018	2019	Growth	2019 Share
1	ARM (Softbank)	1,610.0	1,608.0	-0.1%	40.8%
2	Synopsys	629.8	716.9	13.8%	18.2%
3	Cadence	188.7	232.0	22.9%	5.9%
4	SST	104.8	115.0	9.7%	2.9%
5	Imagination Technologies	124.6	101.1	-18.9%	2.6%
6	Ceva	77.9	87.2	11.9%	2.2%

Figure 13: Semiconductor IP Suppliers by Revenue, Worldwide, 2018 and 2019 (Millions of Dollars)

While the revenue figures from these commercial IP supplier reflect the level of investment in commercial IP by SoC developers, they do not comprehend the investment by large fabless semiconductor companies that are developing their own IP for internal use on their own SoCs (e.g. Intel, Broadcom, Marvell, Qualcomm, etc.). Combining the commercial IP market with what is surely a huge internal investment in IP by large fabless semiconductor companies, it is abundantly clear that Semiconductor IP has become critical to the timely delivery of competitive SoCs – driven by the need to mitigate rapidly rising SoC development costs and complexities by leveraging large blocks of proven IP to reduce TTM, and focus SoC development efforts on core competencies for product differentiation.

Soft and Hard IP

Soft IPs are delivered as synthesizable RTL models or synthesized gate-level netlists. The IP supplier will usually include some verification aids with the Soft IP to assist the user with verifying the functional behavior of the IP, and sample scripts to assist the user in synthesizing the Soft IP into a physical implementation – a process called "*IP hardening*". Generally, digital logic functionality is delivered as Soft IP (RISC-V processor cores, PCIe controllers, DDR controllers, etc.). A notable

exception is "*foundation IP*", a category of digital functionality which includes memories and standard cell libraries, and this soft IP comes with *soft views* (abstract representations of memories and gates), and *hardened* views (more on this later) of the functionality. Most digital SoCs are composed primarily of memories and standard cell libraries, so this category of Soft IP is critical to the fabrication of any SoC and is fundamental to the verification of any Soft IP.

Prior to *hardening*, the Soft IP will be integrated into the complete SoC by the IP user, eventually at the gate-level, and the SoC will be rigorously verified for correct functionality, performance, and power consumption. SoC verification at the gate-level for performance, power, and area can produce highly accurate results because the gates that are used for *hardening* will have been derived from a foundry-specific IP library (*foundation IP*) that has been characterized for that specific foundry, a foundry-specific process node (20nm, 16nm, etc.), and a foundry-specific process variant (high-performance, low power, etc.). Some IP only has internal SoC connections (DMA controllers, PICs, AMBA, etc.) and must only be verified for correct internal SoC functionality. Other IPs having internal SoC connections and connections to the system outside the SoC require system-level verification.

The Soft IP gets hardened together with the rest of the SoC when the SoC is synthesized by the IP user into a physical silicon implementation – meaning that the gates (and memories, etc.) are converted to physical silicon transistors (also called "polygons" for the shape of the layered elements that comprise each transistor in silicon), and the interconnect wires that connect the gates, memories, etc. After the *hardened* SoC is verified, it undergoes additional physical verification for compliance with the foundry's process design rules. At the end of the *hardening* process, the SoC design is converted to another format (GDSII) that is used to produce "masks" for manufacturing the SoC in silicon, and a

mask set (40 to 50 layers at 28nm, while 5nm could have 100 layers [30]) for an advanced silicon process will cost millions of dollars for each mask set.

Another advantage of Soft IP is that it can be *targeted* by the IP user for any foundry processes (depending on available *foundation IP*) by synthesizing the SoC with foundry-specific *foundation IP* and going through the *hardening* process. Some large electronic companies have developed "*portable libraries*" that enable the same design to be fabricated by multiple foundries to assure continuous production supply in times of high silicon demand (also referred to as "multiple foundry sourcing"). The *portable libraries* approach has encountered some isolated glitches – occasionally, subtle differences would be discovered between *foundation IP* from different foundries (e.g., drive strengths) that would delay, or outright prevent, *targeting* the design for a specific multiple foundry source alternative.

Hard IP, unlike Soft IP, will have already been *hardened* by the IP supplier before it is delivered to the IP user. Generally, analog and mixed-signal functionality are delivered as Hard IP (SERDES, PCIe PHYs, DDR PHYs, PLLs, ADCs/DACs, etc.). Hard IP has been optimized by the IP supplier for performance, power, or area and will have been characterized over multiple "*process corners*", for the targeted foundry process. According to João Geada, chief technologist at ANSYS, from a Semiconductor Engineering article, "The foundry doesn't produce exact copies of transistors or chips. It's not an exact process. There's some random variability. To a certain extent, that's kind of hard to deal with from an engineering point of view. *Process corners* are, in a way, an attempt to put the bound on what comes out of the foundry – what's the fastest something can happen, what's the slowest, what's the worst power, what's the least power? There are multiple dimensions that you need to take into account, but you're trying to basically put a box around what will come out [of] the foundry; the corners are the edges of that

box." The article asserts that "The number of corners that need to be checked is exploding at 7nm and below, fueled by everything from temperature and voltage to changes in metal." [31]

Unlike Soft IP, Hard IP cannot be targeted by the IP user for a different foundry process. Some large consumers of silicon with close foundry relationships will go to great lengths to get the earliest possible start with the newest foundry process. They do this by designing with the foundry's pre-production release silicon characterization data (sometimes referred to as "guess numbers"), schedule their final SoC production tape-out to coincide with the later release of the final production silicon characterization data, have the IP supplier re-characterize the IP with the final characterization data ASAP, and re-run final physical verification on the SoC design just before tape-out. Usually, the differences between the "guess numbers" and the final production characterization data are inconsequential – so everything goes as planned – usually.

IP Verification

So, what's all the fuss about IP verification? Well, it's fundamental – it's foundational – it's critical to the realization of the two primary benefits of Semiconductor IP:

- the mitigation of soaring SoC development costs in the face of rapidly increasing design complexities, and
- the reduction of TTM in highly competitive markets.

Although much of the IP verification burden will be borne by the IP supplier before the IP ever reaches the IP user, the ultimate authority of correct IP operation is the SoC user – the IP must be verified stand-alone, when integrated into an SoC, and when the SoC fabricated in silicon and operated in the end-product. So, IP verification is performed by the IP supplier and the SoC developer in cooperation with the end-

product supplier. In some cases, the SoC developer and the end-product supplier are the same company (Apple, Cisco, etc.). In other cases, the SoC developer does not make the end-product and must work closely with the end-product supplier to complete SoC verification (Intel, AMD, Marvell, Broadcom, etc.).

The available EDA tools used for IP verification today support a wide range of IP and SoC abstractions during IP development, including ESL modeling, software simulators, stimulus generators, verification IP (VIP), formal verification, emulation, and FPGA prototyping. [32] The EDA tools at each stage of IP development are under tremendous pressure today to enable SoC developer's ability to:

- run more cycles of operation, and
- subject the SoC design to end-product like operating conditions.

FPGA prototyping is emerging as a critical and cost-effective method to achieve both. S2C has specialized in FPGA prototyping solutions for almost two decades and has evolved its complete verification platforms to scale from semiconductor IP and small SoC verification, to billion gate SoC verification platforms.

S2C offers a range of FGPA hardware platforms supporting Single-FPGA, Dual- and Quad-FPGAs with its Logic Systems – to more than a hundred FPGAs with its Logic Matrix.

Figure 14: Side View of S2C's Quad Prodigy Logic System (4 FPGAs)

S2C's Prodigy Prototyping Solutions can be configured to model a stand-alone IP block, an IP block together with associated VIP, a complete SoC containing the IP blocks that can be operated at hardware speeds. S2C's rich library of Daughter Cards is "snap-together" attachments to the Logic Systems to quickly implement system interfaces so the user can model in-system operations.

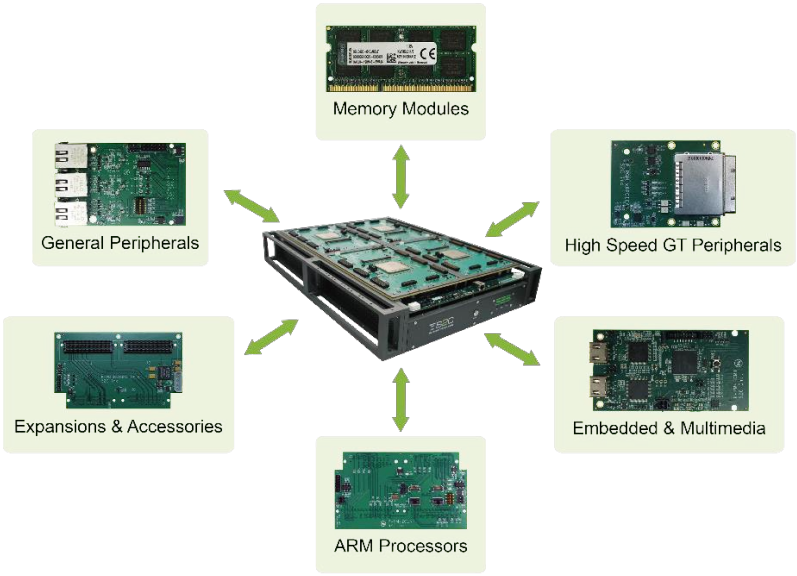


Figure 15: Prodigy Daughter Card Interface IPs

In addition, ProtoBridge, described earlier in this book, is also very handy in linking the FPGA prototyping platform to software platforms. These software platforms can be cycle-level RTL simulators or transaction-level virtual prototyping platforms. When in-depth RTL design verification is required, users connect the RTL simulator with the FPGA prototyping platform through ProtoBridge to accelerate cycle-level simulation. When RTL design has been verified and is ready for software development, users can then combine the virtual prototyping platform and FPGA prototyping platform for software development. Moreover, together with the aid of daughter card interface IP, which allows the RTL design in the FPGA to connect with real-world devices,

the design team can complete the IP design and verification tasks from RTL level to system level on a single FPGA prototyping platform.

Hardware Verification

Introduction

Some say *hardware verification* is the process of verifying that a given hardware design correctly implements the specification. [33] Others say that *hardware verification* is verifying that a hardware design operates as intended (in the system). As SoC-based system products have become more silicon and software-intensive, and as competitive markets continue to drive down TTM in the face of growing complexity and cost, SoC developers are being forced to incorporate *hardware verification/validation* methods that consider the bigger picture – "Did I design the right product"? [34]

In practice, *hardware verification/validation* means different things to different people. If you are developing an SoC for a system company, it means verifying that your own company's systems operate as expected by your company's customers. If you are developing an SoC for a chip company, it means verifying that systems produced by a variety of system companies using the SoC (your customers) operate as expected at their respective end-customers – and, it may not be known by the SoC developer, how all systems companies will connect and use the SoC, a very different verification proposition than what is faced by a system company developing SoCs that will only be used in its own systems. To reduce the risk of silicon failure, SoC developers commonly work closely with select system developer partners to know what it means for the SoC to operate as expected by the end-user. SoC developers at leading chip companies will go to extraordinary lengths to provide early prototype platforms to their select system developer partners to assure correct system operation during the early stages of the SoC development process – sometimes by providing hardware models implemented with FPGA prototypes. So it goes, the SoC developer focuses on designing the product right, then co-works with select system developer partners to

assure that they are designing the right product – and together they agree on what is expected from *hardware verification*.

Because so much of electronic system functionality today is being implemented in silicon, and the total cost of silicon failure continues to escalate, SoC developers and EDA suppliers strive to optimize tools and methodologies to collapse the SoC development process to enable SoC developers to verify working silicon designs in the system – before silicon is available. This thinking was promoted by a decades-old vision for pre-silicon in-system verification that was popularized by early hardware emulation called in-circuit emulation, or "ICE".

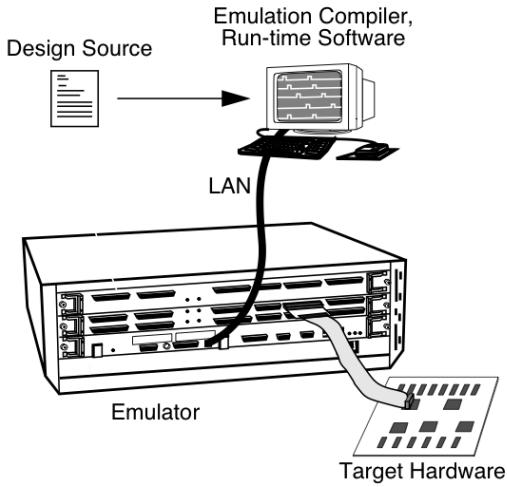


Figure 16: Early In-Circuit Emulation [35]

Lost In Translation

Nevertheless, we are getting a little ahead of ourselves. Somewhere in the hallowed halls of great system companies lurk the uniquely prescient minds that conceive blockbuster electronic product ideas. Products that will sell hundreds of millions of copies (e.g., Apple) or produce billions of dollars of revenue with fewer copies (e.g., Tesla). [36] Steve Jobs was quoted as saying, "People don't know what they want until you show it to them. That's why I never rely on market research. Our task is to read

things that are not yet on the page." [37]

Whether derived from market research, or the Steve Jobs approach to new product development, new product ideas must cross a chasm from vision to reality – and there will be layers of specification translations between the two. Architecture Specifications, Marketing Requirements Documents ("MRDs), Product Requirements Documents ("PRDs"), C-code models, HDL, etc. It should come as no surprise that some things will inevitably get lost in translation. This malady applies to clean-sheet new products, as well as evolutionary products developed as incremental derivatives of existing products. Phil Kaufman (CEO of Quickturn Systems in the 1980s) would say that a product's User Manual should be written first – and all other requirements documents should be derived from the User Manual. This strategy has the goal of first establishing how the product should work from the user's point of view, and then the product requirements would be elaborated in subsequent hardware, software, and system requirements documents to achieve the goal.

Given all the specification translation complexity, TTM pressure, and escalating SoC cost, it's no wonder that product developers, SoC developers, and EDA companies continue to push for better ways to close the product development loop from vision to reality earlier in the product development cycle. It is the high-pressure context of today's *hardware verification*.

Unknown Unknowns

SoC developers are handed a "script" in this grand scheme of product development, usually in some form of a PRD. A PRD is also provided to system and software developers and silicon bring-up engineers – and a symphony of coordinated efforts is begun by the team of stakeholders to produce a working product. SoC developers are asked to produce working silicon and cooperate with the other stakeholders along the path to committing the SoC to silicon. So, today's SoC developers make

every effort to comprehend the scope of how to test the SoC to confirm that they will produce the right product.

At some point in the SoC development process, the design is translated from the PRD into an HDL interpretation of the functionality that will eventually be used to produce the silicon – then they set about to prove (verify) that their HDL describes the product right, and in parallel, they must also plan to assure that they are designing the right product. Starting with the known knowns, SoC developers must eventually consider the known unknowns, and unknown unknowns – terms popularized by Donald Rumsfeld (yes that Donald Rumsfeld!) in a US Defense Department news briefing in 2002; [38]

- There are known knowns - these are things we know we know.
- We also know there are known unknown - that is to say we know there are some things we do not yet know.
- But there are also unknown unknowns - it is this latter category that tends to be the difficult ones.

"Rumsfeld's statement brought fame and public attention to the concepts of known knowns, known unknowns, and unknown unknowns, but national security and intelligence professionals have long used an analysis technique referred to as the Johari Window. The idea of unknown unknowns was created in 1955 by two American psychologists, Joseph Luft (1916–2014) and Harrington Ingham (1916–1995), in their development of the Johari window." [39] The Johari Window concepts may also be applied to hardware and SoC verification, and SoC developers knowingly or unknowingly apply these concepts to hardware and SoC verification. Known unknowns in SoC verification refers to "risks you are aware of." Moreover, "unknown unknowns are risks that come from situations that are so unexpected that they would not be considered". In the context of today's hardware and SoC verification complexities, and considering the high portion of the total SoC

development effort that is devoted to verification, it is clear why designers are constantly seeking better ways to include more real-life end-system operation of SoC designs as early as possible in the development process. Formal verification and randomized testing are approaches to uncovering known unknowns and unknown unknowns. These approaches advance the SoC verification process but still leave areas of uncertainty in a domain of virtually infinite possible combinations.

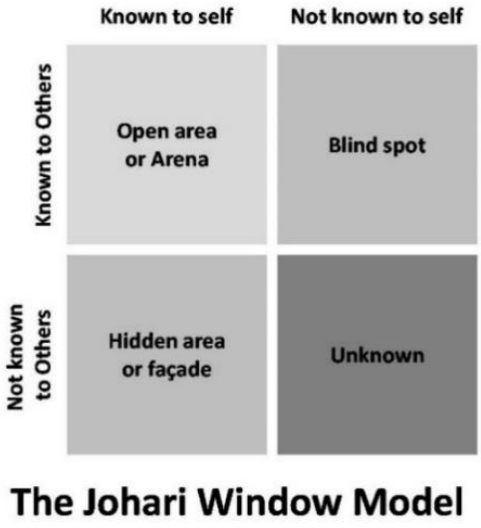


Figure 17: The Johari Window Model [40]

FPGA Prototyping

The need to accelerate design verification beyond software-based simulators inspired the three founders of Quickturn Design Systems (Quickturn) to develop a radical new hardware-based solution to address the chip verification needs of the late-1980s – and they named it "Emulation." The three Quickturn founders, Michael D'Amour, Steve Sample, and Tom Payne, were EDA veterans from Daisy Systems and Silvar-Lisco – and it should come as no surprise that Tom Payne's expertise was partitioning. The three founders were no strangers to the verification challenges facing chip designers of the day – and their

inspiration was driven by two key elements;

- Build a box with multiple Field Programmable Gate Arrays (FPGAs) that behaved as a "sea of gates" that would model any chip design for operation at hardware speeds, and
- Provide an "umbilical cord" from the modeled chip design to a "target system" – which they would call In-Circuit Emulation, or "ICE".

Quickturn's emulators aimed squarely at the two chip verification needs of the day; 1) chip design verification running much faster than simulation, and 2) in-system operation of the modeled chip design as the ultimate test of chip design correctness – with real system interfaces connected to, and real software running on, the emulated chip design. Quickturn's compelling ROI proposition was that ICE would allow chip and system designers to "collapse', or "shift left", their chip development schedules and harvest substantial economic benefits from faster TTM. [41]

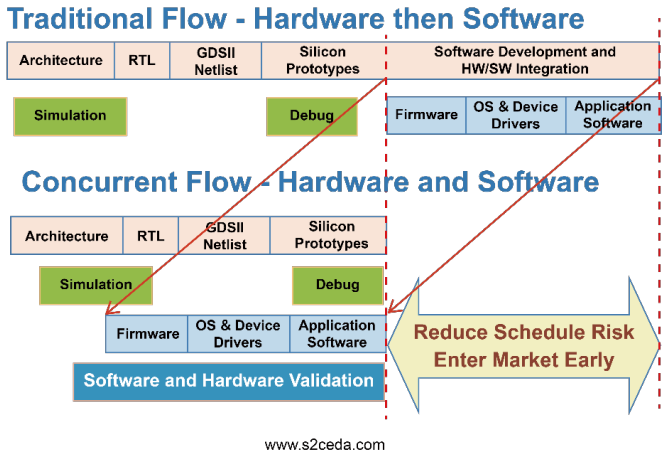


Figure 18: Concurrent Design Flow

Today, emulation is widely used by chip designers to verify chip designs before tape-out to silicon, and emulators still run much faster than software-based simulators – but, not surprisingly, emulators are still not

fast enough for some to keep pace with increased chip design complexity, and too costly for others. Modern emulators are loaded with well-intended evolutionary infrastructure that automates chip design mapping into the emulator (including incremental design changes), provides high design visibility for near-simulator debug, connection to virtual models of portions, or all, of the target system, and supports billion-gate chip designs – and they come with eye-watering price tags.

In the early days of emulation availability for chip verification, some chip designs still required FPGA prototyping – even though early emulation was virtually an FPGA prototype wrapped in a modest amount of EDA automation. For example, wireless chip design prototypes needed to be driven around in vehicles to test radio operation in the shadows of underpasses, hills, and large buildings – big emulation boxes were not portable enough. Similarly, electronic medical device prototypes needed to be carried around by real people to test artifacts caused by body movements. These chip designs were better suited to prototyping in a smaller footprint with a single FPGA. This realization leads to Quickturn positioning its emulation as a chip verification methodology that was "between" simulation and FPGA prototyping in the spectrum of available chip verification methods – Quickturn did not try to replace FPGA prototyping where prototyping was a better fit.

So, it is no surprise that many of today's chip designers are still looking to FPGA prototyping for yet faster chip design verification. In the very beginning, emulators were not much more sophisticated than today's FPGA prototyping products – Quickturn's first emulator would only support about 25K gates in each emulator box, did not have built-in timing analysis (imagine having to manually insert additional logic gates to add delay as a fix for timing problems!), and debug support was meager. To those who experienced early emulation and are still involved with SoC development – well, they are probably scratching their heads

and thinking, as Yogi Berra once famously wisecracked, “It’s like déjà vu all over again.”

FPGA prototyping as a verification method is being applied today to a broad spectrum of chip design types from monster 5G baseband processor designs to smaller wireless designs that fit into a single FPGA – and a plethora of chip design types in between. Some chip designers are driven to FPGA prototyping because they have a “*need for speed*”, while others are driven by a need to enable access to early hardware models of chip designs by dozens of software development team members so they can run software on the chip design before tape-out to silicon (even smaller chips can have extraordinarily complex design verification needs). To address today’s diverse chip design types, suppliers must offer complete FPGA prototyping solutions that address the five pillars of FPGA prototyping;

- Automate FPGA design mapping with flexible FPGA interconnect
- Run orders of magnitude faster than simulation/emulation
- Facilitate in-system-like operation
- Provide for effective debug
- Keep the cost affordable

The Prodigy Player Pro is a tool that works with the FPGA-based prototyping platforms from S2C. It integrates three development processes into one – it configures the prototyping, runs remote system management, and provides set-ups for multi-FPGA debugging. Such an integrated solution alleviates the pain of tackling the complex FPGA flow and plays a significant role in the FPGA prototyping methodology.

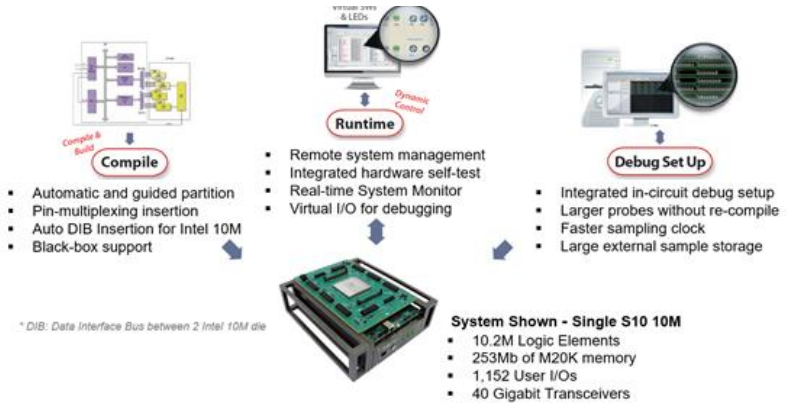


Figure 19: S2C's Prodigy Player Pro – Cockpit for prototype design and multi-debug set up

Design Partitioning for FPGA Prototyping

When an SoC design must be partitioned into multiple FPGAs to build an FPGA prototype, design partitioning adds complexity to the FPGA prototyping effort. Design partitioning adds to the first FPGA prototype bring-up effort, it adds to the time needed to update the prototype with design fixes, and it adds the complexity of maintaining good prototype visibility for debugging. While it is estimated that more than half of SoC designs will require multiple FPGAs to prototype, [42] the benefits of accelerated software development before silicon are so compelling that many experienced SoC developers are not deterred by the added complexity of design partitioning.

Design partitioning has been one of the guiding implementation considerations for multi-FPGA prototyping since – well, since early emulators were implemented using FPGAs. Rent's Rule describes an empirical relationship between pins per logic block and the number of gates in the logic block, and FPGAs have never had enough I/O pins to satisfy Rent's Rule. As it has played out, Rent's Rule has led to a host of unnatural design acts to get around the pin limitations of early FPGAs. To complicate the partitioning impact on prototyping, logic density will follow Moore's Law, but packaging and pin counts will not – the growth

in the number of FPGA I/O continues to lag the growth in FPGA logic.

However, maybe there is a glimmer of hope – system design modules naturally have smaller pinouts than arbitrary partitioning cuts, and Rent's Rule does not apply to SoC design modules ("Well, yes it does but weakly."). The implication of this is that, as the logic density and packaging technologies of the newer FPGAs continue to advance, FPGA logic and I/O pins will reach a point where any SoC design module will fit within a single FPGA – where automated user-guided partitioning may lead to simplified multi-FPGA prototyping, at least for the partitioning part of the prototyping effort.

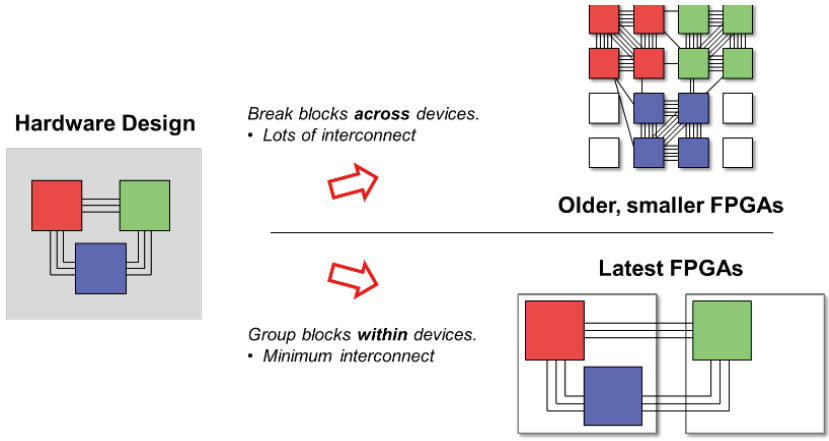


Figure 20: Automated User-guided Partitioning

Besides, thanks to the rapid evolution of SERDES technology, time-division-multiplexing (TDM) is frequently adopted to overcome the FPGA pin count shortage issue. With such methodology, multiple design signals are consolidated and transmitted through one FPGA pin and reassembled at the receiving side into their original format. In this way, we can save hundreds or even thousands of FPGA pins and finish the partition work more efficiently.

Expanding Prototyping Solutions

Today's FPGA prototype suppliers deliver on the promises of faster in-system chip design operation, automated design mapping, plug 'n play system interfaces, better debug – and more affordable verification solutions than emulation.

S2C's newest generation of FPGA prototyping products offers the latest FPGAs from Xilinx and Intel – and they are available in single, dual, and quad-FPGA variants.

Figure 21: S2C Single-FPGA Prodigy Logic System

In 2020, S2C announced a high-density FPGA prototyping platform, the Logic Matrix series, to better address high-complexity hyper-scale prototyping applications. A Logic Matrix packs 8 FPGAs in a single rack-mounted chassis, and it is sized to allow eight Logic Matrix to fit into a single standard server rack – thus supporting 64 FPGAs or chip designs up to 3.1 billion ASIC gates.

To simplify FPGA interconnect while addressing bandwidth and flexibility, S2C also introduced hierarchical connectivity: ShortBridge, SysLink, and TransLink, each with different granularity to manage local, Logic Matrix-to-Logic Matrix, and rack-to-rack interconnect. ShortBridge provides high throughput connectivity between neighboring FPGAs, SysLink connects FPGAs over high bandwidth

cables, and TransLink supports longer distance links between FPGAs with SerDes over copper or optical cables. Logic Matrix also offers server-class features such as real-time system monitors, professional cooling, and redundant hot-pluggable power supplies to ensure high reliability and robust operations. Logic Matrix's high-density architecture also further reduces the cost of ownership by taking up less server rack space or the precious benchtop real estate.

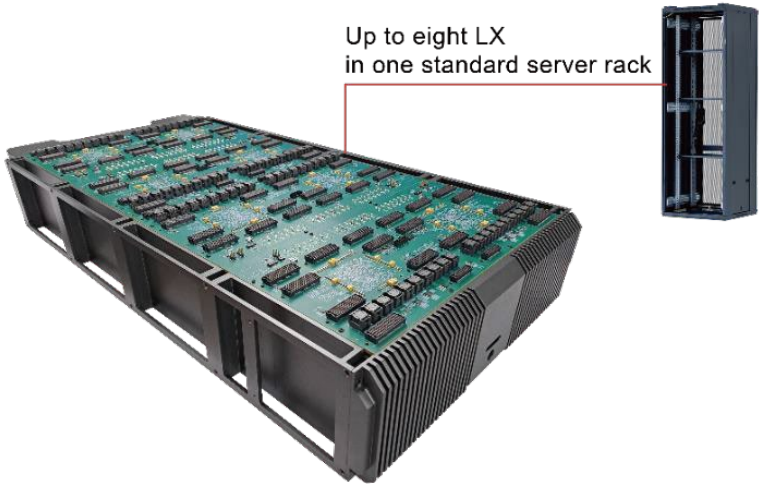


Figure 22: S2C Logic Matrix Series Platform

System Validation

Introduction

Moving from *Hardware Verification* to *System Validation* is an evolutionary transition, and it is more about *when* things are occurring vs. *what* things are occurring. There are some things you simply can't do at the earliest stages of SoC development. You can't run block-level RTL simulation before you have enough of the RTL completed for the block. You can't do IP integration at the RTL level until you have the rest of the RTL completed to interact with the IP. You can't' run firmware on the RTL until you have enough of the RTL working correctly for efficient firmware testing. During the nascent stages of SoC development, when the implementation of the system constituent components is embryonic, the *verification* focus is on specifications.

Specifications are the only recognized reference that developers can base decisions on to measure their progress towards tape-out sign-off on the SoC design. Designers must continually ask themselves the question: "Does my design conform to the specifications at every stage of development (*building the product right*)" – all the time trusting that the specifications faithfully represent what the user wants and expects (*building the right product*). Ideally, all system development would be done against a backdrop of a real system, running at real speeds, with real users "banging" on the system simultaneously while the system components were in the process of being developed. Sadly, that is not reality – not today.

Verification vs. Validation

Consider the IEEE Software Engineering standards definition for *Verification* (IEEE-STD-610). *Verification,* according to the standard, is:

"A test of a system to prove that it meets all its specified requirements at

a particular stage of its development." [43]

Validation, according to the IEEE standard, is:

"An activity that ensures that an end product stakeholder's true needs and expectations are met."

"*Validation* focuses on ensuring that the stakeholder gets the product they wanted."

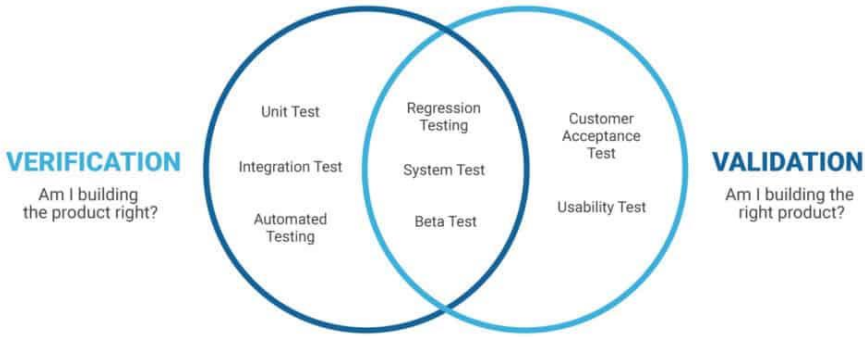


Figure 23: Verification and Validation

There is an old line of reasoning that today's silicon will always be an inadequate platform for designing tomorrow's silicon. If you think about today's SoC development environment, one of the major criticism of today's design tools is that they take too long. "*It takes too long to simulate my SoC.*" Or "*if I only had time to run more verification on my SoC before I tape out.*" Imagine being able to plug a model of your SoC into a real end-product before you commit your SoC to silicon – running at full design speed – with internal visibility and operational control – and with the ability to relate any discovered design faults back to the SoC reference design description for interpretation and resolution. The spectrum of today's design verification tools – software simulation, formal verification, emulation, FPGA prototyping – are all purposed to enable SoC designers to *verify* and *validate* their designs in less time with greater accuracy. Each successive tool enables longer periods of

operation of the design than the previous tool. If you need the fastest operation of your SoC design today while connected to a target system – you turn to FPGA prototyping.

Shift to System Validation

In today's SoC development environment, with mounting TTM pressure, there must be a deliberate effort to pull *System Validation* forward - to move the focus from *verification* to *validation* and to shift the focus to producing the *right product* as early as possible. [44] Realizing early *System Validation* is the primary purpose of EDA tools like emulation and FPGA prototyping. With more real system-level inputs, system validation helps to find the bugs not found in the verification stage. Software simulation takes SoC developers only so far – then, for more accurate real-world system environment modeling and more real-world performance, the *verification* and *validation* process must invariably move into the hardware domain with emulation and FPGA prototyping.

Considering that the end-product is also taking shape progressively from specification to implementation, *System Validation* can only be accomplished with reference to the end-product, or a portion of the end-product, is complete enough to begin *validation*. Knowing if the system design will work the way it was envisioned to work in the user's hands at any stage of development is the intent – but, at early stages of system development, the ability to determine this depends on how complete the previous stage of design implementation is, and how accurately the user environment can be modeled. If SoC developers move too quickly to the next stage of *validation*, the next level *validation* effort will be bogged down by issues that are not relevant to the next development *validation* stage – e.g., firmware and software *validated* on a buggy SoC design. Software developers want to spend their time debugging their software, not the tools. They can only report that the firmware does not work as expected, try to characterize the fault, hope

that the root cause of the problem can be quickly identified by the SoC designers, and wait for an SoC design fix to be applied so they can continue their work.

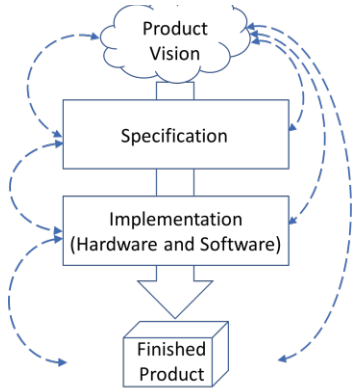


Figure 24: System Validation

Software Content

If there is any doubt about how significant the firmware and software development effort is for today's SoCs, let's be clear – Semico reported several years ago that "software design costs have eclipsed silicon design efforts and have become the largest portion of the SoC creation effort." [45] This means that, to assure the fastest TTM, the SoC development effort must comprehend multiple parallel paths, with the development of all the system's components staged to be progressively ready in time for *validation* with the other system components as early as possible – SoC design, IP integration, firmware design, software design, silicon bring-up, and system bring-up.

Furthermore, the earliest possible time arrives for the implementation of any dependent component (silicon, firmware, or software) when it is "correct enough" to support the development of any successive and dependent component. The longer into the development process that a design fault persists (silicon, firmware, software) – the more costly it will be to remedy the fault.

FPGA Prototyping

When an FPGA prototyping platform is planned for SoC, firmware, software, and system validation, the FPGA prototype development will begin early (before the RTL is ready for prototyping) so the prototype platform is ready when the SoC RTL design is mature enough to run on hardware in the FPGAs. Each SoC development project will have its own criteria for when the RTL is ready to run on the prototype. Some will start prototyping when the RTL is 75% verified in simulation. The other will start when the design fault discovery rate in simulation falls below a certain threshold. FPGA prototyping is purposed for long periods of continuous operations of the SoC design, and debug and reconfiguration for design fault fixes are slower than it is for simulation. Trying to prototype when there is a high frequency of faults can be inefficient because developers will spend more time analyzing faults and reconfiguring the FPGA prototype for fixes than they verify the SoC design.

FPGA prototyping should be managed as a distinct SoC development effort, with FPGA prototype specialists working closely with the SoC RTL design specialists. When design faults are discovered with the FPGA prototype, the RTL specialists must lead the root-cause analysis and development of fixes to the RTL design as well as modifications to the FPGA prototype. Managing RTL revisioning and design fixes will be important disciplines for the two teams to minimize inefficiencies that might arise when a new design fault is identified on a previous version of the RTL that has already been modified with a previous design fix. Keeping the FPGA prototype aligned with the latest RTL version will streamline the verification and validation process – but it is not something the FPGA prototyping team wants to do because it requires a relatively high effort when compared to simulation.

The FPGA prototype will have one or more FPGAs and come equipped

with debugging features that enable visibility into the SoC design being modeled. In addition, the prototype will have external connections to the modeled SoC target system to start system-level validation early. Moreover, the prototype may have connections to a host PC that models system-level data streams intended to mimic the SoC system stimulation. S2C is one FPGA prototype supplier that offers complete solutions for system-level modeling.

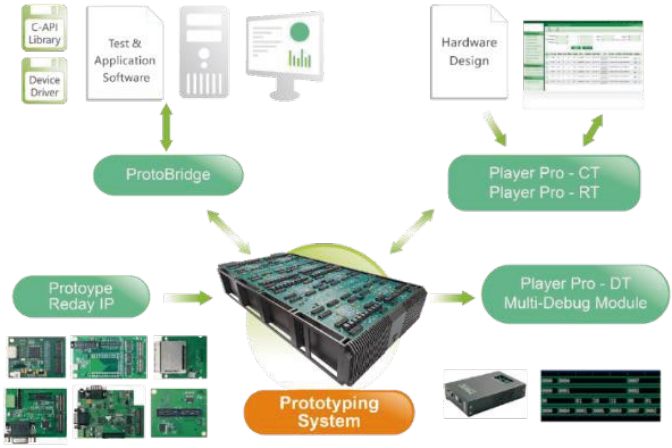


Figure 25: Complete FPGA Prototyping Solutions

S2C's most popular prototyping product is its Prodigy Logic System which is available in Single, Dual, and Quad configurations. Prodigy Logic System is an all-in-one design. In addition to an integrated power supply design that supports remote power cycling, Prodigy Logic System also supports other runtime operations such as multiple programmable global clocks, FPGA programming through Ethernet, USB, JTAG, and micro-SD. It also supports an optional on-board battery for binary file encryption.

For debug, Prodigy Logic System may be paired with S2C's Multi-Debug Module (also called MDM), which provides concurrent waveform viewing from multiple FPGAs in a single host PC window. MDM

supports waveform tracing for up to 32K probes per FPGA in 8 groups of 4K probes and external hardware for storage of up to 8GB of waveform data. Waveform capture triggers can be specified as single events or combinatorial events, and trace data may be transferred to the host PC.

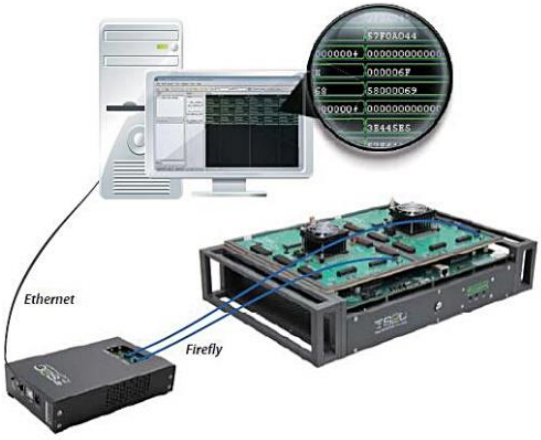


Figure 26: Prodigy Multi-Debug Module

For early *system validation* demanding stimulus data from a system-level software model, ProtoBridge can be a crucial tool. As previously highlighted, ProtoBridge can provide a high-throughput data channel between the host PC running RTL simulator or high-abstraction-level models and the FPGA prototype.

Software Development

Early Software Verification

Software is a critical component of product development effort, whether it is a system or an SoC design. As reported in previous sections of this book, software design costs have eclipsed silicon design efforts and have become the largest portion of the SoC creation effort. [46] And, like all competitive electronic products today, the pressure to complete system verification in less time with full system/SoC functionality running system software has never been greater. Since today's SoCs typically have a 9-month to 18-month development cycle to design the silicon (not including the time it takes for silicon fabrication), software development cannot wait for the availability of silicon before starting the verification process. Running the software stack on full system/SoC functionality before silicon will surface system/SoC software interoperability issues early in the SoC development cycle. If a serious SoC/software interoperability issue is discovered after the SoC is committed to silicon, and the SoC silicon requires a silicon re-spin, the re-spin will add several months to the product development time. So, early software verification delivers two benefits; it "shifts-left" the whole software development schedule, and it may prevent a silicon re-spin by uncovering a critical functional or performance issue prior to SoC tape-out.

It is common practice today for system & SoC development projects to plan for some amount of hardware/software co-verification prior to silicon. That is, running the product software stack on the system & SoC hardware while the components of the system & SoC are still under development – especially the SoC silicon. Effective hardware/software co-verification requires running the system & SoC at some minimum operating speed (ranging from megahertz to tens of megahertz), within an accurate representation of the "target" system (if not the actual

system itself). Software simulation tools are the workhorse tools for system verification, including software verification. With software simulation, system models can easily be tuned to be more "life-like", design fixes are quick and easy to implement, internal design visibility is easy to instrument, the design can be started and stopped at any time when design issues are encountered, and multiple copies of the simulation environment are easily distributed to large design teams in remote geographic locations at a modest price per user. Software simulation tools are leveraged to the extent of their capacity and performance.

Hardware-Dependent Software

Many of today's systems & SoCs require a deep interdependency between the software and the hardware. Designers have learned to manipulate SoC design dynamic operation with low-level software to achieve lower power operation and higher performance when called for, and more effective techniques for data security – but these additional operating modes contribute to increased system & SoC verification complexity with more corner cases that require verification.

For longer periods of software run time on the system & SoC hardware, and more accurate system modeling, developers may add hardware emulation to the verification tool mix. Besides running orders of magnitude faster than software simulation, hardware emulation supports in-circuit emulation, or ICE. ICE enables SoC designs modeled in the emulator to actually "plug" into functioning system hardware that can very accurately model the actual system hardware running the actual system software (sometimes the actual system itself). An iconic historic design tool milestone was achieved in the early 1990s when emulation of Intel's Pentium processor was used to boot the operating system and run a spreadsheet application – well before Pentium silicon was available. [47]

While ICE is a compelling verification approach, it does present some physical challenges for implementation. Consequently, today's emulation tools have evolved to include software virtual models of system hardware and semiconductor intellectual property (IP) that can be easily configured to represent system hardware components for hardware/software co-verification. These virtual models can generate large amounts of transaction-level data to stimulate the SoC design while the software is running on the emulated SoC design environment.

Extending Verification Performance with FPGA Prototyping

Similar to software simulation, hardware emulation has its performance limitations too. Emulation supports the operation of the system& SoC at frequencies in the low megahertz range. When emulation reaches its limits of performance and system modeling, and TTM schedules call for more comprehensive hardware/software verification early in the development process, SoC developers turn to FPGA prototyping. The use of FPGA prototyping for early hardware/software co-verification is growing rapidly, driven by SoC designs for the Internet of Things (IoT), Autonomous Driver Assistance Systems (ADAS), and healthcare – where tighter interdependencies exist between power and performance, and can only be verified with hardware/software co-verification.

Once the SoC design is committed to silicon by taping out, FPGA prototyping enables software developers to continue with verification while silicon is being fabricated. It will be several months, and many millions of irrevocably committed dollars, before the silicon is available for moving to the next level of software verification with real silicon. Discovering SoC design issues before silicon, even during the silicon fabrication process, may save serious time and money. For example, if an SoC design issue is discovered that only impacts one or a few silicon mask layers, the cost for a revised mask set is reduced compared with the cost of full mask set, and the silicon fabrication process with the old

mask set is restarted before it runs to completion. Needless to say, the development team is highly motivated to reduce the number of times they need to pay for new mask sets and re-spin SoC silicon. It may not be intuitively obvious that, while the irrevocably committed silicon re-spin dollars and extended engineering time are significant, the potential dollar value of market share losses when estimated over the projected product lifetime resulting from a delayed market entry can be much larger.

Once it is up and running, hardware/software co-verification may be viewed as having three basic recurring phases;

1. Run-Time – run the hardware/software until the next design issue is encountered (depending on the nature of the design issue, hardware/software verification may or may not be able to progress while the design issue is being fixed).

2. Root Cause Analysis (RCA) – identify the hardware/software design artifact that is causing the design issue (some problematic design artifacts may appear to be random, and most actually occur well before the symptoms of the artifact are detected during system operation).

3. Corrective Action (CA) – design a modification to the hardware design and/or the software design to fix the design issue, distribute the design modification to the system/SoC development team, and update the verification tools with the design revision.

When a design issue is encountered, the RCA for the design issue can be tricky – it may not be immediately obvious from the symptoms whether the design issue is attributable to the software, the SoC design RTL, or if it is being caused by an artifact of the FPGA prototype's system model.

It is easy to envision that the total hardware/software verification time is the sum total of all of the hardware/software Runs Times, plus the sum total of all of the RCA and CA times. The Run Times are very fast with

FPGA prototyping, but the RCA/CA times are unpredictable and totally dependent on the developer's ability to identify the design issue and quickly devise an effective design fix – it will be a hardware/software team effort. Another consideration for managing total hardware/software verification time is how long it takes to deploy the design fix to the FPGA prototyping platform – it would not be helpful if the FPGA prototyping revision deployment time is extended, and days will matter.

Early in the development process, when frequent design faults are still being encountered, Run Times can be relatively short between design issue discoveries. So, project management will be well advised to plan for a smooth and well-documented design revisioning process when design revisions are needed. When a design issue is encountered, the development team must react and drive a fast RCA/CA, the design issue must be documented and communicated to the rest of the development team, and the design revision must be distributed and deployed by the development team in a dynamic system development environment.

Interfacing Processor

One important prerequisite to software development is realizing the processor. Depending on the adopted product development methodology and the design phase, each design team may have their own preference in how the processor is implemented and interfaced.

When a brand new product is first defined, it is common to explore which processor should be used. To solidify the processor architecture, it is typical to conduct early software development over virtual platforms, such as QEMU and GEM5 as an instruction set simulator (ISS). The rest of the SoC and existing hardware IPs are placed in FPGA and connected to the ISS over high-bandwidth bus protocols. This approach helps to select the proper processor core, define the software

architecture, and start the software porting tasks in the early design stage.

Once the processor core is chosen, design teams can replace the virtual processor platform with a real hardware processor platform. In addition to improving the data transfer efficiency between the processor and the FPGA, using a real processor can enable more accurate behaviors and performance measurements. As an alternative, design teams may also look to deploy hardware processor via Intel SoC FPGAs [48] or Xilinx Zynq SoC [49], FPGAs with integrated hardcore processors, to further consolidate the prototyping environment for software development. These SoC FPGAs provide more flexibility and can easily interface with existing FPGA prototyping platforms. The programmability on both ends is useful for protocol translation and additional processing when required. Software code can execute at high speed with high cycle accuracy over such configuration.

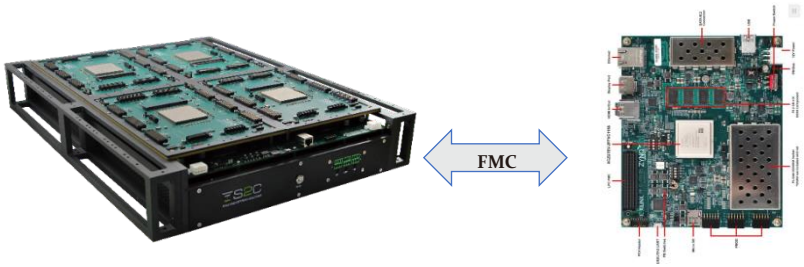


Figure 27: Connecting Xilinx Zynq platform with S2C Prodigy platform for fast software development

For design teams that would like to fine-tune firmware cores, critical RTOS sessions, or the processor micro-architecture, a soft processor IP is needed. With the soft processor IP inside, the FPGA prototype can provide more visibility, control, and flexibility. On the flip side, integrating the processor IP into the FPGA prototype typically requires multiple FPGAs. As a result, tools designed for multiple-FPGA debugging, such as S2C's MDM [50] and Synopsys' DTD [51], are pretty

valuable. These tools offer an effective and efficient debug approach for validating SoC designs to provide good visibility to the SoC behavior without sacrificing execution performance.

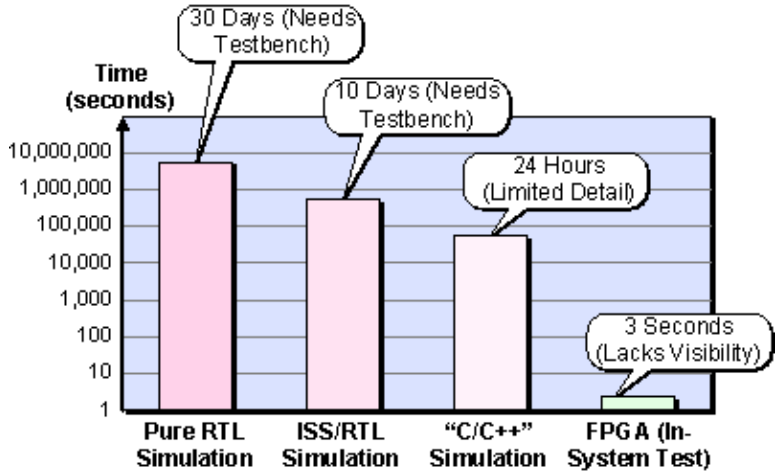


Figure 28: FPGA-based prototypes offer an extreme performance advantage over various software simulation techniques [52]

Compatibility Testing

Introduction

According to a Wikipedia "stub" article definition of Compatibility Testing [53] that references the ISO 25010 standard for System and Software Quality Models, and with some generalization by this author to apply the definition to SoC-based systems and software, *Compatibility* may be defined as: The degree to which a system or product can exchange information with other systems, while sharing a common environment and resources, without detrimental impact on any other system or product. The article goes on to add a subtle distinction between *Compatibility* and *Interoperability* by re-labeling the degree of *Compatibility* as "co-existence", and defining the degree of *Interoperability* as: "The degree to which two or more systems, products or components can exchange information and use the information that has been exchanged." Having established a context for a definition of Compatibility Testing, the article defines *Compatibility Testing* as: "Information gathering about a product or software system to determine the extent of co-existence and interoperability exhibited in the system under test."

Another definition for *Interoperability* highlights an important consideration not captured in our definitions above: "*Interoperability* refers to the basic ability of computerized systems to connect and communicate with one another readily, *even if they were developed by widely different manufacturers in different industries*. Being able to exchange information between applications, databases, and other computer systems is crucial for the modern economy." [54] The *Interoperability* consideration that development of systems and products will be performed by different engineers, working for different companies, in different markets, shines a bright light on the broader challenge of SoC-base product Compatibility Testing – developers must

consider that their products *will be* expected to exchange information with a myriad of other products with a completely different development provenance.

Consequently, as today's SoC-based systems and products have exploded in complexity, the scope of Compatibility Testing of these SoC-based systems and products has increased dramatically. Developers need to consider a wide variety of environments and resources that their systems and products must operate in – and that the critical functionality (including electrical specifications and performance) must be confirmed prior to committing to silicon because of the high risk that some minor incompatibility will impair the quality of the silicon in a way that requires a silicon re-spin.

Industry Standards

The availability of standards in the semiconductor electronics industry has gone a long way to reducing the required scope of *Compatibility Testing* – but the standards themselves are becoming very complex, are constantly changing with new standards generations, and are subject to subtle misinterpretations by different product developers. Today's standards-based commercial semiconductor IP has matured to the point where IP adherence to the standards has been thoroughly tested by the IP supplier, and possibly by some of their customers, before being delivered to SoC developers. In some cases, the specifications for the next generation of a standard may still not have been frozen when the development of a new SoC-based product must be started – be reminded that SoC-based product development takes several years to complete, so some aggressive SoC-based product developers may feel compelled in competitive markets to start a new project incorporating a new standard when the standard is still changing but is expected to be frozen prior to SoC commitment to silicon.

The primary driver of the commercial semiconductor IP market has been the management of SoC design complexity for reducing SoC development cost and TTM, enabling an SoC design team to focus their development effort on product differentiation in their area of subject matter expertise, and that industry standardization of the IP assures a high level of interoperability with all commercial products that claim to operate according to the standards. Being able to implement complex SoC-based systems and products with large blocks of pre-verified, standards-based functionality dramatically reduces the Compatibility Testing burden for new products – under the assumption that these IP blocks are designed and tested to functional and electric standards that are widely available and strictly observed by the entire professional product development community.

Software Enables Comprehensive Compatibility Testing

SoC-based product developers today take advantage of early FPGA prototyping to extend SoC *Compatibility Testing* to include some, if not all, of the product's software running on the SoC-based hardware before silicon – a testimony to the claim that the primary application of FPGA prototyping today is for early hardware/software co-verification. Execution of a product's software stack – from the device drivers at the bottom of the software stack, to the operating system, and to application software – provides another level of improvement in verification environment accuracy to better support early *Compatibility Testing* with a more comprehensive coverage. For some applications, SoC development is supported by FPGA prototyping tools that enable developers to run the product's full software stack on a prototype of the SoC-based hardware prior to silicon tape-out – perhaps not at the full SoC silicon speed, but at sufficient speeds to exercise the full software stack on the SoC-based hardware. For example, at the low end of SoC complexity, new Bluetooth IP developers are using today's FPGA prototyping technology to operate their designs at speed up to 24 MHz –

fast enough to run the full software stack prior to tape-out, and fast enough to allow an FPGA prototype of one Bluetooth radio to "talk to" an FPGA prototype of another Bluetooth radio as part of their *Compatibility Testing*.

STACK	What it does?
Embedded software application	Control algorithms, processing, services
Application framework	Security & safety frameworks
Operating environment	AutoSAR classic, AutoSAR Adaptive, Inputs/Output channels
Embedded virtualizations	Real-time OS, ECU abstractions
Firmware	Boot-loaders, secure-storage, secure-threading
Hardware	Silicon based devices, micro-controllers, single or multiple layered boards

Figure 29: Siemens - ECU and its Software Stack [55]

Modern FPGA prototyping systems offer software developers a cost-effective, near-real-world verification environment, a desktop platform that enables them to start SoC-based hardware/software co-development much earlier than a simulation-based or emulation-based verification platform, providing that the software executing performance is feasible – and it is all about the speed of the platform. Indeed, simulation is just too slow for comprehensive software testing, and if the software testing cannot begin until after silicon, TTM will be extended. Subsequently, it is not uncommon that further delays may be caused by a silicon re-spin resulting from some crippling SoC/software incompatibility.

"How about emulation?" you may ask. While emulation is hardware-assisted and it performs an order of magnitude faster than simulation, it is still an order of magnitude slower than FPGA prototyping. It is also too expensive for each software developer to have their own desktop emulator. We can easily model, by making a few assumptions on the setup time of emulation vs. FPGA prototyping, system performance, and the number of test runs, to find out the time-saved crossover point at which FPGA prototyping is more advantageous. [Prototyping/Emulation Performance Calculator https://w2.s2ceda.com/resource-

library/prototyping-calculator]

Compatible Testing in the Cloud

The need for adequate system validation and compatible testing is convoluted by the vast combinations of hardware configurations and peripheral connections, thus making it necessary for project teams to conduct testing in parallel to meet project schedules. This calls for a validation platform that is both capable of supporting concurrent testing operations and scalable to provide sufficient resources. Yet another challenge to overcome is the precise constructions and reconstructions of testing environments to ensure the validity of the compatibility testing itself. Both criteria calls for a hardware-assisted validation platform – in the cloud. Not only would a cloud-based solution provides the scalability, the resource and the environment to address the challenges mentioned, it can lower cost through higher utilization rate and simplify deployment and maintenance.

Figure 30: S2C Scalable Prodigy FPGA Prototyping Platforms

S2C's Prodigy Prototyping technology provides industry-leading remote management capabilities, ranging from remote FPGA download, virtual IOs, switches, UARTs, LEDs, and remote power cycles, to enterprise-class server/client management software, to provide effective resource

sharing and management. Such remote access, especially with prototyping hardware deployed in masses, provides software engineers with the means to run massive parallel regression tests and conduct analysis on multiple platforms simultaneously. To bring remote access and utilization management to the next level, S2C assisted customers in building a system validation and compatibility testing platform with more than one hundred FPGAs. Combing S2C's Neuro cloud management software and VMware virtual machines to manage the high-density Logic Matrix hardware, a resilient and flexible FPGA prototyping cloud service was constructed within six weeks. Fewer servers, lower total cost of ownership, but large design capacity and benefiting a large number of users. After completion, hundreds of users in the enterprise, no matter where they are, can remotely use the hardware platform, never interfere with each other, and efficiently perform compatibility testing, system validation, and software development concurrently.

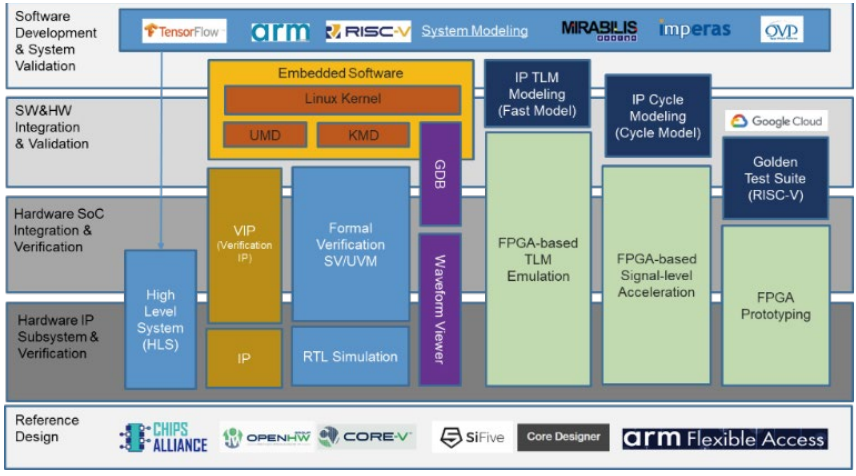


Figure 31: Various design and verification technologies for different design stages and SoC integration levels

References

[1] "TSMC brings 1,600 jobs to Arizona with new $12B factory," AZBIGMEDIA, 19 5 2020. [Online]. Available: https://azbigmedia.com/business/tsmc-brings-1600-jobs-to-arizona-with-new-12b-factory/.

[2] Harry Foster, "The 2020 Wilson Research Group Functional Verification Study - Part 1," Siemens, 5 11 2020. [Online]. Available: https://blogs.sw.siemens.com/verificationhorizons/2020/11/05/part-1-the-2020-wilson-research-group-functional-verification-study/.

[3] "1947 – Invention of the Point-Contact Transistor," Computer History Museum, [Online]. Available: https://www.computerhistory.org/siliconengine/invention-of-the-point-contact-transistor/.

[4] "1958 – All semiconductor 'Solid Circuit' is demonstrated," Computer History Museum, [Online]. Available: https://www.computerhistory.org/siliconengine/all-semiconductor-solid-circuit-is-demonstrated/.

[5] "1960 – First Planar Integrated Circuit is Fabricated," Computer History Museum, [Online]. Available: https://www.computerhistory.org/siliconengine/first-planar-integrated-circuit-is-fabricated/.

[6] "1963 – Standard Logic IC Families Introduced," Computer History Museum, [Online]. Available: https://www.computerhistory.org/siliconengine/standard-logic-ic-families-introduced/.

[7] Jerry Heller and Irwin Jacobs, "Viterbi Decoding for Satellite and Space Communication," *IEEE Transactions on Communication Technology,* vol. 19, no. 5, pp. 835 - 848, 10 1971.

[8] "The Story of the Intel 4004," Intel, [Online]. Available: https://www.intel.com/content/www/us/en/history/museum-story-of-intel-4004.html.

[9] "Philips Data Sheet – PLS100 Programmable Logic Array," 22 10 1993. [Online]. Available: https://docs.rs-online.com/7761/0900766b800255f1.pdf.

[10] S. Leibson, "'XILINX DEVELOPS NEW CLASS OF ASIC.' Blast from the Past: A press release from 30 Years ago, yesterday," Xilinx, 3 11 2015. [Online]. Available: https://forums.xilinx.com/t5/Xcell-Daily- Blog/XILINX-DEVELOPS-NEW-CLASS-OF-ASIC-Blast-from-the-Past- A-press/ba-p/663224.

[11] Paulmcl, "Why did EDA have a hardware business model?," 21 1 2009. [Online]. Available: http://edagraffiti.com/?p=151.

[12] P. McLellan, "A Brief History of ASIC, part I," SemiWiki, 21 8 2012. [Online]. Available: https://www.semiwiki.com/forum/content/1587-brief-history- asic-part-i.html.

[13] "Intel 80386," WIKIPEDIA, [Online]. Available: https://en.wikipedia.org/wiki/Intel_80386.

[14] I. Gelsinger et al, "Coping with the Complexity of Microprocessor Design at Intel – A CAD History," *IEEE Solid-State Circuits Magazine,* p. 6, 2010.

[15] X. Stephen Trimberger, "A Reprogrammable Gate Array and Applications," *Proceedings of the IEEE,* vol. 81, no. 7, pp. 1030 - 1041, 7 1993.

[16] Al Ries and Jack Trout, Positioning: The Battle for Your Mind, McGraw Hill.

[17] "The Pentium Processor," [Online]. Available: https://image.slidesharecdn.com/comparisonofpentiumprocessorwith80386and80486-120903115019-phpapp01/95/comparison-of-pentium-processor-with-

80386-and-80486-13-728.jpg?cb=1346673083.

[18] W.-Y. Koe, "Model Validation Methodology," [Online]. Available: https://old.hotchips.org/wp-content/uploads/hc_archives/hc05/3_Tue/HC05.S6/HC05.6.1-Koe-Intel-Pentium.pdf.

[19] N. Mike Butts, "Logic Emulation and Prototyping: It's the Interconnect (Rent Rules)," 8 2010. [Online]. Available: http://ramp.eecs.berkeley.edu/Publications/RAMP2010_MButts20Aug%20(Slides,%208-25-2010).pptx.

[20] Wikipedia, "Electronic system-level design and verification," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Electronic_system-level_design_and_verification.

[21] G. M. a. A. P. Brian Bailey, ESL Design and Verification: A Prescription for Electronic System Level Methodology, Morgan Kaufmann/Elsevier, 2007.

[22] F. Schirrmeister, "Electronic System-Level Design: Are We There Yet?," Semiconductor Engineering, 2018. [Online]. Available: https://semiengineering.com/electronic-system-level-design-are-we-there-yet/.

[23] Wikipedia, "Semiconductor intellectual property core," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Semiconductor_intellectual_property_core.

[24] SemiconductorEngineering, "Verification IP," SemiconductorEngineering, [Online]. Available: https://semiengineering.com/knowledge_centers/intellectual-property/verification-ip-vip/.

[25] MarketsandMarkets, "Semiconductor Intellectual Property (IP) Market," MarketsandMarkets, [Online]. Available: https://www.marketsandmarkets.com/Market-Reports/semiconductor-silicon-intellectual-property-ip-market-651.html.

[26] V. M. Research, "Semiconductor IP Market Size By Design IP, By IP Source, By Vertical, By Geographic Scope and Forecast," Verified Market Research, [Online]. Available: https://www.verifiedmarketresearch.com/product/global-semiconductor-intellectual-property-market/.

[27] C. Today, "Semiconductor IP market size worth $9.3 billion by 2026," Communications Today, [Online]. Available: https://www.communicationstoday.co.in/semiconductor-ip-market-size-worth-9-3-billion-by-2026/.

[28] AnySilicon, "What is an IP (Intellectual Property) core in Semiconductors?," AnySilicon, [Online]. Available: https://anysilicon.com/ip-intellectual-property-core-semiconductors/.

[29] D. &. Reuse, "Arteris IP Advances onto List of Top 15 Semiconductor IP Vendors," Design & Reuse, [Online]. Available: https://www.design-reuse.com/news/48119/arteris-ip-top-15-semiconductor-ip-vendors.html.

[30] S. Engineering, "Battling Fab Cycle Times," Semiconductor Engineering, [Online]. Available: https://semiengineering.com/battling-fab-cycle-times/.

[31] S. Engineering, "Process Corner Explosion," Semiconductor Engineering, [Online]. Available: https://semiengineering.com/process-corner-explosion/.

[32] E. Times, "New strategies shorten IP verification," EE Times, [Online]. Available: https://www.eetimes.com/new-strategies-shorten-ip-verification-process/.

[33] T&VS, "Hardware Verification," T&VS, [Online]. Available: https://www.testandverification.com/solutions/hardware-verification.

[34] Guru99, "Design Verification & Validation Process," Guru99, [Online]. Available: https://www.guru99.com/design-verification-process.html.

[35] M. Butts, "Logic Emulation and Prototyping: It's the Interconnect (Rent rules)," Mike Butts, [Online]. Available: http://ramp.eecs.berkeley.edu/.

[36] Barron's, "Apple Stock Is Rallying on a Report iPhone Demand Is Stronger Than Expected," Barron's, [Online]. Available: https://www.barrons.com/articles/apple-stock-rallying-on-report-iphone-demand-is-stronger-than-expected-51608049027.

[37] Goodreads, "Steve Jobs Quotes," Goodreads, [Online]. Available: https://www.goodreads.com/quotes/988332-some-people-say-give-the-customers-what-they-want-but.

[38] Wikipedia, "There are known knowns," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/There_are_known_knowns.

[39] Wikipedia, "Johari window," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Johari_window.

[40] C. Theory, "The Johari Window Model," Communication Theory, [Online]. Available: https://www.communicationtheory.org/the-johari-window-model/.

[41] E. Times, "Early verification cuts design time and cost in algorithm-intensive systems," EE Times, [Online]. Available: https://www.eetimes.com/early-verification-cuts-design-time-and-cost-in-algorithm-intensive-systems/.

[42] J. Dorsch, "FPGA Prototyping Gains Ground," SemiconductorEngineering, 25 8 2016. [Online]. Available: https://semiengineering.com/fpga-prototyping-

gains-ground/.

[43] Plutora, "Verification vs Validation: Do You know the Difference?," Plutora, [Online]. Available: https://www.plutora.com/blog/verification-vs-validation.

[44] V. Excellence, "Verification, Validation, Testing of ASIC and SOC designs – What are the differences?," Verification Excellence, [Online]. Available: http://verificationexcellence.in/verification-validation-testing-soc/.

[45] Semico, "Complex SoC Silicon and Software Design Costs," Semico, [Online]. Available: https://semico.com/content/complex-soc-silicon-and-software-design-costs.

[46] Semico, "Complex SoC Silicon and Software Design Costs," Semico, [Online]. Available: https://semico.com/content/complex-soc-silicon-and-software-design-costs.

[47] M. Butts, "Logic Emulation and Prototyping," 2010. [Online]. Available: http://ramp.eecs.berkeley.edu/Publications/RAMP2010_MButts20Aug%20(Slides,%208-25-2010).pptx.

[48] Intel, "Intel® Agilex™ FPGA and SoC," Intel, [Online]. Available: https://www.intel.com/content/www/us/en/products/details/fpga/agilex.html.

[49] Xilinx, "ZYNQ," Xilinx, [Online]. Available: https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html.

[50] S2C, "Prodigy™ Multi-Debug Module," S2C, [Online]. Available: https://www.s2ceda.com/products/prodigy-multi-debug-module.

[51] Synopsys, "HAPS Deep Trace Debug," Synopsys, [Online]. Available: https://www.synopsys.com/verification/prototyping/haps/haps-deep-trace-debug.html.

[52] M. Larouche, "How to achieve 100% visibility with FPGA-based ASIC prototypes running at real-time speeds," EE Times, 1 8 2007. [Online]. Available: https://www.eetimes.com/how-to-achieve-100-visibility-with-fpga-based-asic-prototypes-running-at-real-time-speeds/#.

[53] Wikipedia, "Compatibility testing," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Compatibility_testing.

[54] S. O'Connor, "What Is Interoperability, and Why Is it Important?," Stephen O'Connor, [Online]. Available: https://www.adsc.com/blog/what-is-interoperability-and-why-is-it-important.

[55] Siemens, "Embedded Software," Siemens, [Online]. Available: https://www.plm.automation.siemens.com/global/en/our-story/glossary/embedded-software/64121.