PROTOTYPICAL






The Emergence of FPGA-Based Prototyping for SoC Design

Includes All-New S2C FPGA PROTOTYPING SOLUTIONS Field Guide

DANIEL NENNI & DON DINGEE

A SEMIWIKI PROJECT

PROTOTYPICAL

The Emergence of FPGA-based Prototyping for SoC Design

Foreword by Mon-Ren Chene, CTO of S2C Inc.

Daniel Nenni
Don Dingee

Published by SemiWiki LLC
Danville, CA

Custom printed for S2C at DAC 2016 by OneTouchPoint, Austin TX

First printing: June 2016
Printed in the United States of America

"Implementing an FPGA Prototyping Methodology" Field Guide included with permission of S2C, Inc.

Edited by: Shushana Nenni

Contents

Foreword

The Future of FPGA Prototyping

Nearly two decades ago during our time at Aptix, my S2C co-founder, Toshio Nakama and I recognized the power of prototyping. At that time, prototyping was only accessible by large design houses with the budget and means to employ a prototyping architecture. We also recognized that FPGAs had become a popular alternative to the much more expensive and rigid ASICs. It was then that we both decided to team up to develop a prototyping board around an FPGA, and S2C was born. Our commitment to our customers has been to push the limits of what FPGA prototyping can do to make designing easy, faster, and more efficient. Our goal has always been to close the gap between design and verification which meant that we needed to provide a complete prototyping platform to include not only the prototyping hardware but also the sophisticated software technology to deal with all aspects of FPGA prototyping.

Fast forward to today and you'll find that FPGAs and FPGA prototyping technology has advanced so much that designers and verification engineers can no longer ignore the value that they bring, especially when dealing with the very large and complex designs that we see today. These advances have made FPGA prototyping poised to become a dominant part of the design and verification flow. This book will hopefully give you a sense of how this is achieved.

But what's next for FPGA prototyping? Having dedicated my time to working with our customers in developing the evolution of FPGA prototyping, I have figured out two things: FPGA prototyping needs to be part of the conversation early on in the design process, and FPGA prototyping needs to move to the cloud.

What do I mean by these two statements? Well, let's break it down.

Design for FPGA Prototyping

Making FPGA prototyping part of the design process early means actually thinking about how the design will be prototyped via an FPGA

as you design – Design for FPGA Prototyping. Designing for prototyping will significantly speed up the FPGA prototyping process downstream. It will aid in the act of synthesis, partitioning, and debug. I've outlined six ways that this is achieved:

1) Prototyping-friendly Design Hierarchies

Design architects can make the job of prototyping much easier for engineers to implement FPGA prototyping by modifying the design hierarchy to work better in a prototyping environment. The engineers who perform implementation or verification usually have very little ability to improve prototyping performance once the design hierarchy is fixed. The need to do partitioning down to the gate level can be removed if the size of each design block can be kept to one FPGA. Furthermore, modifying the design hierarchies early can help to avoid pin congestion as many times a design becomes very difficult to implement in an FPGA or becomes very slow because there's a central block that has tens of thousands of signals that need to go to multiple blocks in different FPGAs. Design architects can also ease prototyping by providing guidance to their FPGA implementation team(s).

2) Block-based Prototyping

Instead of hoping the entire design will magically work when mapped and downloaded to multiple FPGAs, bringing up sub-systems of the entire design, block by block, will allow quick identification of both design issues in a sub-block as well as any issues related with mapping the design to the FPGA(s). Block-based prototyping works well especially with designs that contain many 3rd party IPs that also needs a lot of real time testing and early software development.

And very often, designers don't even have the RTL source code for the IP blocks from 3rd parties (for example, ARM processors) and therefore cannot map the IP to the FPGAs themselves. This can be solved by requesting the IP provider to supply the encrypted netlist so that you can synthesize and partition the

entire design while treating that IP as a black-box. As long as you specify the correct resources (LUT, registers, I/Os), the prototype compile software should take those resources into account when partitioning to multiple FPGAs. You can then integrate the encrypted netlist during the place and route stage.

I've come across customers that want to do an FPGA implementation but are reusing some very old blocks with only the ASIC netlist and without RTL. Implementation becomes very difficult since the details of the design are unknown. These legacy designs are usually only accompanied by a testbench. In this case, the best approach is to covert the ASIC gates to an FPGA and to use a co-simulation environment (such as S2C's ProtoBridge™) to verify if the functionality of the block is correct before integrating it with the entire design. Unfortunately, this is still a painful process so designers should consider either not using those legacy blocks or re-writing them.

Note that a reconfigurable and scalable prototyping system is needed for a block-based prototyping methodology, as well as a robust partitioning and FPGA prototyping software flow.

3) Clean and Well-defined Clock Network for Prototyping

Many ASIC designs have tens or even hundreds of clocks and most of them are just for power management/saving. Even with the most complex designs there are usually a few real system clocks plus some peripheral clocks such as PCIe and DDR. Peripheral clocks usually reside in a single FPGA which has the actual external interface pins and therefore are easy to implement. System clocks, however, need to go to every FPGA and therefore should be clean for FPGA implementation.

ASICs use a lot of gated clocks to save power. Today's FPGA synthesis tools have advanced to take care of most of the gated clocks, but there may still be some gated clocks that go undetected and therefore cause design issues. This can easily be avoided by creating two different RTL clock implementations for the ASIC and the FPGA by using IFDEF.

Internally generated clocks can also be a problem for an FPGA prototyping environment as they all need to get on the FPGAs' global clock lines and synchronize among all the FPGAs. A Multi-FPGA prototyping system will have a limitation on how many of these global clocks can be supported therefore the number of the internally generated clocks should be restricted (or again use two implementations in the RTL: one for ASIC, and one for FPGA).

4) Memory Modeling

ASICs support many different types of memories while FPGAs usually support two types: synchronous dual port memories, or the use of registers and LUTs to build custom memories. The latter one consumes large amounts of logic resources and might cause place and route congestion. Most ASIC memories can be re-modeled to take advantage of the block memories in the FPGA but a manual process may be required to do that. Again, instead of having the engineers who try to implement the ASIC design in a FPGA model the memories, a better approach would be to have the architects plan the designs with two memory implementations both for ASICs and FPGAs. The RTL designers then code using IFDEF to have the two implementations. FPGA prototyping becomes easy by just instantiating the correct memory implementations.

5) Register Placement on the Design Block I/Os

FPGAs usually have a lot of register resources available for the design but most ASIC designs try to use less registers to save area and power. Ideally, all block I/Os should be registered for FPGA implementation to achieve the best results. At a minimal all outputs should be registered so no feed-through nets (which impact system performance by half) will be created after design partitioning. As a result, there will be a noticeably higher FPGA prototyping performance with this approach.

6) Avoid Asynchronous or Latch-based Circuits

Asynchronous circuits and latch-based designs are FPGA un-friendly. It is very hard to fine-tune timing in an FPGA with every FPGA having to be re-place and re-routed multiple times. These issues become even worse when the asynchronous circuits have to travel across multiple FPGAs.

Moving to the Cloud

We are living in an age where design teams no longer reside in one geographic location. No matter how big or small, companies have multiple design teams in multiple locations. A cloud-based FPGA prototyping system is an ideal way for dispersed teams to manage the prototyping process and resources.

Furthermore, as smaller IoT designs proliferate the market, FPGA prototyping must become accessible to these designers. Today's FPGA prototyping, although effective, can be costly for smaller IoT designers to adopt. The reusability of boards becomes less viable so costs cannot be amortized over multiple design starts. By moving to the cloud, FPGA prototyping solutions can become a shared resource and thus can reduce cost inefficiencies.

The future of FPGA prototyping is strong. It has and will continue to demonstrate itself as one of the most effective solutions to realizing the full potential of any design.

Mon-Ren Chene
CTO of S2C, Inc.
May 2016

Introduction: The Art of the "Start"

The health of the semiconductor industry revolves around the "start". Chip design starts translate to wafer starts, and both support customer design wins and product shipments. Roadmaps develop for expanding product offerings, and capital expenditures flow in to add capacity, enabling more chip designs and wafer starts. If all goes according to plan, this cycle continues.

In the immortal words of an engineering manager from once upon a time, "You didn't say it has to work." Chip designs have progressed from relatively simple to vastly complex and expensive, and the technology to fabricate them has shrunk from dimensions measured in tens of microns to tens of nanometers. Functions once dictated by distinctive symbols and lines or ones and zeroes in a table now center on executing powerful operating systems and application software and streams of rapidly flowing data.

Keeping the semiconductor cycle moving depends on delivering complex chip designs, completely verified with their intended software environment, faster and more accurately. Wafer fab facilities now approach tens of billions of dollars to construct and equip, producing massive high-capacity wafers. One malevolent block of logic within a chip design can cause expensive wafers to become scrap. If that flaw manages to escape, only showing itself in use at a critical moment, it can set off a public relations storm questioning a firm's design capability.

Verification is like quality: either it exists, or it does not. Only in a context of project management does partial verification of a design mean anything.

With the stakes so high for large, sophisticated chips, no prudent leader would dare avoid investments in semiconductor process quality. Foundries such as GlobalFoundries, Intel, Powerchip, Samsung, SMIC, TSMC, UMC, and others have designed entire businesses around producing quality in volume at competitive costs for their customers.

Yet, chip design teams often struggle with justifying verification costs, settling for doing only part of the job. A prevailing assumption is the composite best efforts of skilled designers using powerful EDA tools should result in a good design. Reusing blocks from known-good sources, a long-standing engineering best practice in reducing risk and speeding up the design cycle, helps.

Any team that has experienced a chip design "stop" knows better. Many stories exist of a small error creeping through and putting a chip design, and sometimes a reputation, at risk. The price of non-verification of both hardware and software of a design can dwarf all other investments, and instantly thwart any prior success a firm may have enjoyed.

This is where FPGA-based prototyping comes in. A complete verification effort has traceable tests for all individual intellectual property (IP) blocks and the fully integrated design running actual software (co-verification), far beyond what simulation tools alone can do in reasonable time. Hardware emulation tools are capable, fast, but highly expensive, often out of reach for small design teams. FPGA-based prototyping tools are scalable, cost-effective, offer improved debug visibility, and are well suited for software co-verification and rapid turnaround of design changes.

In this book, we uncover the history of FPGA-based prototyping and three leading system providers – S2C, Synopsys, and Cadence. First, we 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, power consumption, and reliability. 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. [1]

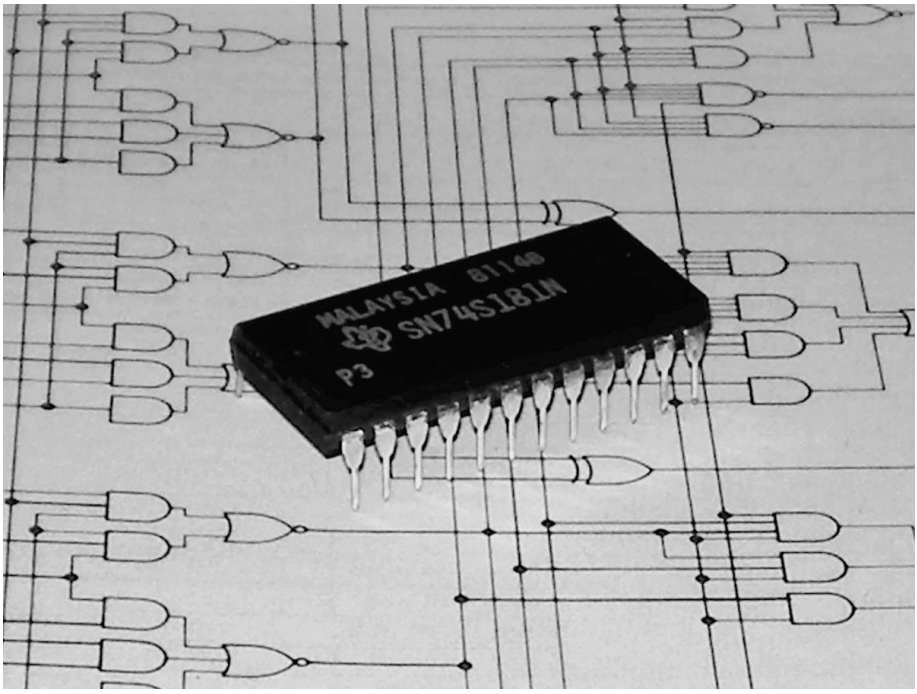
In 1958, Jack Kilby of TI demonstrated a simple phase-shift oscillator with one bipolar transistor and roughly hewn resistors and capacitors on one slice of germanium, with flying wire connections on the chip. By 1960, Fairchild teams led by Robert Noyce had a monolithic integrated four-transistor flip-flop running in silicon, a more stable and mass-producible material and process. [2, 3]

Standard small-scale integration (SSI) parts appeared in 1963, with Sylvania's SUHL family debuting as the first productized TTL family. 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. [4]

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 equivalent 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 product 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.

Image I-1: Texas Instruments SN74S181N 4-bit ALU with 63 logic gates



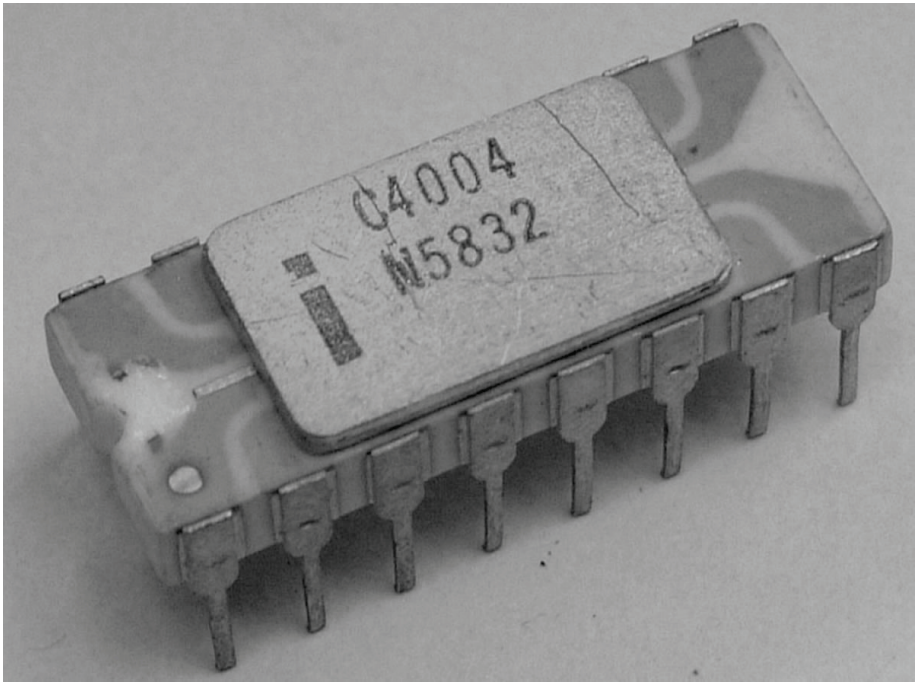
That changed drastically when large-scale integration (LSI) parts emerged. The early 1970s saw chips for digital watches, calculators, and the first integrated computer memories, each with a few thousand transistors. LSI parts were analogous to Mount Rushmore – carved from the monolith in labor-intensive 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 data sheet 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 respin.

Microprocessors, ASICs, and FPGAs

Entire "processors" combined LSI, MSI, and SSI chips. 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 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. [5]

Image I-2: 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 fabbed in 10 micron and running up to 740 MHz, the 4004 had 16 internal registers and offered 46 instructions. [6]

Feverish competition ensued as a slew of vendors created new 8-, 16-, and 32-bit microprocessor architectures during the late 1970s and early 1980s. 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 change was occurring in VLSI (very large scale integration), with several technological advances opening the way for new vendors.

The first usage of ASICs was as glue logic for improved integration, or as companion chipsets to microprocessors, often customized to a specific board design. A growing roster of ASIC vendors eventually including AT&T, Fujitsu, IBM, LSI Logic, Plessey, Toshiba, TI, and VLSI Technology were working to abstract the design flow with tools, IP libraries, and fab qualification. For the first time, design teams at a customer could create parts using "standard cells" and get them produced at moderate risk and reasonable lead times of a few months.

The average 32-bit microprocessor trended toward bloated, with more transistors to execute maddeningly complex instruction sets (CISC) with routine and not-so-routine operations and specialized addressing modes. Researchers tore into the flow of instructions, deciding that only a few mattered, and came up with the idea of Reduced Instruction Set Computing, or RISC. ASICs and RISC were a match made in heaven, and MIPS Computer Systems, Sun Microsystems, and others soon burst on the scene with new processor architectures.

Another breakthrough was near. Altera took an idea from the research halls of GE, combining the elements of EPROM memory 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", and easily erased with an ultraviolet light and reprogrammed as needed, in a matter of hours. Customers with

conventional digital logic schematic entry skills had access to relatively high customization with very low turnaround time. [7]

Image I-3: Altera EP300 programmable logic device

A different technology appeared on November 1, 1985, with the thundering headline, "Xilinx Develops New Class of ASIC". 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. [8]

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 significant. The SPARC I processor took four respins 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.

EDA tools from pioneers Daisy, Mentor, and Valid were being adapted from circuit board design to ASIC tasks. Rather than capturing a design

and tossing it into silicon and hoping for good results, more emphasis was being placed on logic simulation. EDA workstations were relatively fast, but simulation of a VLSI design was still a tedious and slow process, requiring skill to create a testbench providing the right stimuli. Still, ASIC simulation was cheaper than a failed piece of silicon and more dollars and several more months waiting for a fix. [9]

Major innovation was happening at Intel. Thanks to success in PC markets, its microprocessor families progressed rapidly. 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 saw wider use of RTL synthesis and a move toward CMOS standard cells, with only several specific logic blocks hand optimized.

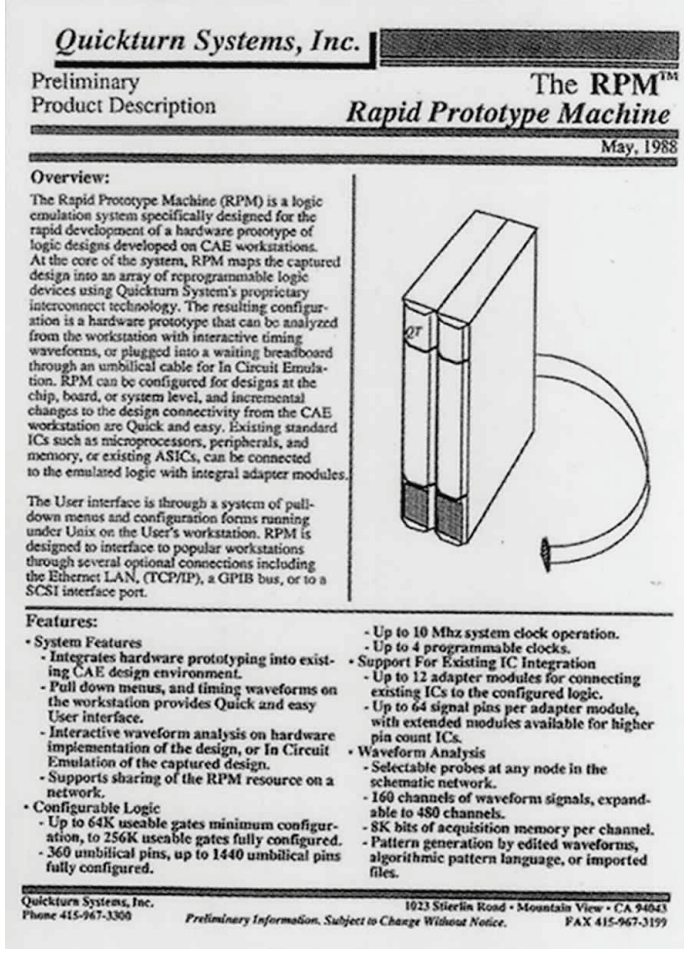
If Intel was to keep its winning streak going, development processes had to change to shorten the cycle time for increasingly complex parts. Beginning in 1986, Intel made a $250M investment for its next microprocessor design, including a proprietary system of EDA tools and practices. To enable fully 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 the standard cell library. The result was the 80486, breaking the 1 micron barrier with a staggering 1.18M transistors in 1989. [10]

Just as ASIC vendors discovered, Intel found simulation too slow and falling further behind. RTL simulations were chewing up more than 80% of Intel's EDA computing resources, and verification was growing non-linearly with processor size. A solution would come from an unexpected source: the FPGA community.

In May 1988, a small company – Quickturn Systems – introduced a new type of development platform aimed at ASIC designers. The Rapid Prototype Machine (RPM) used an array of Xilinx XC3090 FPGAs in a hypercube interconnect. Its software could take an ASIC netlist of hundreds of thousands of gates, partition it into the FPGA array, and

emulate the design up to a million times faster than software simulation. [11]

Image I-4: Quickturn Systems RPM datasheet



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
  - Integrates 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 useable 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

At those speeds, much more serious pre-silicon testing became feasible. Intel embraced the concept, putting its new P5 microarchitecture through its paces on a cluster of 14 Quickturn RPM systems – 7 for integer operations, 4 for caches, and 3 for floating point. In a November 1991 demonstration, an Intel VP ran a Lotus 1-2-3 spreadsheet on a P5 model in the Quickturn cluster. Customers considering RISC processors shelved plans, opting to wait for Intel to deliver. [12]

Running trillions of simulation clocks in this environment, Intel was able to debug development tools and clobber several errata and various operating system incompatibilities before committing to expensive silicon. More importantly, the effort sawed several months off their development schedule, leading to a timely release of the Pentium microprocessor in March 1993. [13]

Enabling Exploration and Integration

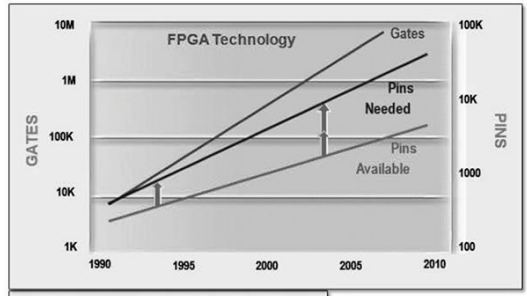
Strictly speaking, although it deployed Xilinx FPGAs and was essential in prototyping ASICs, the Quickturn RPM was the first commercial hardware emulator. From that point, advanced hardware emulator and FPGA-based prototyping platforms developed, on divergent paths for different use cases.

Hardware emulators are automatic, meant for big projects and broader application on more than one design. A user need not know details of the logic implementation, or how interconnects are organized. An arbitrary netlist for an ASIC is loaded, chopped into many smaller pieces, and spread out across many partitions – in the beginning, implemented with tens or hundreds of FPGAs.

These partitions are subject to a relationship known as Rent's Rule, describing a necessary ratio of logic gates to interconnect pins. Paradoxically, as FPGA logic capacities improved, pin counts fell behind and Rentian interconnect limitations worsened, 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 emulator can cost over $1M.

Prototypes are more specific, often configured and tuned for one project. Assuming adequate logic capacity and interconnect pins, a design can be synthesized for a single FPGA target, or perhaps partitioned across a handful of FPGAs with optimized interconnect. Rent's Rule becomes less applicable for a design of manageable size. This is the basic premise of FPGA-based prototyping, which becomes more and more attractive as FPGA logic capacities improve. [14]

Image I-5: FPGA gates versus pin count, courtesy Cadence



Commercial FPGAs are increasing gate count far faster than pin count necessary for emulation

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

Capacity cannot be effectively utilized, it is limited by I/O

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 typical in the industry evolved dramatically, looking less often like an enormous Intel microprocessor. System-on-chips, microcontrollers, application-specific standard product (ASSPs), and other designs take advantage of a growing field of IP for customized implementations.

Reuse and integration is now paramount. Using FPGA-based prototyping, stand-alone verification of individual IP blocks is cost-effective. 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 place FPGA-based prototyping platforms on desks. What-if scenarios run at 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 a critical product enhancement pre-silicon.

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 designers via networks. 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 starts over the last decade. Design starts are likely to be smaller and more frequent, with highly specialized parts targeting niches. 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.

NOTES

[1] "1947 – Invention of the Point-Contact Transistor", Computer History Museum, http://www.computerhistory.org/semiconductor/timeline/1947-invention.html
[2] "1958 – All semiconductor 'Solid Circuit' is demonstrated", Computer History Museum, http://www.computerhistory.org/semiconductor/timeline/1958-Miniaturized.html
[3] "1960 – First Planar Integrated Circuit is Fabricated", Computer History Museum, http://www.computerhistory.org/semiconductor/timeline/1960-FirstIC.html
[4] "1963 – Standard Logic IC Families Introduced", Computer History Museum, http://www.computerhistory.org/semiconductor/timeline/1963-TTL.html
[5] "Viterbi Decoding for Satellite and Space Communication", Jerry Heller and Irwin Jacobs, Linkabit Corporation, IEEE Transactions on Communication Technology, October 1971, pp. 835-848, http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=1090711
[6] "The Story of the Intel 4004", Intel, http://www.intel.com/content/www/us/en/history/museum-story-of-intel-4004.html
[7] "In the Beginning", Ron Wilson, Altera, https://www.altera.com/solutions/technology/system-design/articles/_2013/in-the-beginning.html
[8] "'XILINX DEVELOPS NEW CLASS OF ASIC.' Blast from the Past: A press release from 30 Years ago, yesterday", Steve Leibson, Xilinx, November 3, 2015, https://forums.xilinx.com/t5/Xcell-Daily-Blog/XILINX-DEVELOPS-NEW-CLASS-OF-ASIC-Blast-from-the-Past-A-press/ba-p/663224
[9] "A Brief History of ASIC, part I", Paul McLellan, SemiWiki, August 21, 2012, https://www.semiwiki.com/forum/content/1587-brief-history-asic-part-i.html
[10] "Coping with the Complexity of Microprocessor Design at Intel – A CAD History", Gelsinger et al, Intel, IEEE Solid-State Circuits Magazine, June 2010,

http://webee.technion.ac.il/people/kolodny/ftp/IntelCADPaperFinal2.pdf

[11] "A Reprogrammable Gate Array and Applications", Stephen Trimberger, Xilinx, Proceedings of the IEEE, Vol. 81 No. 7, July 1993, http://arantxa.ii.uam.es/~die/[Lectura%20FPGA%20Architecture]%20A%20reprogrammable%20gate%20array%20-Trimberger.pdf

[12] "Inside Intel", Robert Hof, BusinessWeek, June 1, 1992, http://www.businessweek.com/1989-94/pre88/b326855.htm

[13] "Pre-Silicon Validation of Pentium CPU", Koe et al, Intel, Hot Chips 5, August 10, 1993, http://www.hotchips.org/archives/1990s/hc05/

[14] "Logic Emulation and Prototyping: It's the Interconnect (Rent Rules)", Mike Butts, NVIDIA, RAMP at Stanford, August 2010, http://ramp.eecs.berkeley.edu/Publications/RAMP2010_MButts20Aug%20(Slides,%208-25-2010).pptx

Chapter 1: SoC Prototyping Becomes Imperative

Electronic design was changing in the mid-1980s. Brute force schematic capture was being supplanted by logic synthesis from hardware description language. Wire wrapping or stitching techniques for prototyping were being rendered obsolete as denser, higher pin count packages appeared. Printed circuit board technology was advancing rapidly, hand in hand with microprocessors, more complex ASIC parts, and new FPGA technology.

Competing to launch products on time meant creating chip designs faster. While hardware emulation proved very useful in ASIC verification, it was prohibitively expensive for smaller use cases. Sometimes, students and engineers just wanted to tinker with a design in a lab to prove a concept, or had only small production volumes in mind. If a project could bear the moderate cost of an FPGA, it was an ideal vehicle for experimentation. Two new use cases emerged for FPGAs: reconfigurable computing, and rapid prototyping.

Programmable Logic in Labs

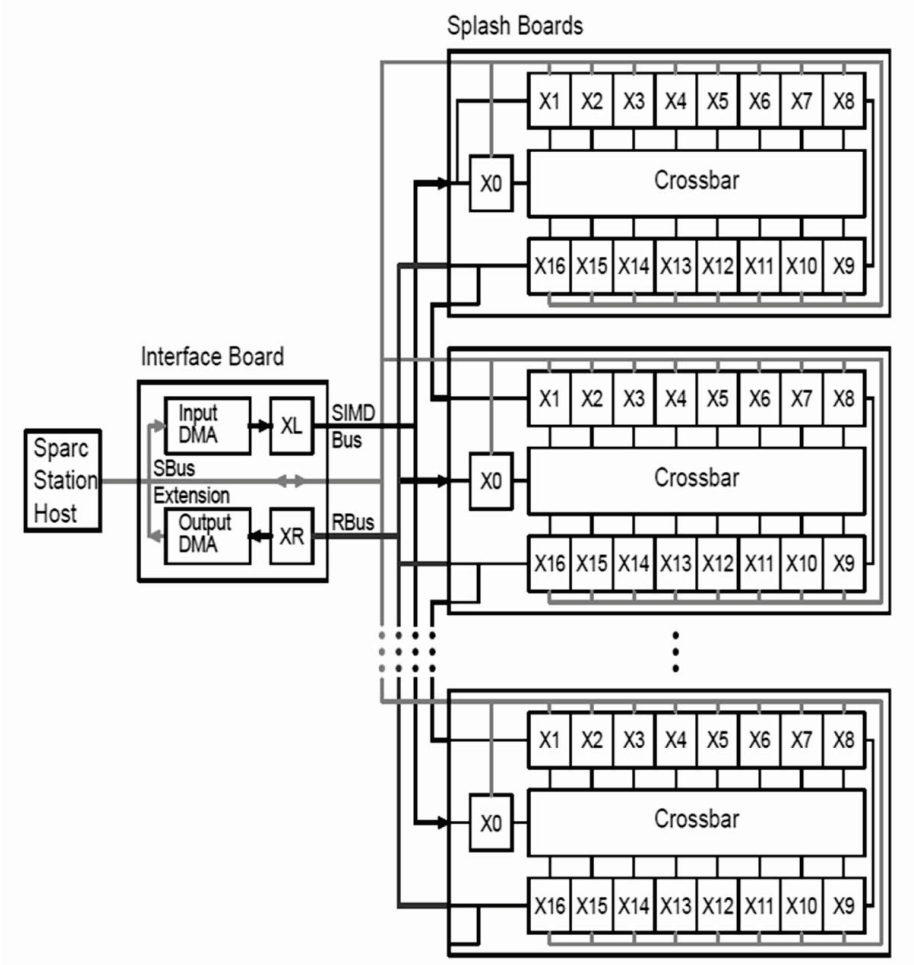
Reconfigurable computing was the holy grail for signal processing, an ideal application for FPGAs with DSP-like primitives. An FPGA card could be added as a co-processor to an engineering workstation, and its logic architected to provide efficient data flow computational capability. The same workstation and FPGAs could be reconfigured for different applications quickly, especially if Xilinx SRAM-based FPGA technology were used.

Prime examples of early reconfigurable computing platforms were Splash 1 and Splash 2, originally created to perform DNA sequence comparison. Created in 1988, Splash 1 was a VMEbus system with 32 Xilinx XC3090 FPGAs in a linear systolic array. While powerful, the Splash 1 architecture quickly proved to be limited by the available FPGA interconnect, typically in the range of 200 to 300 pins and subject to clocking and delay variables.

Splash 2 began in 1991, upgrading to XC4010 FPGAs with a crossbar interconnect. It allowed chaining of up to 16 array boards each with 16

processing FPGAs (with a 17th part controlling the crossbar), and added an SBus adapter for easy connection to a Sun Microsystems SPARCstation. [15]

Image 1-1: Splash 2 block diagram



Interconnect in hardware emulators and reconfigurable systems was becoming a hot topic. A new technology debuted in 1992, the Aptix FPIC (field programmable interconnect chip). Aptix parts used similar SRAM-based technology to provide around 1000 interconnect pins sans logic, somewhat relaxing Rent's Rule limitations and allowing

much more flexible configurations on a printed circuit board. FPICs were expensive, however, due to their high pin count packages. [16]

Academic researchers took these concepts in a new direction, scaling down to run HDL chip designs of a few thousand gates in smaller FPGA-based rapid prototype boards. The idea behind rapid prototyping was software would run immediately with full fidelity to production silicon – or not. If the project did not have access or could not afford to fabricate a chip, a rapid prototype could still prove the validity of the concept.

The first of these rapid prototyping boards appearing in 1990 was the AnyBoard from North Carolina State University. It returned to a simple linear array of five Xilinx XC3090s and soon added automated circuit partitioning built on Xilinx place & route software. The partitioning software understood interconnect pins, clock rates, and logic and I/O constraints. Researchers compared gradient descent algorithms with a multi-bin version of Kernighan and Lin graph partitioning, testing designs of varying complexity. [17]

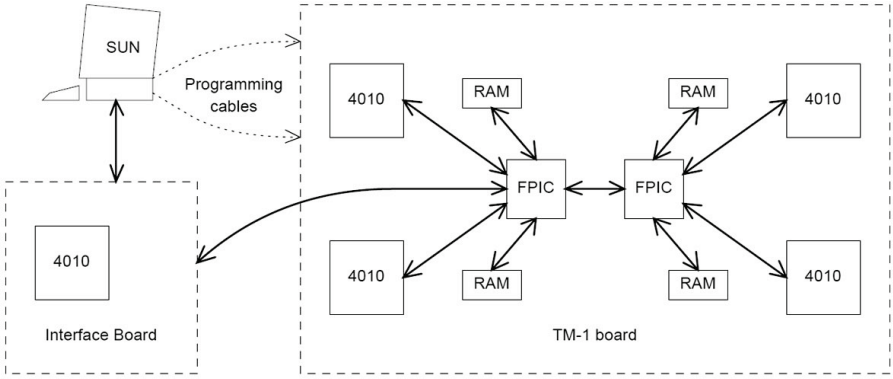
Also in 1990, researchers at Stanford University created Protozone, a single Xilinx FPGA on a PC add-in card for experimentation. Protozone became a jumping-off point for other research projects in FPGA programming, but as a degenerate single-part configuration it did little to advance partitioning and routing science. However, it did spur broader educational programs at both Altera and Xilinx to provide simple, low-cost FPGA boards for prototyping. [18]

From the University of California, Santa Cruz came the aptly named BORG in 1992. Two Xilinx FPGAs contained logic, two more held reconfigurable routing, and a fifth performed configuration and interfacing to a PC host. Much of the research focused on the problem of pin assignment using bipartite graphs and new algorithms for a two-commodity flow solution. (In a bit of irony, the first BORG prototype itself was wire wrapped.) BORG illustrated the complexity of programmable interconnect between parts even with relatively small FPGA packages. [19]

Another single-chip Xilinx XC3030 implementation debuted in 1993, the Generic Reusable Module (GERM) from Duke University. Researchers were promoting rapid design and prove out of subsystems, with concepts of VHDL design and IP reuse. Students were encouraged to build realistic designs in smaller pieces, then reuse those concepts for larger projects in subsequent courses that could still be completed in a semester. [20]

The growing popularity of the Aptix FPIC influenced the design of the Transmogrifier-1 at the University of Toronto in 1994. It indirectly scaled up the BORG concept, with four more powerful Xilinx XC4010 FPGAs interconnected by two FPICs, and a fifth FPGA providing the interface to a SPARCstation. Researchers used the platform to speed designs of three example projects: a Viterbi decoder, a memory organizer that emulated various configurations, and a logarithmic number system processor. Using SRAM blocks in the FPGAs allowed algorithm optimization compared to full-custom chip designs (multi-chip modules using FPGA dies), resulting in higher clock speeds and other implementation insights that were fed back to future modules. [21]

Image 1-2: Transmogrifier-1 block diagram

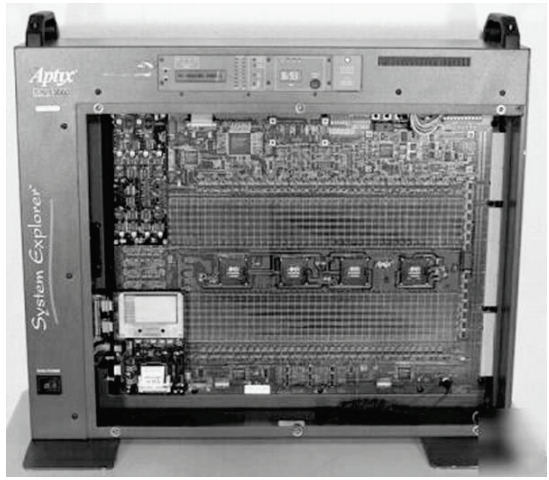


First Productization of Prototyping

These research projects were stimulating broader interest and exploring critical issues in FPGA-based prototyping, but were far from ready for prime time. Hardware emulation still had a significant head start in commercialization.

Aptix CEO Amr Mohsen reflected on the early business challenges with the FPIC parts, saying they were "probably two to three years ahead" and being asked by customers to move into complete, turnkey hardware emulation. At the Design Automation Conference in June 1994, Aptix launched two products. Explorer ASIC targeted single-chip emulation at 10 MHz using 21 Xilinx XC4000-class FPGAs and FPICs for interconnect, with automatic partitioning software provided by third party Software & Technologies. System Explorer MP3 provided general-purpose 50 MHz system-level emulation with configurable FPGA payloads and I/O, but lacking automatic partitioning tools. Automation would be added later with the System Explorer MP4 family in May 1996. [22, 23]

Image 1-3: Aptix System Explorer MP4



IKOS Systems bought its way into the hardware emulation market by acquiring Virtual Machine Works in May 1996. The VirtuaLogic SLI hardware emulator was productized and released by late 1996 with a basic 200K gate capacity upgradable to over 1M gates. VirtualWires technology created at MIT provided synthesis for FPGAs, avoiding a need to move toward ASICs as other vendors were doing. [24]

Major EDA players then moved in and competition got a bit ugly.

Meta Systems created the SimExpress emulator family in 1994, and after shopping itself to both Quickturn and Mentor Graphics agreed to a Mentor acquisition in May 1995. By early 1996, Quickturn and Mentor Graphics were involved in suit and countersuit over technology (some previously licensed to Quickturn in 1992), blocking sales of Meta Systems emulation platforms in the US for several years.

Mentor Graphics then licensed emulation technology from Aptix and promptly sued Quickturn again in 1998, unfortunately based on bogus claims in an Aptix engineering notebook. In a bid to resolve the patent issues, Mentor Graphics launched a hostile takeover for Quickturn in early 1998. It drew the attention of Cadence Design Systems, who raised the takeover offer to $253M and secured Quickturn by December 1998. Legal wrangling continued. [25]

With the hardware emulation providers locked in expensive battles over high end platforms, the door was open for lower cost solutions from smaller providers. Gidel, based in Israel, converted its expertise in FPGA-based reconfigurable computing to a commercial FPGA-based ASIC prototyping board in 1998 featuring an Altera FPGA. Also in 1998 The Dini Group in the US took its ideas from ASIC design consulting into its first commercial FPGA-based prototype board, the DN250k10 with six Xilinx XC4085 FPGAs. [26, 27]

ASIC complexity in both gate and pin counts had overwhelmed most FPGA implementations, even attempts with programmable interconnect. HARDI Electronics AB, a small Swedish firm, reinvestigated the problem and decided to route FPGA I/O to high speed connectors leading off board. By insuring impedance and trace length matching, external cabling could be used to complete connections in the desired configuration. The result was the first HARDI ASIC Prototyping System (HAPS) created in 2000, based on the Xilinx Virtex FPGA. To get larger configurations, HARDI began work on a board stacking scheme and bus interconnect – HapsTrak.

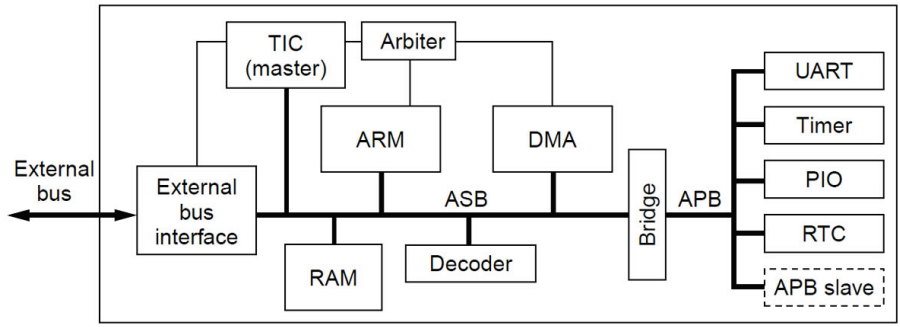
Fabless and Design Enablement

The third major change in the industry was the rise of ARM processor core technology and a corresponding increase in foundry capability.

Powered largely by the meteoric rise of the ARM7TDMI core introduced in 1995, designers outside of the traditional semiconductor companies were gaining confidence in their ability to create producible chips, entirely fabless. [28]

ARM understood that as their IP became more complex and foundries became more diverse, they needed to foster tools and methodology enabling their design customers to succeed. At first, ARM scaled via consulting in efforts led by Warren East beginning in 1994. Next, ARM introduced the Advanced Microcontroller Bus Architecture (AMBA) in 1997, seeking to standardize interconnect and make IP integration easier.

Image 1-4: AMBA system and peripheral buses, courtesy ARM



Then, ARM cores became fully synthesizable. The impetus was an effort at ASIC vendor LSI Logic who launched a CoreWare synthesizable version of ARM7TDMI in late 1997. ARM soon responded with standard synthesizable versions of its ARM7TDMI-S core and ARM946E-S and ARM966E-S macrocells, opening choices for using industry-standard EDA tools. By 2000, both TSMC and UMC had joined the new ARM Foundry Program and taken “per use” licenses.

Also in 2000, ARM made a strategic equity investment in CoWare and its IP models with an eye on providing more accurate simulations of the processor core. While useful, simulation was slow and models scarce, especially for many third-party peripheral IP blocks. Hardware emulation tools were prohibitively expensive for third-party peripheral block designers, often very small shops.

However, a bigger opportunity was developing. An IP block could be fully tested standalone, but when integrated into a larger system-on-chip with other peripheral blocks, new issues would develop. Either simulation would fail to uncover at-speed problems, or interaction between blocks would expose conditions untested in the standalone case.

The solution for affordable, faster, more complete testing of both IP blocks and integrated SoC designs was becoming FPGA-based prototyping. Both startups and larger EDA firms sought to capitalize on the trend.

NOTES

[15] "Reconfigurable Computing", Jeffrey M. Arnold, IDA Supercomputing Research Center, published in New Horizons of Computational Science: Proceedings of the International Symposium on Supercomputing, Tokyo, Japan, September 1997, edited by Toshikazu Ebisuzaki and Junichiro Makino, pp. 95-106.
[16] "The Roles of FPGAs in Reprogrammable Systems", Scott Hauck, Northwestern University, published in Proceedings of the IEEE, Volume 86 Issue 4, April 1998, pp. 615-638.
[17] "Automatic circuit partitioning in the AnyBoard rapid prototyping system", Douglas A. Thomae and David E. Van den Bout, North Carolina State University, Microprocessors and Microsystems, Volume 16 Issue 6, 1992, pp. 283-290.
[18] "Chips on the Net: An FPGA prototyping platform", M. J. Smith and H. Fallside, University of Hawaii and Xilinx, published in Proceedings of the 3rd European Workshop on Microelectronics Education, May 2000, pp. 151-154.
[19] "BORG: A Reconfigurable Prototyping Board Using Field-Programmable Gate Arrays", Pak K. Chan, Martine D. F. Schlag, and Marcelo Martin, University of California, Santa Cruz, published in Proceedings of the 1st International ACM/SIGDA Workshop on Field-Programmable Gate Arrays, 1992, pp. 47-51.
[20] "FPGA Based Low Cost Generic Reusable Module for the Rapid Prototyping of Subsystems", Apostolos Dollas, Brent Ward, John D. S. Babcock, Duke University, published in Lecture Notes in Computer Science, Volume 849, 1994, pp. 259-270.
[21] "The Transmogrifier: The University of Toronto Field-Programmable System", Galloway et al, University of Toronto, June 1994, http://www.eecg.toronto.edu/~jayar/research/Transmogrifier1.pdf
[22] "Aptix aims to be 'system-emulation' pioneer", Richard Goering, Techweb, CMP Publications, June 27, 1994, p. 33, http://www.xsim.com/bib/papers.d/aptix.html
[23] "Aptix expands System Explorer family of emulation tools", Aptix press release, May 20, 1996, http://www.thefreelibrary.com/APTIX+EXPANDS+SYSTEM+EXPLORER+FAMILY+OF+SYSTEM+EMULATION+TOOLS-a018303369
[24] "Logic Emulation for the Masses Arrives; IKOS prepares to roll out production versions of its innovative VirtuaLogic SLI emulation

solution", IKOS Systems press release, September 16, 1996, http://www.thefreelibrary.com/Logic+Emulation+for+the+Masses+Arrives%3B+IKOS+prepares+to+roll+out...-a018675799

[25] "Too much preoccupation with patents?", Russ Arensman, embedded.com, January 1, 2003, http://www.embedded.com/print/4347248

[26] Gidel web site, http://www.gidel.com/ASIC-prototyping/index.asp

[27] The Dini Group corporate presentation, http://www.dinigroup.com/files/FPGA-based-Cluster%20computing_9-09-10-HPC.pdf

[28] "Mobile Unleashed", Daniel Nenni and Don Dingee, SemiWiki, December 2015.

Chapter 2: How S2C Stacked Up Success

Startups are often pure-plays focused on a particular technology. When a new market opportunity appears, it is often startups who are able to move in first. Expertise gained in a previous round of technology development can propel a startup, with a new brand and strategy, from obscurity into prominence.

FPGA-based prototyping systems presented just such an opportunity. The groundswell in the IP ecosystem and the addition of foundry players in Asia brought a new audience of SoC designers into the mix. A wave of Asian firms, or Asian design centers for companies based in the US and Europe, were among the first ones open to new ideas and new EDA tools from a new innovator.

Making ESL Mean Something

Aptix had delivered outstanding FPGA interconnect technology, but as its legal issues deepened, it lost focus and became unable to compete for new business. By 2003 its key talent was defecting, ready to take the lessons learned from the reconfigurable computing days elsewhere.

Three ex-Aptix principals, Thomas Huang, Mon-Ren Chene, and Toshio Nakama pooled their own money to form S2C, Inc. – a creative spin on "system to chip". Based in San Jose, California, the vision for S2C was helping accelerate time-to-success for SoC design companies. To do that, S2C would need to build a new organization, and carefully craft and document their intellectual property to avoid the quagmire their previous engagement became mired in.

Focusing on the Chinese market as a major opportunity, S2C quickly set up its first offshore research and development center in Shanghai in 2003. This not only provided a talent pool, but also offered a way to connect and service customers based in Asia. Since the methodology behind FPGA-based prototyping was fairly new, and design teams using it were often working on small- or medium-sized projects, there would be a fair amount of customer handholding required.

S2C was essentially in stealth mode for nearly two years. One of the key problems in making the leap from reconfigurable computing to FPGA-based prototyping was the tools used with FPGAs and the IP that went inside.

The buzzword making the rounds in EDA circles at the time was ESL: *electronic system-level* design. The idea of ESL sounded great on paper, bringing together approaches using high-level hardware descriptions with hardware and software co-design strategies and adding virtual prototyping and co-verification. (It was a lot of buzzwords inside buzzwords.) EDA vendors were scrambling to unify their tool suites and create a cohesive flow that shared design data and results.

FPGA tools were different, even foreign to most ASIC designers. For many, although the benefits of prototyping were increasing, the extra steps in becoming familiar with FPGA synthesis and debug were troubling. Worse yet, FPGA IP blocks were usually tuned for FPGA constructs. Steps to obtain logic and timing closure in FPGAs were different from those in ASICs. Concerns over the fidelity of an FPGA-based prototype were valid; if too much effort was required to resolve differences when moving a design back into an ASIC flow, the time-to-success gains from prototyping and exploration would be undone.

S2C's first task was to develop a complete methodology – a set of tools and IP that would not only make FPGA-based prototyping more productive, but would smooth out the transition of a design from the prototyping stage back into an EDA flow bound for an SoC.

TAI IP and "Prototype Ready"

In February 2005, S2C filed its patent for a "Scalable reconfigurable prototyping system and method." It described a system for automating validation tasks for SoCs, with a user workstation, data communication interface, and an emulation platform with multiple FPGAs plus interfaces to a real-world target system. Its key observation was that SoC designs were composed of multiple IP blocks, and those could be synthesized from HDL into FPGAs, but needed some standardized method to communicate with a host for download, debug, and modification. [29]

A few months later, they gave a name to the concept: Testable, Analyzable, and Integratable IP, or TAI IP for short. A TAI IP block contained dynamically reconfigurable tag memory, multiplexers, buffers and latches, and could be added to user IP blocks to allow control and incremental run-time configuration. Both debug and performance analysis modes existed; for instance, a bandwidth analyzer mode could gather information on throughput and latency. [30]

This allowed S2C to deliver a cohesive design flow. A top-level design including imported modules from HDL or netlists could be synthesized, assigned symbols, partitioned into FPGAs, automatically instrumented with TAI IP blocks, and placed and routed. Software development could begin as soon as a prototype was ready, including using the SoC prototype to communicate with a target interface at hardware speeds – a nearly impossible task using only host-based simulation.

In May 2005, S2C announced its first product at the Design Automation Conference (DAC). The IP Porter system supported SoC designs of up to 3M gates with four Xilinx XC2VP100 FPGAs. It connected to a host via USB 2.0, where the TAI Compiler and Navigator software packages ran. TAI Compiler automated creation of TAI IP modules and libraries, including encryption if desired. Navigator provided links to System C models and SCE-MI transactors connecting event-driven simulation tools on the host with points in the FPGA-based prototype.

Beta customers working with the product estimated their design time was cut by 3 to 6 months. Productivity gains came not so much from the initial setup of a prototype, but from a significant reduction in iteration time as debug and analysis uncovered changes and improvements. TAI IP enabled reconfiguring only the part of the design where changes were made, speeding up the synthesis.

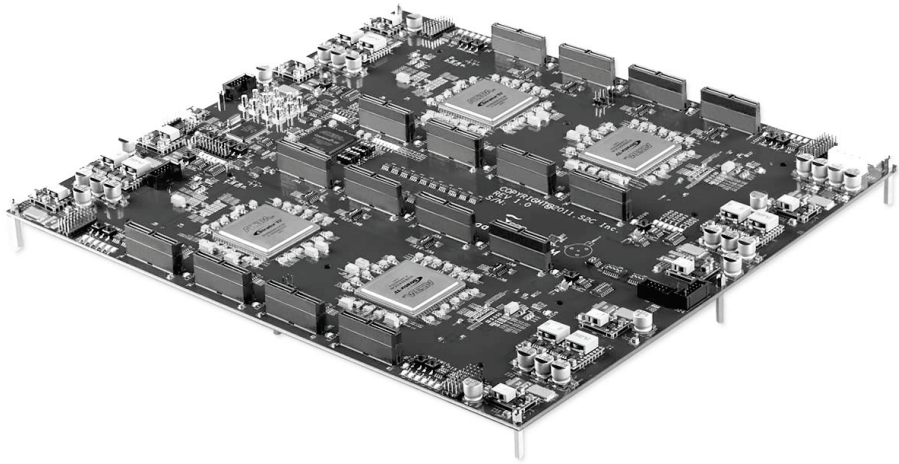
Image 2-1: S2C IP Porter

Second-generation product beginning in 2006 moved to a new hardware arrangement with high-performance Mictor connectors allowing stacking of TAI logic module boards each with one or two Xilinx Virtex-4 FPGAs. These connectors also enabled direct addition of a library of expansion modules for memory, video and audio interfaces, logic analyzer breakouts, and more, and allowed customer I/O designs to be added easily.

2007 brought new IP partners, most notably processor core vendors Tensilica and CAST. Work continued on SCE-MI with Hitachi and other co-modeling projects with Asian customers, and a new office opened in Shenzhen to support the growing SoC community in China. In 2008, the TAI IP patent was granted in the US, and the TAI logic modules received the latest Xilinx Virtex-5 FPGAs. Host software was unified under the TAI Player name, with expanded logic analysis and SCE-MI co-modeling. [31]

By 2010, the fourth generation logic module based on dual Xilinx Virtex-6 FPGAs appeared, along with a first – support for Altera FPGAs with a dual Stratix IV Logic module, unique among the major FPGA-based prototyping vendors. Design sizes accommodated on a single module topped 15M gates, and module stacking pushed that higher. More modules were added to the accessory library, including PCIe and Gigabit Ethernet. Larger Altera-based systems released in 2011, with the Quad S4 TAI Logic Module carrying four Altera Stratix IV 820 FPGAs upping the capacity to 32.8M gates. [32, 33, 34]

Image 2-2: S2C Quad S4 TAI Logic Module

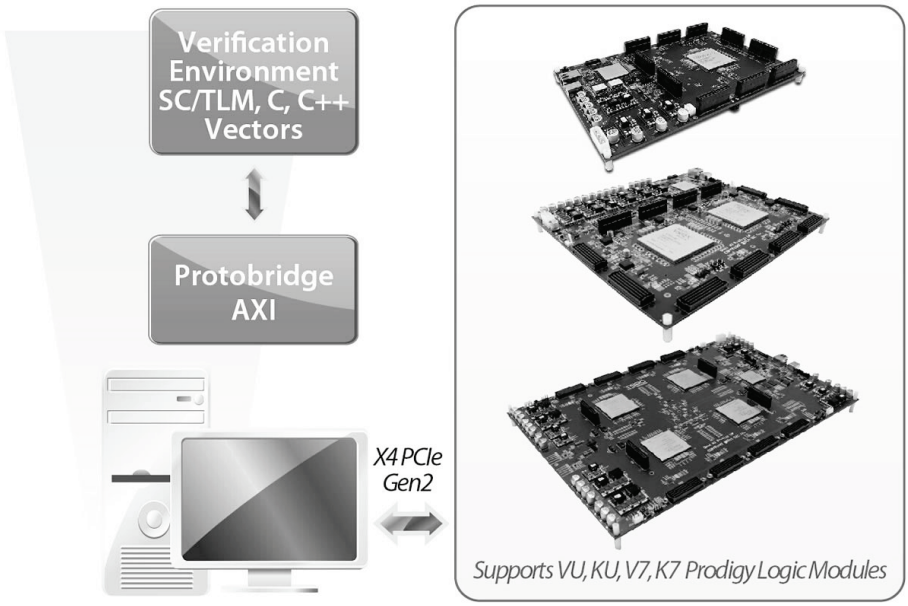


To handle larger designs and faster verification, S2C also announced a x4 PCIe Gen2 host interface board in 2011, and continued expansion of its Prototype Ready accessory library. Xilinx Virtex-7 logic modules released in 2012, along with ARM1176 and ARM926 Global Unichip Corporation (GUC) Test Chip modules with external AMBA interfacing for ARM designers to quickly incorporate merchant cores. [35, 36, 37]

2013 saw the addition of more logic modules based on the Xilinx Zynq-7000 All Programmable SoC with its integrated dual ARM Cortex-A9 cores. With up to four Zynq-7000 parts on one board, plus a high-frequency LVDS pin multiplexing scheme, a single board could handle up to 80M gates. New Prototype Ready modules added HDMI, GTX transceiver interfacing, and other support around the Zynq-7000.

ProtoBridge AXI capability came in 2014, providing native AXI transactors suitable for advanced ARM designs and other IP adopting AXI. [38, 39]

Image 2-3: S2C ProtoBridge software environment

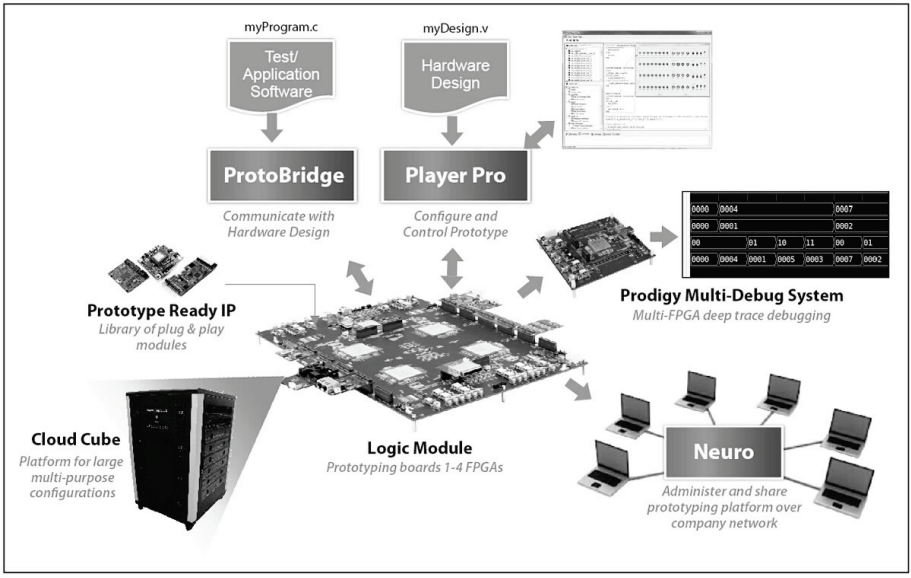


Taking on the Cloud

S2C continues its expansion under founders Nakama, now CEO, and Chene, now Chairman and CTO. In 2014, new offices opened in Japan and Korea, and a round of $4.6M Series C financing reinforces R&D and sales and support channels development. Keeping pace with the Xilinx roadmap for larger and faster parts, including the latest Virtex UltraScale 440 FPGA, is just part of the strategy.

In April 2015, S2C announced a new brand: Prodigy. In many ways, the story remains the same under a new name. Prodigy unifies the offering of FPGA logic modules, Player Pro partitioning and configuration software, ProtoBridge system-level simulation link tool, and the library of Prototype Ready modules now numbering over 80 designs. In other ways, Prodigy marks a new beginning. [40]

Image 2-4: S2C Prodigy Complete Prototyping



For DAC 2015 in June, ten years after introducing its first product to the public, S2C unveiled a new concept in scalability. Prodigy Cloud Cube introduces a capacity breakthrough providing up to 1.4B gates in a single chassis with up to 32 FPGAs. Simultaneous access for up to 16 engineers is supported, with remote access via Ethernet. Configuration of the platform itself is automatic, with detection of installed logic modules, cabling, and daughter cards, along with self-tests to isolate issues. [41]

S2C's focus on Asia continues to increase, with a new R&D and manufacturing center in Taiwan opened in October 2015. With energy in mobile shifting from flagship smartphones into mid-range devices, and new ideas appearing daily in wearables and the IoT, more designs are starting with a wider variety of hardware IP and software support. Asia is at the epicenter of many of these changes, increasing the need for a distributed FPGA-prototyping solution made for design teams and IP providers to collaborate from wherever they happen to be based.

Scalability also remains important. Single FPGA logic module solutions are now large enough to hold many designs. The ability to use the same

tool set and FPGA-based prototyping methodology from the latest Single KU115 Prodigy Logic Module introduced in January 2016 all the way up to the Prodigy Cloud Cube is essential for SoC design team productivity. [42]

Image 2-5: S2C Cloud Cube 32

Now serving over 200 customers, S2C has delivered innovation by staying close to its users. Extra handholding required in the first generation has now turned into a competitive advantage in later generations. By looking closely at each step in the SoC design phase, enhancements in S2C FPGA-based prototyping tools are discovered, removing friction and increasing capability and efficiency.

NOTES

[29] "Scalable reconfigurable prototyping system and method", US Patent 7353162, April 1, 2008, http://www.google.com/patents/US7353162
[30] "S2C delivers breakthrough FPGA-based ESL Design with TAI IP", Design & Reuse, May 31, 2005, http://www.design-reuse.com/news/10519/s2c-breakthrough-fpga-esl-design-tai-ip.html
[31] "A New Era for SoC Prototyping Flow: From System Architecture Plan to Verification", S2C presentation by Mon-Ren Chene, January 15, 2009, http://www.digitimes.com.tw/tw/B2B/Seminar/Service/download/0519801150/DTF_980115_Track1_04.pdf
[32] "S2C Announces 4th Generation Rapid SoC Prototyping Solution", S2C press release, June 14, 2010, http://www.s2cinc.com/company/press-releases/2010/s2c-announces-4th-generation-rapid-soc-prototyping-solution
[33] "S2C Announces Virtex-6 Based 4th Generation Rapid SoC Prototyping Solution", S2C press release, August 30, 2010, http://www.s2cinc.com/company/press-releases/2010/s2c-announces-virtex-6-based-4th-generation-rapid-soc-prototyping-solution
[34] "S2C Releases 32.8 Million Gate SoC/ASIC Prototyping System", S2C press release, April 21, 2011, http://www.s2cinc.com/company/press-releases/2011/s2c-releases-32.8-million-gate-socasic-prototyping-system
[35] "S2C Announces a Breakthrough Verification Module", S2C press release, June 6, 2011, http://www.s2cinc.com/company/press-releases/2011/s2c-announces-a-breakthrough-verification-module
[36] "S2C Releases Dual Virtex-7 2000T FPGA Rapid SoC Prototyping Hardware", S2C press release, May 31, 2012, http://www.s2cinc.com/company/press-releases/2012/s2c-releases-dual-virtex-7-2000t-fpga-rapid-soc-prototyping-hardware
[37] "S2C Releases New Prototype Ready ARM11 and ARM9 Modules for FPGA-Based Prototypes", S2C press release, June 1, 2012, http://www.s2cinc.com/company/press-releases/2012/s2c-releases-new-prototype-ready%E2%84%A2-arm11-and-arm9-modules-for-fpga-based-prototypes
[38] "New Quad Virtex-7 2000T 3D IC Rapid ASIC Prototyping Platform from S2C Optimized for Design Partitioning", S2C press release, January 21, 2013, http://www.s2cinc.com/company/press-

releases/2013/new-quad-virtex-7-2000t-3d-ic-rapid-asic-prototyping-platform-from-s2c-optimized-for-design-partitioning

[39] "S2C ProtoBridge AXI Expands FPGA-Based Prototype Usage", S2C press release, June 30, 2014, http://www.s2cinc.com/company/press-releases/2014/s2c-protobridge%E2%84%A2-axi-expands-fpga-based-prototype-usage

[40] "S2C Sets New Standards for FPGA-Based Prototyping with Prodigy Complete Prototyping Platform", S2C press release, April 21, 2015, http://www.s2cinc.com/company/press-releases/2015/s2c-sets-new-standards-for-fpga-based-prototyping-with-prodigy-complete-prototyping-platform

[41] "S2C Prodigy Cloud Cube Enables FPGA Prototyping of 1 Billion Gate Designs", S2C press release, May 26, 2015, http://www.s2cinc.com/company/press-releases/2015/s2c-prodigy%E2%84%A2-cloud-cube%E2%84%A2-enables-fpga-prototyping-of-1-billion-gates-designs

[42] "S2C Expands Kintex UltraScale Prototyping Solutions for Consumer-based IoT and Other Small to Medium Sized Designs", S2C press release, January 11, 2016, http://www.s2cinc.com/company/press-releases/20161/s2c-expands-kintex-ultrascale-prototyping-solutions-for-consumer-based-iot-and-other-small-to-medium-sized-designs

Chapter 3: Big EDA Moves In

Larger EDA firms are constantly hunting for other product lines to fill out their portfolios. With ASIC and SoC starts accelerating, and smaller firms becoming successful with FPGA-based prototyping technology, the large firms started seeing a gap in their offering. The trick in a large firm acquiring a small firm is to make those focused tools fit in the bigger picture.

In the case of FPGA-based prototyping, first came an acquisition of one of its pioneers by another FPGA tools vendor, which was then in turn swallowed up by a large EDA firm. Those product lines were rebranded, then expanded. When a comprehensive strategy for FPGA-based prototyping systems was published, it drew an immediate a response from a major competitor. As the need grows, competition is heating up.

A Laurel and HARDI Handshake

In 1987, the IEEE ratified the initial version of its standard IEEE 1076-1987, VHSIC Hardware Description Language. EDA firms rushed to embrace VHDL technology, both to satisfy its major backer – the US Department of Defense and in particular the US Air Force – and to capture the benefits of a high-level language for design and simulation of ASICs. [43]

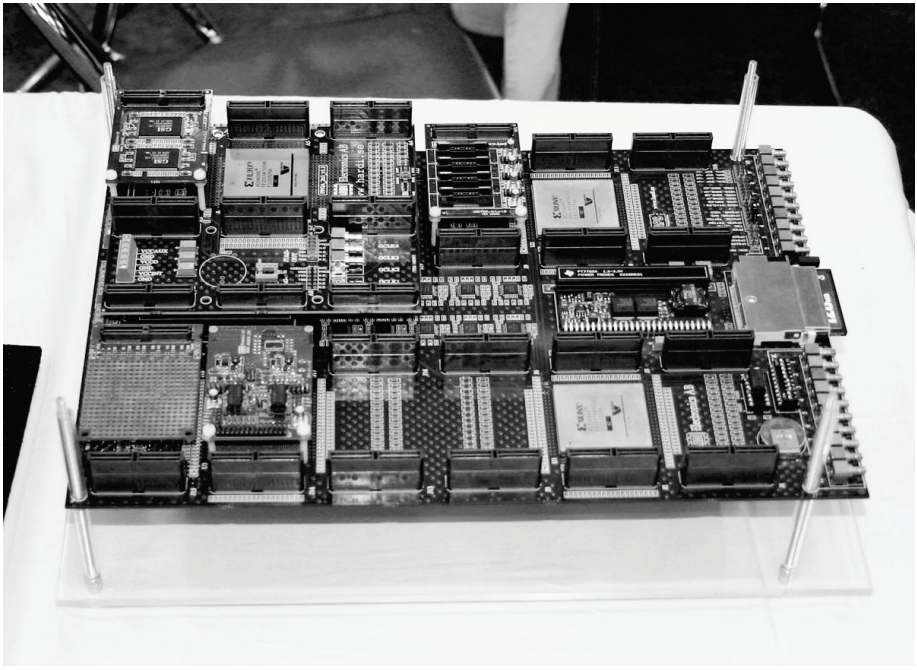
Consultants in VHDL instantly sprang up. One of those firms was HARDI Electronics AB, launched shortly after the original release of the IEEE standard in 1987. HARDI quickly produced its first all-VHDL design within a few months. They went on to develop extensive expertise, publishing its VHDL Handbook in 1997 reflecting the updated VHDL '93 version of the standard. [44]

As VHDL usage grew, so did the size of ASIC designs performed using it. Simulation, though effective, was falling behind in terms of providing enough speed to run the necessary verification tests on a larger ASIC. Moving the verification tasks into hardware became the path forward.

In 2000, HARDI took the next logical step, creating the HARDI ASIC Prototyping System. For several years, the system then just known as HAPS was used in consulting activity for ASIC customer engagements. HARDI coordinated closely with both Xilinx and a relatively new firm formed in 1994, Synplicity, for FPGA synthesis and debug technology including Synplicity's Certify and Identify.

Demand for the HAPS platform rose over the next several years to the point where HARDI began more aggressive external marketing, launching version 2.1 of HAPS (soon to be rebranded as HAPS-10) at the Design Automation and Test in Europe show in March 2003. HAPS 2.1 held up to four Xilinx Virtex-II 8000 FPGAs providing a total of up to 8M gate capacity running at up to 200 MHz. [45]

Image 3-1: HARDI Electronics AB HAPS-FPGA_2x3



HARDI launched its product in the US at the Design Automation Conference in June 2003, and soon found a major customer in Texas Instruments. A low end version, the single FPGA HAPS-FPGA_2x3 accommodating up to 1M gates, introduced the idea of stackable

prototyping modules – with an early version of the HapsTrak interconnect – for scalable capacity and expansion. HAPS-FPGA_2x3 also had 120 low voltage differential signaling (LVDS) pairs for high-speed I/O. [46]

The second-generation HAPS-20 debuted in December 2004 with a quad Xilinx Virtex-II Pro configuration, again stackable via HapsTrak. It focused on I/O speed, providing 80 multi-gigabit serial links and 1600 LVDS pairs. The Virtex-II Pro FPGAs also each carried two IBM PowerPC RISC processor cores for real-time software. User I/O was divided into three voltage regions so separate voltages could be used simultaneously. HAPS-20 added built-in self-test capability to assure users of system integrity. [47]

Xilinx Virtex-4 parts appeared on the third-generation HAPS-34 at the ARM Developers Conference in October 2005, a sign that adoption of FPGA-based prototyping among SoC designers was accelerating. HAPS-34 delivered quad FPGAs with 9 I/O voltage regions. Smaller versions quickly appeared, with the single HAPS-31 and the dual HAPS-32 added in March 2006, all based on HapsTrak for stacking "like LEGO™ blocks" as the HARDI PR team put it. [48, 49, 50]

Verification is Very Valuable

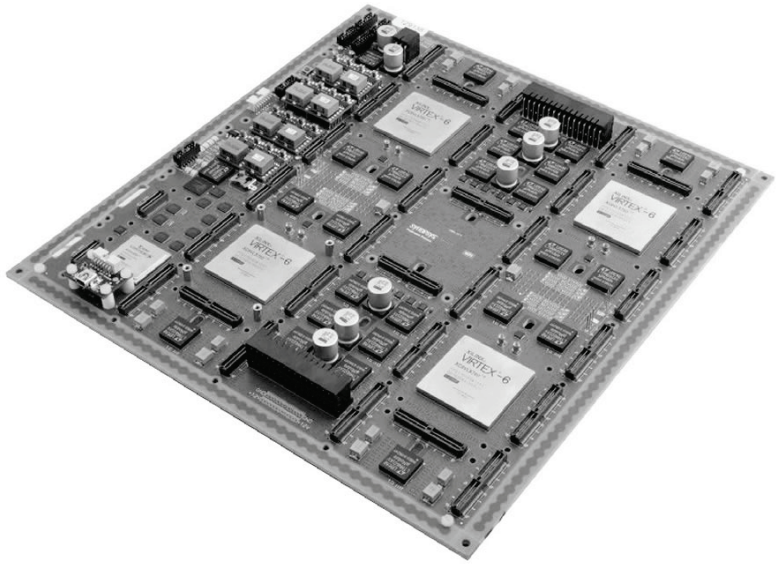
For the fourth generation, HARDI made a break with their logical nomenclature, instead skipping to the HAPS-50 series with an introduction in April 2007. The HAPS-52 featured a pair of Xilinx Virtex-5 LX330 FPGAs. It was basically more of the same approach; however, the quad board was not yet available, with only a reference to more boards available in two months. [51]

Almost two months to the day later came a much more surprising announcement. The HAPS-50 news included a quote on the growing partnership between HARDI and Synplicity, including the addition of the new Total Recall debugging technology. The partnership cemented on June 1, 2007 with the news that Synplicity was acquiring HARDI for $24.2M in cash. [52]

In turn, the new and improved Synplicity was suddenly on radar of one of the big three EDA firms: Synopsys. In a slightly more complicated transaction since Synplicity was publicly traded, Synopsys paid around $227M to acquire Synplicity on March 20, 2008. HAPS was now a Synopsys brand. [53]

Synopsys took a breath to integrate HAPS into their development flow, creating the Confirma Rapid Prototyping Platform including the Synplicity suite and CHIPit technologies acquired from Pro Design in 2008. Syncing up with Xilinx for the Virtex-6 FPGA, Synopsys released the HAPS-60 in April 2010 with a capacity of up to 18M gates. Synopsys began using the HAPS environment for their own DesignWare IP, and was able to pass through those artifacts to customers. They also added support for the UMRbus, a high-speed host interface allowing co-simulation with a HAPS platform. [54]

Image 3-2: Synopsys HAPS-64



HAPS-70 appeared in November 2012 with the Xilinx Virtex-7 2000T on nine model variants. An upgrade to HapsTrak 3 improved time-domain pin multiplexing, and the Certify software became "HAPS-aware" understanding partitioning and interconnect needs of the hardware, resulting in a 10x productivity improvement. Deep Trace

Debug was added with a DRAM module to capture more real-time signals with complex triggering. The UMRbus was also upgraded to support up to 400 MB/sec transfers. [55]

Capacity had long been an objective of HAPS, keeping pace with each successive Xilinx FPGA release. The efforts with DesignWare IP and customers showed how valuable smaller FPGA-prototyping platforms could be, easy to set up for a software developer to work on code or for an IP block developer to work on a single piece of IP prior to integration. Streamlining the larger HAPS-70 platform resulted in the HAPS Developer eXpress, or HAPS-DX, in December 2013. HAPS-DX added an FMC interface for industry-standard daughterboards to add I/O, and ProtoCompiler (formally released in April 2014, replacing the short-lived Confirma tools) extending the flow and hardware awareness in software tools. [56, 57]

An Either-Or Response

In March 2011 after working in conjunction with Xilinx, Synopsys released the FPGA-Based Prototyping Methodology Manual. Mentor Graphics was (and is) still focused on hardware emulation technology. The third member of the big three, Cadence, was also in hardware emulation but needed some kind of response to the growing FPGA-based prototyping movement.

Cadence had focused on design of FPGA boards themselves. After a multi-year technology agreement proved the concept, Cadence decided to acquire Taray and their FPGA design-in solution in March 2010. Taray pioneered route-aware pin assignment synthesis, optimizing an FPGA design together with the circuit board. This would form an aid for designing an FPGA-based prototyping platform. [58]

The formal response to Synopsys came in a Cadence white paper, “ASIC Prototyping Simplified”, in April 2011. In it, they questioned the approach of an off-the-shelf FPGA board to meet a wide variety of ASIC prototyping needs. They suggested instead that customized FPGA boards be built matching the needs of each project, of course using Cadence tools. [59]

It was a time-buying maneuver, just to get the industry talking. A month later in May 2011, Cadence announced a strategy for "app-driven" electronics within its bigger EDA360 vision going beyond just co-verification. Part of that was a new Rapid Prototyping Platform, a family of FPGA boards based on Altera Stratix IV devices. Cadence was attempting to unify its tools and flows, so that a customer could choose either hardware emulation or FPGA-based prototyping and migrate back and forth as needed. [60]

Image 3-3: Cadence Protium

A Bright Future Ahead

Cadence has a lot of ground to make up in FPGA-based prototyping, but is making investments to try to do just that. In July 2014, they introduced the second-generation Protium rapid prototyping platform, curiously moving to Xilinx Virtex-7 2000T FPGAs matching the competition and increasing capacity by 4x over the first-generation product. The bring-up flow between Palladium hardware emulation and Protium FPGA-based prototyping was further refined, with productivity gains for customers using both environments. [61]

Synopsys has stayed its course as one of the leaders in the field. Moving into the Xilinx UltraScale generation, HAPS-80 launched in September 2015. Synopsys says this gets them to 1.6B ASIC gates with stacked HAPS-80 modules, supported with an improved ProtoCompiler handling the high-speed time-division multiplexing awareness. HAPS-80 runs at 300 MHz for a single FPGA, 100 MHz with non-pin-multiplexed multi-FPGAs, and 30 MHz with pin multiplexing.

Image 3-4: Synopsys HAPS-80

Combined with the efforts of S2C and others including Aldec, The Dini Group, HyperSilicon, Pro Design Electronic GmbH, ReFLEX, and even small prototyping systems from ARM, these developments have moved the art of FPGA-based prototyping systems forward. Next, we'll take a look at where the technology is headed and how designers in segments enjoying a renaissance of design starts can benefit from the ideas.

NOTES

[43] "1076-2008 – IEEE Standard VHDL Language Reference Manual", IEEE Standards Association, https://standards.ieee.org/findstds/standard/1076-2008.html
[44] "VHDL Handbook", HARDI Electronics AB, 1997, http://www.csee.umbc.edu/portal/help/VHDL/VHDL-Handbook.pdf
[45] "HARDI Electronics Releases a Real-Time ASIC Prototyping Platform at DATE", HARDI Electronics AB press release, March 3, 2003, http://www.businesswire.com/news/home/20030303005294/en/HARDI-Electronics-Releases-Real-Time-ASIC-Prototyping-Platform
[46] "HARDI Electronics Releases a New Single-FPGA Module in the HAPS Prototyping Family", HARDI Electronics AB press release, December 2, 2003, http://www.businesswire.com/news/home/20031202005773/en/HARDI-Electronics-Releases-Single-FPGA-Module-HAPS-Prototyping
[47] "HARDI Electronics Unveils Second Generation ASIC Prototyping Platform", HARDI Electronics AB press release, December 13, 2004, http://www.businesswire.com/news/home/20041213005828/en/HARDI-Electronics-Unveils-Generation-ASIC-Prototyping-Platform
[48] "HARDI Electronics Unveils Industry's Most Advanced ASIC Prototyping Platform at the ARM Developers Conference", HARDI Electronics AB press release, October 4, 2005, http://www.businesswire.com/news/home/20051004005734/en/HARDI-Electronics-Unveils-Industrys-Advanced-ASIC-Prototyping
[49] "How to Make an ASIC Prototype", Lars-Eric Lundgren, HARDI Electronics AB, Electronic Engineering Journal October 18, 2005, http://www.eejournal.com/archives/articles/20051018_hardi/
[50] "HARDI Electronics Announces Two New Motherboards in The HAPS ASIC Prototyping Family at DATE 2006 (Booth A1)", HARDI Electronics AB press release, March 6, 2006, http://www.businesswire.com/news/home/20060306005867/en/HARDI-Electronics-Announces-Motherboards-HAPS-ASIC-Prototyping
[51] "HARDI announces FPGA-based HAPS-50 prototyping system", Max Maxfield, EETimes, April 4, 2007, http://www.eetimes.com/document.asp?doc_id=1304157
[52] "Synplicity Announces Agreement to Acquire HARDI Electronics AB", Synplicity press release, June 1, 2007,

http://www.sec.gov/Archives/edgar/data/1027362/000119312507129831/dex991.htm

[53] "Synopsys to Acquire Synplicity, Inc.", Synopsys press release, March 20, 2008, http://news.synopsys.com/index.php?item=122910

[54] "Synopsys Introduces the HAPS-60 Series of Rapid Prototyping Systems", Synopsys press release, April 19, 2010, http://news.synopsys.com/index.php?s=20295&item=123150

[55] "New FPGA-Based Prototyping Solution Delivers Up to 3x System Performance Improvement", Synopsys press release, November 12, 2012, http://news.synopsys.com/index.php?s=20295&item=123433

[56] "Synopsys Extends HAPS-70 Prototyping Family with New Solution Optimized for IP and Subsystems", Synopsys press release, December 16, 2013, http://news.synopsys.com/2013-12-16-Synopsys-Extends-HAPS-70-Prototyping-Family-with-New-Solution-Optimized-for-IP-and-Subsystems

[57] "Synopsys' New ProtoCompiler Software Speeds Time to First Prototype by Up to 3X", Synopsys press release, April 23, 2014, http://news.synopsys.com/2014-04-23-Synopsys-New-ProtoCompiler-Software-Speeds-Time-to-First-Prototype-by-Up-to-3X

[58] "Cadence Strengthens Leadership in FPGA Design-In Solutions with Acquisition of Taray", Cadence Design Systems press release, March 22, 2010, http://www.cadence.com/cadence/newsroom/features/pages/feature.aspx?xml=taray

[59] "ASIC Prototyping Simplified", Cadence Design Systems white paper, April 2011, http://www.cadence.com/rl/Resources/technical_papers/asic_prototyping_tp.pdf

[60] "Cadence Announces Breakthrough in System Development to Meet Demands of 'App-driven' Electronics", Cadence Design Systems press release, May 3, 2011, http://www.cadence.com/cadence/newsroom/press_releases/pages/pr.aspx?xml=050311_sys_dev

[61] "Cadence Announces Protium Rapid Prototyping Platform and Expands System Development Suite Low-Power Verification", Cadence Design Systems press release, July 17, 2014, http://www.cadence.com/cadence/newsroom/press_releases/Pages/pr.aspx?xml=071714_Protium

Chapter 4: Strategies for Today and Tomorrow

SoC design has gone from relatively simple parts with a handful of IP blocks to massive designs with around 150 IP blocks. Reuse is becoming more important than ever, and integration is still where many designs meet with challenges. With schedules under pressure and verification and validation needs urgent, design teams are bringing a combination of EDA tools to work.

As the cost of an SoC has escalated, the need for pre-silicon exploration has increased. Tradeoffs in performance and power consumption are part of nearly every design, especially mobile devices where recharging factors heavily into user experience. Expanding software content must be co-verified, with testing beginning long before production silicon is available. Complex workloads present an opportunity for optimization at the system level, if understood.

The State of FPGA-Based Prototyping

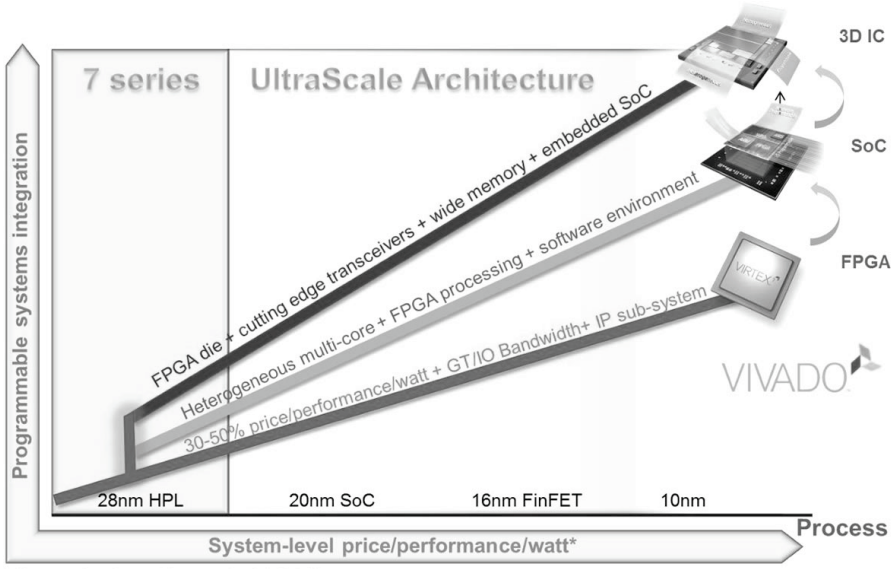
More and more teams are deploying FPGA-based prototyping tools today. The technology has adapted as FPGAs themselves and system-level hardware and software have improved, and use cases within the SoC design flow have clarified.

Changes in the FPGA itself have been dramatic. Foundries quickly discovered the uniform structures of FPGAs were ideal to prove out new process nodes. For example, first-generation Xilinx UltraScale FPGA parts use 20nm technology to pack something around 20 billion transistors in a part. This has driven single FPGA equivalent gate counts into tens of millions, and I/O pins well into the thousands. I/O speed has also improved with advanced SERDES transceivers and better memory interfaces. Clocking and power domains present far more flexibility, allowing FPGAs to more accurately mimic ASIC-like constructs.

Diverse I/O interfaces and greater numbers of I/O pins motivate FPGA-based prototyping board designers. High-speed connectors provide flexibility with signal integrity. Common physical interfaces are now

available on standardized daughter cards, and custom interfaces can be designed and added rapidly using the same daughter card strategy – either by the FPGA-based prototyping system vendor, or by the customer. Debug interfaces have also improved, as have high speed host interconnect ports for downloading and managing code.

Image 4-1: Xilinx UltraScale process improvements



** System level combines unit with BOM cost*

FPGA-related software tools are also advancing. Logic synthesis tools efficiently translate high-level design into FPGA primitives, and understand nuances such as clock domain crossings and block RAM resources. Partitioning tools now not only divvy up larger designs across multiple FPGAs, they understand how the FPGA interconnect works on the FPGA-based prototyping board and optimize accordingly, reducing the need for manual partitioning. Debug tools provide visibility without intrusiveness. Co-simulation tools allow use of familiar simulation environments accelerated by execution in the FPGA hardware.

Researchers are still considering the problem of multi-FPGA interconnect, as I/O pins continue to be a limiting factor and partitioning is challenging as inter-FPGA delays are still present. A new

thesis looks at the various approaches and considers a new congestion-aware routing algorithm exploiting multi-point tracks and if flexible cabling distribution eases the solution. The author confirms that given automated flows, even complex multi-FPGA partitioning can be reduced to a matter of several hours instead of days or weeks – a huge productivity gain. It remains to be seen if the multi-point track approach, essentially using an interim FPGA hop between two other FPGA destinations, becomes a new best practice in automated synthesis. [62]

Image 4-2: Example of multi-point tracks in FPGA routing, courtesy Qingshan Tang, Pierre and Marie Curie University (UPMC)

(b) Existing routing algorithm (c) Proposed routing algorithm

The biggest changes may be in the design workflow. Design teams are often geographically distributed, and access to a lab-based system is impractical. Teams are also working together; a software developer may use a small FPGA-based prototyping system to exercise an IP block, then pass those results on to another team working on the fully integrated design on a larger FPGA-based prototype. Enterprise-class solutions are emerging, leveraging network connectivity and cloud resources to connect and manage multiple FPGA-based platforms. This reduces handoffs, improves scalability and reuse, and opens up access across the globe 24/7 in a flexible, yet secure environment.

With benefits of FPGA-based prototyping rising, adoption is steadily improving. The most recent 2014 data from Wilson Research Group places 32% of small projects up to 5M gates, 45% of medium projects up to 80M gates, and 28% of projects over 80M gates using FPGA-based prototyping. In the small category, adoption outpaces hardware emulation by nearly double, and in the medium category the two approaches are nearly equal in use. In the large category, FPGA-based

prototyping adoption is accelerating as capacities increase and ease of use improves, particularly where distributed teams and smaller IP blocks tested and then rolled up for integration are involved. [63]

Developing for ARM Architecture

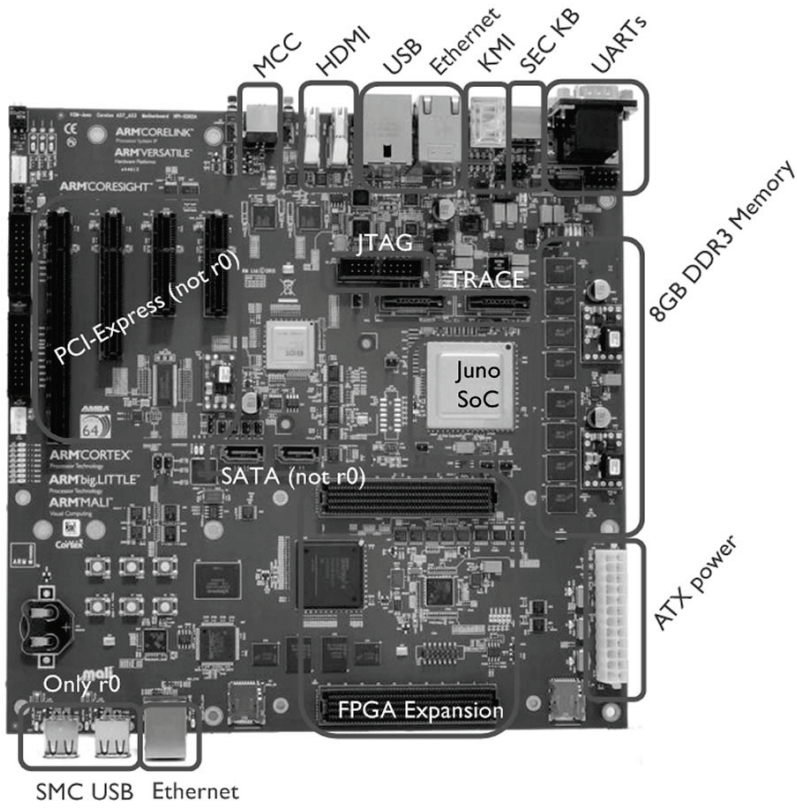
Since ARM introduced its Cortex strategy, with A cores for application processors, R cores for real-time processors, and M cores for microcontrollers, designers have been able to choose price/performance points – and migrate software between them. How do designers, who are often doing co-validation of SoC designs with production software, prototype with these cores?

Some teams elect to use ARM's hard macro IP offering, with optimized implementations of cores. ARM has a mixed prototyping solution with their CoreTile Express and LogicTile Express products. CoreTile Express versions are available for the Cortex-A5, Cortex-A7, Cortex-A9, and Cortex-A15 MPCore processors, based on a dedicated chip with the hardened core and test features. The LogicTile Express comes in versions with a single Xilinx Vertex-5, dual Virtex-6, or single Virtex-7 FPGAs, allowing loose coupling of peripheral IP. [64]

Others try to attack the challenge entirely in software. Cycle-accurate and instruction-accurate models of ARM IP exist, which can be run in a simulator testbench along with other IP. With growing designs come growing simulation complexity, and with complexity comes drastic increases in execution time or required compute resources. Simulation supports test vectors well, but is not very good at supporting production software testing – a large operating system can take practically forever to boot in a simulated environment.

Full-scale hardware emulation has the advantage of accommodating very large designs, but at substantial cost. ARM has increased its large design prototyping efforts with the Juno SoC for ARMv8-A, betting on enabling designers with a production software-ready environment with a relatively inexpensive development board.

Image 4-3: ARM Juno SoC Development Platform



However, as we have seen SoC design is rarely about just the processor core; other IP must be integrated and verified. Without a complete pass at the full chip design with the actual software, too much is left to chance in committing to silicon. While useful, these other platforms do not provide a cost-effective end-to-end solution for development and debug with distributed teams. Exploration capability in a prototyping environment is also extremely valuable, changing out design elements in a search for better performance, power consumption, third-party IP evaluation, or other tradeoffs.

The traditional knock on FPGA-based prototyping has been a lack of capacity and the hazards of partitioning, which introduces uncertainty and potential faults. With bigger FPGAs and synthesizable RTL versions of ARM core IP, many of the ARM core offerings now fit in a

single FPGA without partitioning. Larger members of the ARM Cortex-A core family have been successfully partitioned across several large FPGAs without extensive effort and adverse timing effects, running at speeds significantly higher than simulation but without the cost of full-scale hardware emulation.

A hybrid solution has emerged in programmable SoCs, typified by the Xilinx Zynq family. The Zynq UltraScale+ MPSoC has a quad-core ARM Cortex-A53 with a dual-core ARM Cortex-R5 and an ARM Mali-400MP GPU, plus a large complement of programmable logic and a full suite of I/O. If that is a similar configuration to the payload of the SoC under design, it may be extremely useful to jumpstart efforts and add peripheral IP as needed. If not, mimicking the target SoC design may be difficult. [65]

True FPGA-based prototyping platforms offer a combination of flexibility, allowing any ARM core plus peripheral IP payload, and debug capability. Advanced FPGA synthesis tools provide platform-aware partitioning, automating much of the process, and are able to deal with RTL and packaged IP such as encrypted blocks. Debug features such as deep trace and multi-FPGA visibility and correlation speed the process of finding issues.

The latest FPGA-based prototyping technology adds co-simulation, using a chip-level interconnect such as AXI to download and control joint operations between a host-based simulator and the hardware-based logic execution. This considerably increases the speed of a traditional simulation and allows use of a variety of host-based verification tools. Using co-simulation allows faster turnaround and more extensive exploration of designs, with greater certainty in the implementation running in hardware.

Integration rollup is also an advantage of scalable FPGA-based prototyping systems. Smaller units can reside on the desk of a software engineer or IP block designer, allowing dedicated and thorough investigation. Larger units can support integration of multiple blocks or the entire SoC design. With the same synthesis, debug, and visualization tools, artifacts are reused from the lower level designs,

speeding testing of the integrated solution and shortening the time-to-success.

Another consideration in ARM design is not all cores are stock. In many cases, hardware IP is designed using an architectural license, customized to fit specific needs. In these cases, FPGA-based prototyping is ideal to quickly experiment and modify designs, which may undergo many iterations. Turnaround time becomes very important and is a large productivity advantage for FPGA-based prototyping.

Adoption Among Major System Houses

Perhaps the most striking examples of the usefulness of FPGA-based prototyping strategies are its use in flagship mobile and networking SoC designs. These sophisticated design teams are pursuing massive designs with a phalanx of EDA tools, customizing design flow to meet specific needs.

Image 4-4: Apple A9 chip, courtesy AnandTech and iFixit

At Apple, where ARM-based SoC designs have been in progress even before they signed an architectural license in 2008, highly optimized chips are co-verified with iOS producing stunning designs for the iPhone and iPad families. FPGA-based prototyping systems are used in

conjunction with many technologies, some purchased commercially and some created in-house. A team of 30 engineers routinely use these systems, performing hardware and software exploration and integration test.

Samsung is a bit unique in that its distributed SoC teams work on designs handed off to Samsung foundry facilities and used in Samsung consumer electronics. When vertically integrated, passing along verification artifacts can be extremely beneficial. They are a big user of commercial FPGA-based prototyping platforms, including locally-designed FPGA boards from Korean vendors. Again, the design flow is highly customized, leveraging the flexibility of FPGA-based prototyping platforms for rapid turnaround and support of many configurations.

Huawei is another example of vertical integration, with even more self-designed FPGA platforms created for their networking infrastructure operations. As Huawei has entered the merchant SoC business with their HiSilicon brand, they have come under similar time-to-success pressure as other merchant vendors. They are adding commercial FPGA-prototyping systems in order to leverage more development tools rather than designing them all internally.

Application Segments in Need

High volume SoC applications in mobile devices and consumer electronics have been a proving ground for FPGA-based prototyping strategies. As PC and mobile markets mature, where is the frontier for SoC design? What new requirements make a strong case for use of FPGA-based prototyping?

Software content is growing at a rapid rate, now exceeding hardware effort in most projects. Co-verification is on the rise, where pre-silicon efforts explore production software long before committing a chip to production. Safety-critical needs are also rising, where both hardware and software must be validated to stringent requirements. These characteristics point toward three application segments, all on the rise.

Automotive electronics are undergoing a renaissance, after a period where many vendors avoided the harsh environmental requirements. Microcontroller content in cars has been increasing for decades, with

more points of control and interconnects such as CAN. Infotainment presented an opportunity for in-dash multimedia elements, very similar to those found in mobile SoCs.

Now, connectivity and intelligence are taking automotive electronics to the next level. Advanced driver assistance systems (ADAS) are on a rapid rise. Some estimates place as many as eight to ten cameras in each vehicle soon, with sophisticated embedded vision processing. Research in completely self-driving cars is also on the rise. New developments combine faster processing, improved interconnect such as one wire Ethernet, and cloud connectivity.

Modeling a car with its electromechanical systems is complex. Accuracy and speed is paramount, and use of production software is mandatory in achieving ISO 26262 requirements. Part of the compliance testing calls for fault injection to analyze potential failure modes – relatively easy with FPGA-based prototyping platforms and co-simulation, far more difficult around actual silicon.

Distributed development teams are also the norm in automotive. FPGA-based prototyping systems are often the only solution, providing virtual hardware for many developers at several locations instead of expensive production hardware. Using cloud technology, FPGA-based systems can be interconnected and accessed remotely, avoiding unnecessary duplication of platforms.

Another area getting a lot of attention is *wearable technology*. These devices fall into several broad categories: fitness bands, smartwatches, activity and health monitoring, fashion, vision-enhanced productivity, and more. Compute power and sensors vary, as does connectivity. Some devices are intended to be tethered to a smartphone, while others have their own 3G or 4G modem.

Two common requirements exist across wearables: small size and weight, and very low power consumption. In the Apple Watch, many chips were packed into the SI system-in-package (SiP), creating a highly optimized form factor. Consumer adoption is showing preference for wearable devices that operate for a week, not requiring recharging at the end of every day of use.

Using FPGA-based prototyping for wearables brings many benefits. In the area of small size and weight, exploration into packaging options can be performed, moving IP blocks and swapping pins until an optimum system-level solution is obtained. Tradeoffs between performance and power consumption can also be done, with visualization allowing designers to see what is going on with fine granularity and make changes in the design where necessary.

Finally, there is the *Internet of Things, or IoT* – not a single application segment, but more of a collection of interconnected technologies into a bigger idea. While not declared explicitly as safety-critical, many IoT applications provide important control functions and collect and process critical data that individuals and businesses are coming to rely on. IoT devices have to work correctly, all the time, and must be secure; in a word, these devices must be trusted.

Creating that trust is challenging. IoT design falls into three tiers: edge, gateway, and infrastructure. Edge devices interface with sensors and actuators, taking readings and interfacing with the physical world, and most often connect wirelessly to a gateway. Incoming data is aggregated and analyzed in gateways, then passed to an infrastructure (often referred to as "the cloud", but implementation can vary with use cases) for further processing, storage, and presentation. Any weak link in performance, power consumption, wireless signal integrity, security, or other issues can cause a system-wide problem.

IoT applications will test the mettle of design teams. Rushing hardware and software to market in order to declare "first" versus competition may be counterproductive when flaws are uncovered. Business customers in particular are proceeding very cautiously, asking for pilot installations on a small scale before rolling out full-scale deployments. SoC designers will need to customize, explore, test, and perhaps adapt rapidly but carefully.

FPGA-based prototyping will prove crucial for IoT designs. Many of these design starts, particularly for edge devices, will be small in terms of gate counts, but may seem larger in terms of verification testing needs. Expertise in wireless technology, security, power management, and other disciplines will become a differentiator for SoC teams.

Running more tests on a wider variety of IP blocks within limited schedules calls for cost effective, flexible platforms that can handle any configuration.

Many IoT-related specifications exist, and it is very possible designs may be forced to respond mid-stream as new ones appear and others fall away. Algorithms will also adapt as researchers provide new breakthroughs. With the right FPGA-based prototyping tools, hardware and software can be explored thoroughly against real-world use cases, which will be better understood as more IoT deployments occur.

The overall theme in these applications is system-aware SoC designs win. Rather than just implementing a set of functional requirements, the new era of smart SoC design anticipates use cases and tests hardware and software accordingly – pre-silicon.

Next, the "Implementing an FPGA Prototyping Methodology" Field Guide authored by the teams at S2C looks at FPGA-based prototyping from a practical viewpoint with tips on how to choose a platform, addressing scalability, and implementing a design flow.

NOTES

[62] "Methodology of Multi-FPGA Prototyping Platform Generation", Qingshan Tang, Universit´e Pierre et Marie Curie - Paris, January 13, 2015, https://hal.inria.fr/tel-01256510/document
[63] "Part 9: The 2014 Wilson Research Group Functional Verification Study", Harry Foster, July 19, 2015, https://blogs.mentor.com/verificationhorizons/blog/2015/07/19/part-9-the-2014-wilson-research-group-functional-verification-study/
[64] "CoreTile Express", ARM website, https://www.arm.com/products/tools/development-boards/versatile-express/coretile-express.php
[65] "All Programmable Heterogeneous MPSoC", Xilinx web site, http://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html

Implementing an FPGA Prototyping Methodology

FIELD GUIDE

Authored by the team at

When Do You Need an FPGA-based Prototyping Solution?

We all know that complexities in design and shrinking time-to-market windows are driving up design and verification costs. More and more, engineers are turning to hardware platforms for their quest to verify their designs on time. Hardware platforms such as FPGA prototyping are growing in popularity due to their relative low expense and ability to test system designs at speed versus simulation which is too slow and often can't provide an accurate assessment of design behavior. FPGA prototyping has often been typecast as a solution used solely for small designs late in the verification process just before the software development stage, with concerns that the difficulties of employing prototyping across multiple FPGAs have outweighed the cost and speed benefits of implementation for large designs. Emulation has been the choice for verifying large designs because of its capacity, but it too has limitations.

The truth is that today's FPGA-based prototyping advancements are breaking that restrictive notion. Innovative hardware and the addition of cutting-edge software have made it possible to realize the benefits of FPGA prototyping not only for system validation and software development, but also much earlier and throughout the design and verification flow as well as for extremely large designs.

FPGA-based prototyping is well suited for even the largest designs. FPGA capacity has increased exponentially to reach up to 44M ASIC gates (per FPGA) and can fit up to a billion gates (by using an array of FPGAs in a single system). These increases in capacity have not affected FPGA prototyping speed and costs relative to emulation; it's still much faster and cheaper than emulation.

To figure out if FPGA-based prototyping is the optimal choice for your design and verification flow, you should ask yourself these questions.

1) Is testing of my design in real-time critical to design success?

To answer this question, you should analyze the importance and complexities of your design's functionality. As mentioned

earlier, simulation can only get you so far when dealing with complex device behavior. Deep and accurate assessment can't be achieved simply through simulation no matter how many regressions are done. Confidence in a design can often times only be achieved through the ability to test it in real-time scenarios especially for designs heavily dependent on timing accuracy.

Many applications need to be tested in real time or close to real time to assess the quality of the design. Examples of this are Video and Audio applications. In addition, some designs require real-world testing involving outside environments, noise, or interfacing with 3rd party designs and infrastructures.

2) How many tests will you need to run and what is the time window that needs to be achieved between testing and implementation?

As your design stabilizes and matures, validating the software components come into play. At this stage, emulation and prototyping have distinct advantages. If you need to get your model up and running quickly with only the need to run a few tests, then emulation is ripe for your application. Emulation may only need a few hours to set up and get going, while an FPGA prototype can take weeks. However, if the number of tests you need to perform are more significant and you need faster performance for software development and compatibility testing, then FPGA prototyping might serve your needs better. Although it may take much longer to set up, FPGA prototyping is unarguably much faster than emulation. Typical emulation speeds run at about 500 KHz where prototyping can easily run between 10 and 50 MHz with some reaching as much as 100MHz.

Given these speed differences, the point of performance/testing crossover between these two solutions is strikingly short (even with FPGA prototyping's long set up time) and the performance gap grows dramatically thereafter. The difference in performance is particularly steep when prototype replicates

(copies) are used in parallel. The cost advantage that FPGA-based prototypes enjoy – typically 5 to 1 – allows multiple platforms to be deployed, thereby accelerating overall performance. Therefore, you can complete exponentially more tests in a shorter amount of time when using FPGA prototyping.

You can read more about the performance cross over comparison in the EE Times article "Emulation vs. Prototyping -- The Performance Curve Crossover."

3) What is your budget?

A simplistic view is that emulation is expensive when compared to FPGA prototyping. However, a deeper analysis of this idea is merited. Most companies can afford to implement a few emulators for early design verification, but when implementing for a large number of replicates for software development and compatibility testing, the costs of emulation soar. As mentioned in the answer to question 2, the cost advantage of FPGA prototyping is 5 to 1 compared to emulation for even faster and more cost effective performance.

There's no arguing against the tried-and-true methodology of emulation. The inherent strengths of emulators are well-suited to system integration efforts and rigorous verification testing. In fact, the direct results of performing emulation are designs that stabilize and mature more quickly. This in turn precipitates a shift from verifying hardware elements to validating software components.

When this change in focus occurs, FPGA-based prototypes become the natural platform to pick up the pace of validation and further drive software development. Performance crossover analysis serves as an aid in determining when to make that shift, and why. Ultimately, this is a powerful demonstration of how emulation and FPGA-based prototypes are complementary tools – not despite a performance crossover, but because of it.

How Do I Choose Which Solution to Implement?

Now that you understand when and why you need FPGA prototyping, you need to know the various FPGA prototyping solutions that you can employ to maximize your investment. To set up your analysis for the various FPGA prototyping options, you have to consider your design size and application, design stage, and resource management requirements. You'll also need to take a look at your FPGA prototyping specifications and then various options in terms of off-the-shelf or building your own.

Design Specifications

For the size of your design, think in terms of capacity. Without enough gate-level capacity to accommodate your design, you can't build a prototype. Most systems need adequate memory too, so having sufficient memory available is critical.

You also need to think about the type of application you are building. Is it IoT, Automotive, Super Computing, Data Storage, Cloud Computing, Image Processing, a Communication Network, or is it something else? Is it a design that contains a large number of DSPs or does it require a lot of logic resources or memory resources? Is it based on a specific protocol like PCIe or a particular bus standard like AXI? There are many different types of prototyping hardware and software solutions that cater to these different application types. Some hardware boards are flexible enough to scale with your design and allow you to adapt to different design types through extensions and daughter cards while others do not.

The design stage refers to when within your design methodology flow you'll implement FPGA prototyping. We talked earlier about the FPGA prototyping sweet spot being used during software testing and validation and that it is well suited for designs that are fully rendered in RTL that can be mapped to an FPGA. However, recent advances in FPGA prototyping technology have extended its value into other areas. For example, many designs may not be completely mapped to an FPGA and may be only partially available as behavioral models in descriptions such as C++ or SystemC.

In these cases, transaction-level interfaces play a critical role in being able to bridge the abstraction level between behavioral models and live hardware. Transactors offer a way to communicate between software running on a host and an FPGA-based prototyping platform that often includes memories, processors, and high-speed interfaces. These transactors can be implemented over a well-known bus protocol such as AXI or an industry-standard transaction protocol such as SCE-MI. Transactors extend the functionality of the system allowing it to be used for algorithm/architectural exploration, system integration with virtual prototypes, and exhaustive testing through software generated corner tests.

An often-overlooked aspect of design and verification is the management of hardware, software and personnel resources to maximize efficiency. This not only includes the assignment of tools to key engineers throughout the flow but also includes the behavioral reporting of these assets. In today's connected world, companies now have the luxury of taking advantage of engineering talent across the globe. Because of this, many design teams are geographically dispersed.

Access to FPGA prototyping systems have typically been constrained by the use of localized systems that require local management and control. This limited access has presented a significant hindrance to modern SoC design teams – especially software development teams – which again are often globally distributed. There are advances in FPGA technology to alleviate these circumstances to allow for wide distribution of hardware and management software resources through the cloud. If you are working with a globally dispersed team, this will be a factor in choosing the right FPGA prototyping platform for you and your team.

FPGA Prototyping Specifications

There are a number of FPGA prototyping solutions on the market. The above analysis will impact your decision on which one you choose. First, let's take a look at how your design size factors into the specific FPGA prototyping solutions. Individual FPGA capacity has increased significantly over the years reducing the number of FPGA devices needed. The vast array of FPGA options on the market span a wide

range of capacities to fit your design size requirements. The larger your design, the most likely you'll need an FPGA or FPGAs with the capacity to follow suit. Today's FPGAs can handle designs of up to 44 million gates. Even with these high-capacity FPGAs you must keep in mind that the usable capacity of an FPGA is roughly 50-70% when incorporated into an FPGA prototyping environment regardless of the FPGA prototyping solution that is chosen. Given this fact and that most designs scale beyond the limits of a single FPGA, a multiple FPGA prototyping solution is the norm.

Choosing an FPGA prototyping solution that can scale with your design's needs is preferred. To get the most from your prototyping solution, you must consider how easily the prototyping environment can scale. Does the architecture of the prototyping solution accept additional hardware be it more prototyping boards or specific daughter cards geared for a particular design type or design characteristic? Moreover, does the prototyping solution have the necessary integrated components to scale with your design? Some designs are so large that only a handful of FPGA prototyping solutions have the scalable architecture to keep pace. For example, S2C's Cloud Cube (a chassis) can connect up to 32 FPGAs to reach design capacities of up to 1 billion gates. However, capacity can scale even further when multiple Cloud Cubes are employed.

Multi-FPGA prototyping platforms do have significant issues when it comes to I/O count and performance. Not only is mapping large multi-million gate designs to multiple FPGAs a challenging task, but performance may suffer because of timing delays between the FPGAs. Therefore, it becomes apparent that choosing a platform with the ability to handle complex partitioning is essential to reduce repartitioning and maintain proper real-time performance.

Multi-FPGA platforms also come with the added difficulty of debugging. It used to be that signals internal to an FPGA could not be probed unless they were brought out through the I/O. Fortunately, major FPGA vendors have internal logic analyzers to address the visibility issue. However, many of these internal logic analyzers have several limitations, including support for only single FPGA debug,

limited memory size using FPGA internal memory, and long place-and-route times to change probes. Debugging a design partitioned across multiple FPGAs is all but impossible without a tool that helps set up probes and makes signals easy to track based on their RTL-level names. Debugging should use FPGA I/O efficiently and maintain a useful debug trace.

Of equal importance is the ability to reuse a prototype (or even part of one) to save development time and lower implementation risk for future projects. But this is difficult to achieve with a board built for a specific project. As SoC designs grow in size, they may no longer fit in older FPGAs. If the interface to an external system is built directly on the prototyping board, it can't be reused for projects in which the interface is different. The ability to reuse your prototype platform enhances its usefulness, speeds the process of developing new prototypes, and reduces overall costs.

Given the points made above, there are three options for the type of FPGA prototyping platform you can implement. All of these options utilize FPGAs from such vendors as Xilinx and Altera. Specifications for each of these vendors' latest FPGAs are shown below.

Image FG-1: Comparison of latest FPGAs from Xilinx and Altera

		Xilinx (Virtex UltraScale)	Altera (Stratix 10)
Type		XCVU440	GX 5500/ SX 5500
Logic Resources		5,541K System Logic Cells	5,510K Equivalent LEs[1]
Memory Resources		88.6Mb	137Mb
Clock Resources		30	-
I/O Resources	**Max User I/O**	1,456	1,640
Integrated IP Resources	**DSP Resources**	2,880 DSP Slices	1,980 Variable-Precision DSP Blocks
	PCIe Gen1/2/3	6	3
	GT Transecivers	48	72

The first option is a full custom board often referred to as a build-your-own platform. The connections for custom platforms for both inter-FPGA and the external interfaces are very specific to a particular design, which is their advantage. The nature of these platforms lends itself well to increased performance and maximized use of external interfaces. Creating a custom platform is an extremely time-consuming endeavor resulting in an eventual reduction in productivity. The

expertise necessary to create and implement such a solution can be daunting and not necessarily in the "wheel house" of most prototyping teams. Beyond this limitation, there is the fact that most custom platforms cannot be reused for other projects due to the specificity of project(s) they were designed for. When you factor in the time, energy, and risks associated with unproven build-your-own boards the expense get be quite high.

A second option is the off-the-shelf platform that is comprised of a pre-built prototyping board with fixed connections for communication between each FPGA as well as the external interfaces. This type of board is often referred to as an application-specific FPGA board. These can be comprised of a single FPGA or multiple FPGAs. Examples include Xilinx and Altera evaluation boards and many PCIe-based FPGA boards. The advantages of using an off-the-shelf solution are reliability and faster time-to-market. These solutions have been thoroughly tested to avoid bring-up errors when deployed in the field. The real differences between the various off-the-shelf solutions depend on the following:

- The type of FPGA that is used,
- How many FPGAs are used,
- What external interfaces are on the board,
- What expansion capabilities are there for the board,
- Does the board come as a reference design?

These systems typically lack support for partitioning and debug. Scalability is usually not an option, often resulting in having to toss out the system when the design expands or external interfaces change.

The third option is a scalable or modular approach where cabling and connectors are used to connect multiple off-the-shelf FPGA boards. The connections for both inter-FPGA and external interfaces are user-defined. The benefit of this solution is that performance may be enhanced by the varying distribution of the interfaces and cables. This approach also fits most design needs in terms of capacity and external interfaces. These are reusable platforms, scalable and flexible as designs grow and design specifications change. When it comes to partitioning across multiple FPGAs (covered in depth later), the interconnections

can be adjusted through cables and/or interconnection modules for higher performance. The flexibility with this approach is inherent, but along with it comes an enormous amount of cables to manage scaling beyond 4 FPGAs. There needs to be a balance between utilizing on-board interconnections versus cables. Should you use Single, Dual, or Quad FPGA modules as a basis to scale?

Building a Scalable Prototyping Platform

If you've determined that you need a scalable FPGA prototyping solution, then this next section will guide you through the process of creating a scalable platform.

Whether you need scalability for your current design as you move through the design and verification process or whether you need your FPGA platform to be reusable and able to scale for future designs that may be larger than your current one, it all starts with identifying and selecting the ideal building blocks. The foundational prototyping board you choose must have flexibility to expand so a custom platform is usually out of the question as a custom board requires even greater customization to grow. When crafting your platform, there are three initial FPGA building blocks to evaluate: Single FPGA boards, Dual FPGA boards, and Quad FPGA boards.

Selecting either a single, dual, or quad board depends on your design's size, memory requirements, and the number of inter-FPGA connections and external I/Os that will best fit your needs. The chart below provides an example of the differences in these board types based on S2C's solutions for its Xilinx Virtex UltraScale Logic Modules.

These comparisons don't tell the whole story though. You must take a closer look at the architecture for each of these solutions. Besides the number of physical interconnections between FPGAs, the type (DDR3 or DDR4) and capacity (4GB, 8GB, or more) of on-board memory is equally important to your design. Of additional interest should be the number of high-speed gigabit transceivers and their performance level. The following diagrams provide in-depth comparisons of each of the architectures for single, dual, and quad FPGA prototyping boards.

Image FG-2: single FPGA module architecture

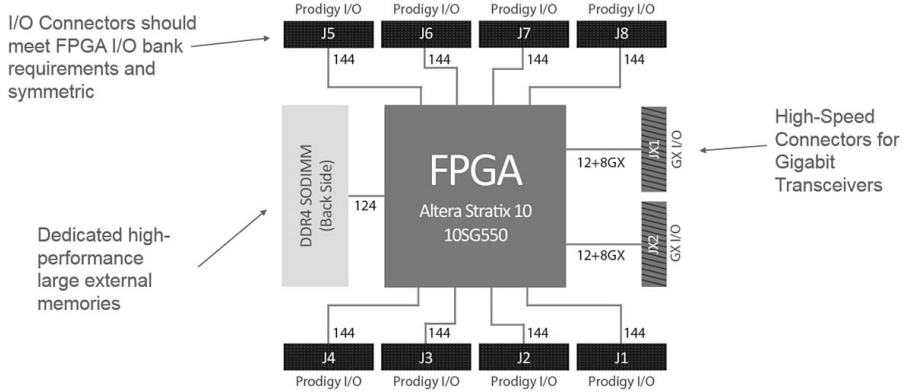


Image FG-3: dual FPGA module architecture

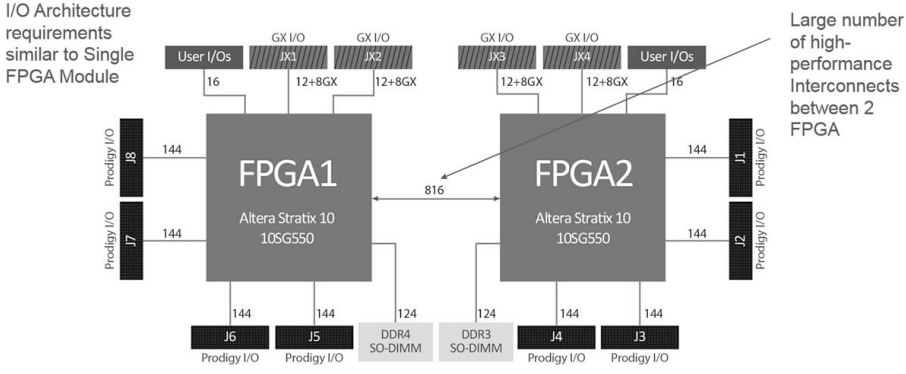
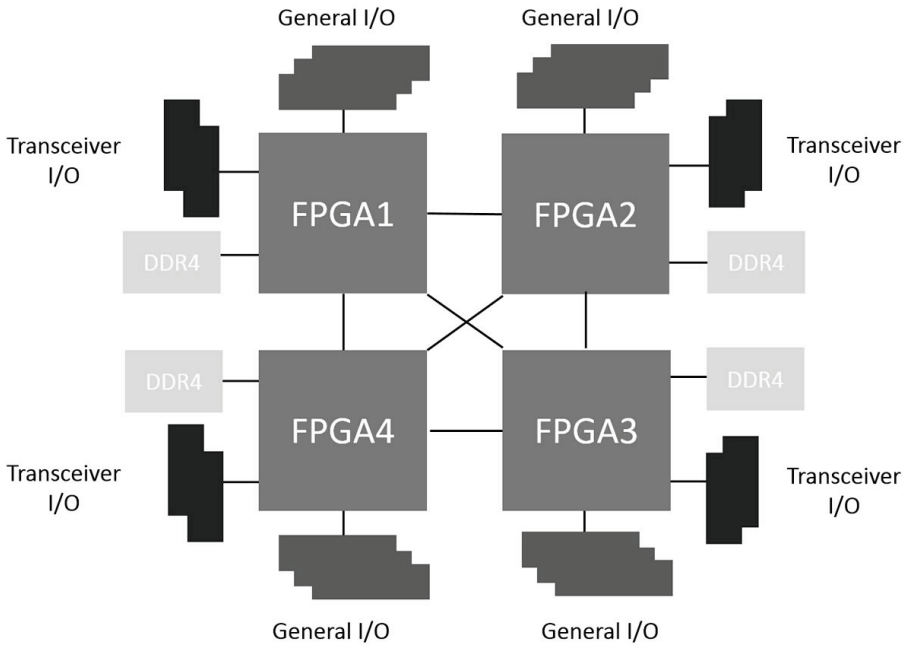


Image FG-4: quad FPGA module architecture



The type of I/O connectors used in the FPGA module may have a big impact on your design mapping and performance. First, they must be optimized for FPGA I/O banks, and even the FPGA die, in case some FPGAs have multiple internal die. In addition, having I/Os from different die will decrease performance. All traces from the FPGA to the same I/O connector should have the same trace length to increase bus performance. Connector performance itself may also play an important role especially if the connectors are optimized for running high performance LVDS (low voltage differential signaling), especially at rates over 1 GHz.

It's All About Flexibility

The foundational prototyping board is the first step in building scalability. Each solution whether a single, dual, or quad system must allow you to grow, you must be able to have the flexibility to grow your single system into a dual, quad or beyond. Likewise, your dual system should allow you to stitch together other systems of the same FPGA type and architecture to create a quad system.

Even with this flexibility, there are some implications to the number of interconnects and I/Os when stitching together these systems. Careful consideration must be given to which system you initially choose. You will notice in the following diagrams that building these multi-FPGA systems require the ability for the boards to be connected via cables or interconnection modules. These systems will also need some sort of external module to manage global clocking and reset mechanisms.

Image FG-5: Connection of two single FPGA prototyping modules

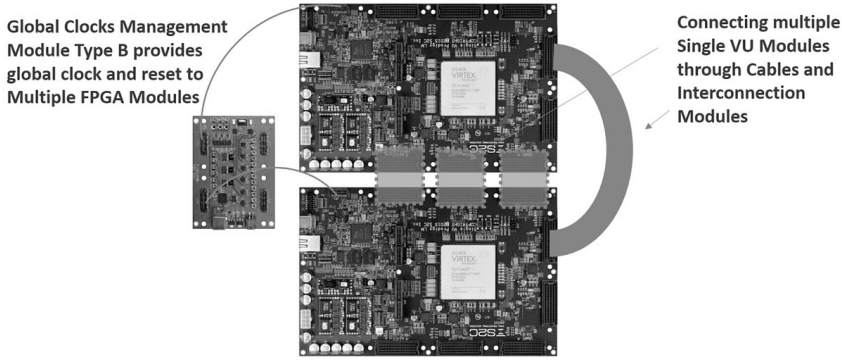
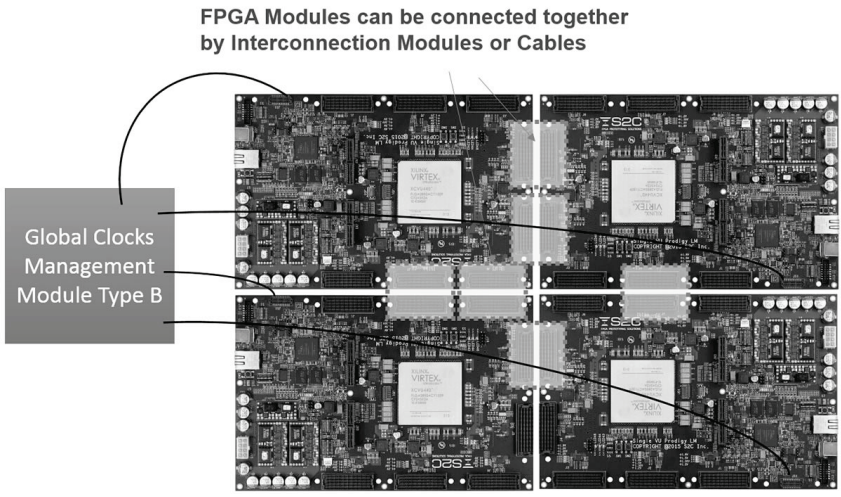


Image FG-6: Connection of 4 single FPGA prototyping modules



Going Beyond 4 FPGAs

What happens if your design needs require going beyond the use of either 4 single FPGAs, 2 dual FPGAs, or a quad FPGA system? This increase in complexity triggers a whole new set of scalability questions. These questions can be broken down into several categories.

Space – How big of a desk or lab area do you need to work with a large number of FPGAs? Although you can continue to stitch together multiple prototyping boards to expand beyond a quad system, your physical lab space may be limited making the connections of these boards much more complicated. Not only will you be dealing with space issues, but also the cabling of these systems will become very unwieldy.

Scalability & Flexibility– What if you require more logic and memory capacity or the system interfaces or memory types change? Can you configure the large number of FPGA resources for multiple designs? Because of the investment into large multiple board systems, these reusability type questions become important. It is much easier to invest in single board systems if the expectation is that the board will have limited use beyond the initial design. However, when the initial design requires the use of a larger prototyping system, your investment must consider possible changes in the prototyping environments and future project uses.

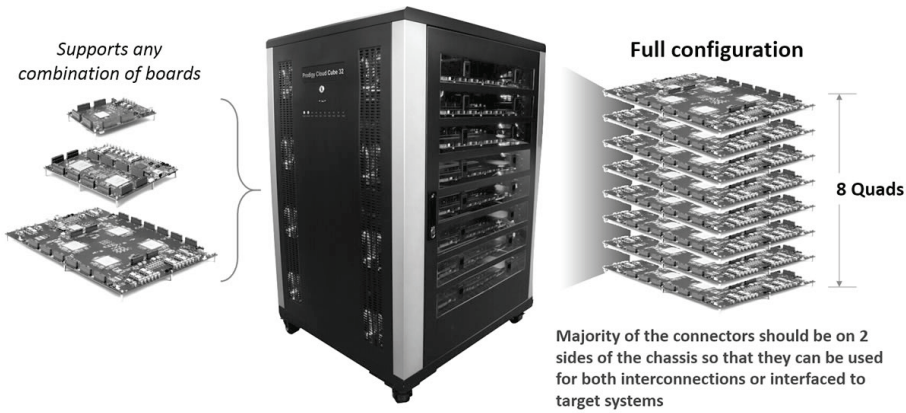
Global System Control – How do you provide low-skew clocks and resets to a large number of FPGAs that you are using for the same design? Is there a way to easily download to FPGAs remotely and how fast is it? Lower-end software can provide some sort of support for these questions but may miss some basic requirements. Furthermore, the larger the overall hardware system, the more difficult it is to control such things as clocks and resets. Downloading for larger systems can be a cabling nightmare. Higher-end systems that offer complete runtime support and chassis with minimal cabling help reduce the pain dramatically.

Power Supply – How do you provide power to a large number of FPGAs? Can each FPGA be individually controlled (On/Off/Recycle)? Is

there a power-monitoring feature that you can employ? Providing power individually to each board can impose even more physical lab space issues not to mention complicating the management of powering each board.

Reliability – How do you verify that all your clocks and interconnections are correct? Is there an easy way to monitor the system as well as the individual FPGA statuses? Making sure a complex prototyping system as large as 32 FPGAs works correctly is extremely difficult without automation. If a design isn't running correctly, a great deal of time can be wasted trying to manually determine if the error is due to the design itself or the FPGA system. Software that provides automated self-test capabilities as well as automated voltage, current, and temperature monitoring with shut down will provide much needed peace of mind.

Image FG-7: S2C's Cloud Cube supports any combination of FPGA boards, and up to 8 Quad boards can fit into the chassis.



Working with a Chassis Architecture

Many of the issues raised by the above questions can be alleviated through the use of a chassis for the FPGAs. Employing the use of the right chassis will allow any combination of boards (whether they be single, dual, or quad) to be easily housed to fit restrictive lab space requirements, connected to reduce cabling, and managed to improve

overall efficiency. As an example, we'll use S2C's Cloud Cube to outline how this type of architecture can be fully leveraged.

A chassis system should easily support both single and multiple module clock & reset requirements including available global clock resources and types, internally generated clock, and clock skew. The diagrams below illustrate how this is done.

Image FG-8: Single Module clock and reset

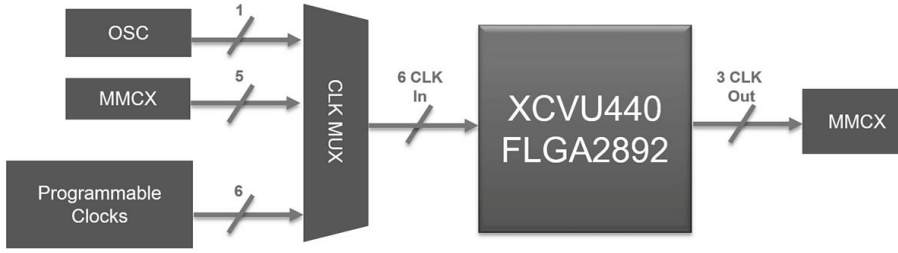
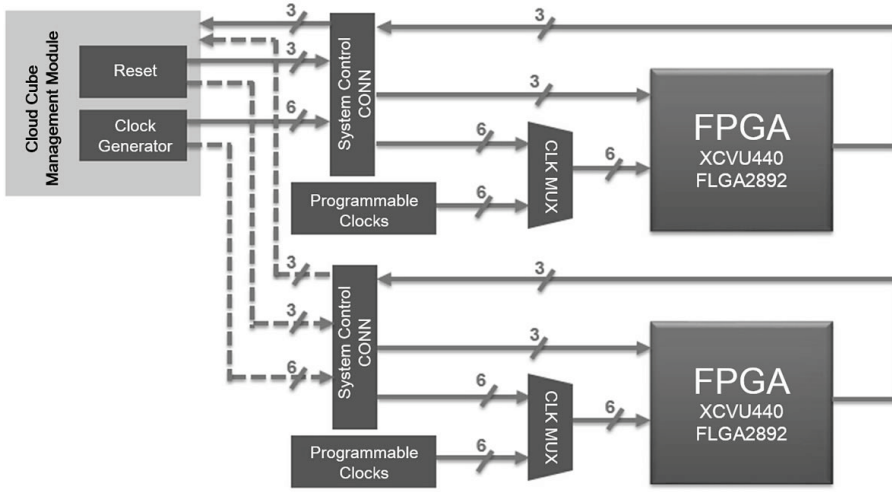


Image FG-9: Multiple Module clock and reset



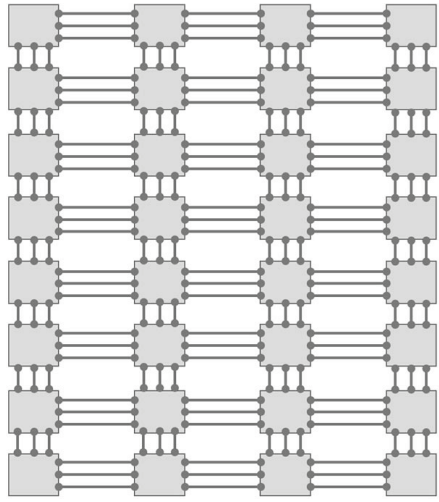
Regardless of utilizing a chassis or not, the ability to control, monitor, and manage the FPGA modules is critical. However, working within a

chassis system makes performing the following tasks much easier with some having built-in automation:

- Monitoring the system status,
- Powering the on/off/recycle of individual FPGA modules,
- Controlling global clock & reset and board clock & reset,
- Auto recognition and detection of installed FPGA modules, cable, and daughter cards,
- Remote FPGA downloading,
- Self-testing of cables, hardware, and FPGA modules,
- Monitoring of the entire system in real-time.

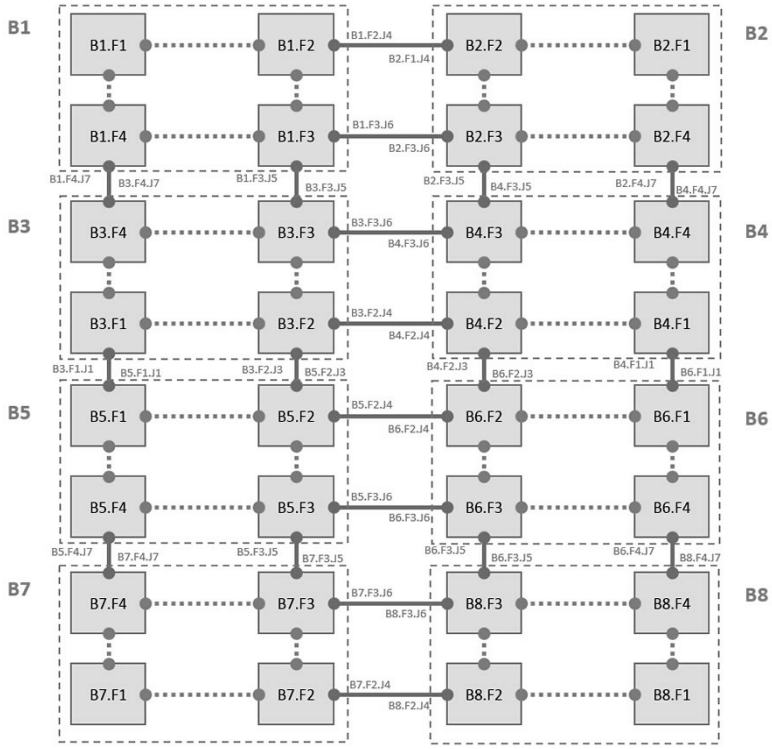
Let's compare the differences in building a 32 FPGA system using a 2-dimensional approach versus a 3-dimensional build using a chassis. We'll start with creating a 4 by 8 MESH of the FPGAs. A 2D setup requires a very large lab space with complex power supply to all 32 individual FPGAs as well as a complex clock distribution. As you can see in the illustration below, using single FPGA modules requires 52 interconnections. If each connection requires 3 banks and each cable transmits 1 bank, then the system needs 156 cables – a cumbersome amount of cables to manage.

Image FG-10: A 2D setup to connect 32 FPGAs requires 52 interconnections resulting in 156 cables



If you implement a 3D approach using Quad FPGA modules in a chassis the setup is much cleaner and requires far fewer cables to manage. The illustration below shows the optimization that can be achieved.

Image FG-11: A 3D setup to connect 32 FPGAs reduces interconnect

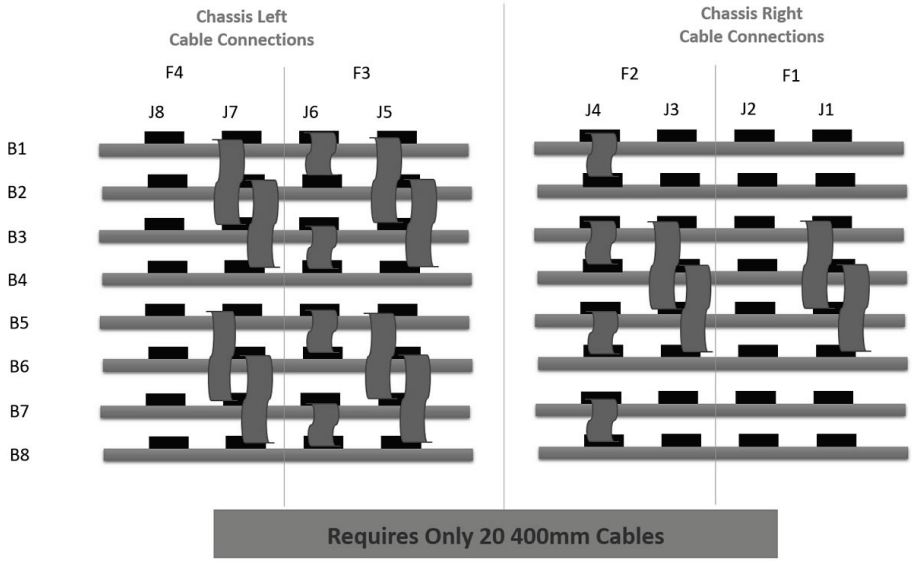


In this logical view of a 4 by 8 MESH system mapping to 32 FPGAs on 8 Quad FPGA modules, Bn is the Quad board number, Fn is the FPGA number on the Quad module, and Jn is the connector number on a Quad FPGA module.

In order to minimize the cable connections and also cable lengths, we group the FPGAs in a specific pattern with some rotations from the logical view. This will also keep the cables from crossing from one side to the other.

Dotted lines signify the on-board interconnects. You'll notice that 32 of the FPGA interconnection points are now using on-board traces and 20 of them use cables. A closer look at the left and right sides of the chassis reveals how these cable connections would look.

Image FG-12: Chassis cabling for 3D interconnect



Overview of the FPGA Prototyping Methodology Flow

So now that you've determined the best FPGA prototyping solution for your needs, let's look at the flow for setting up a prototype.

Setting Up a Prototype

A typical implementation flow for a prototype with multiple FPGAs contains three general parts: Partitioning, Routing / Multiplexing, and Place and Route.

After the design RTL is created and goes through the synthesis process, it is then ready for the partitioning stage. As mentioned earlier, many designs are larger than a single FPGA so the design must be compartmentalized or partitioned into several FPGAs. Partitioning is tricky as the design can't simply be cut into equal parts based on the

number of FPGAs being used. The design needs to be disseminated not only according to capacity, but also according to how and what signals cross between the FPGAs.

However, this is changing as FPGA capacity increases. Proper partitioning minimizes the interconnect counts among FPGAs and thus increases the overall system speed. Partitioning done the right way also keeps the critical design blocks together and allows you to manually lock certain blocks to specific FPGAs for external interfaces. Partitioning can be a daunting task but if done with the right tools it can be easy and efficient. We'll cover techniques for conquering partitioning in the Compiling and Partitioning section of this guide.

Once partitioning is deemed successful, the design moves on to the routing or pin multiplexing stage. When a design goes through the partitioning stage, cut nets are created. Cut nets are the parts of signals that get crossed between each of the FPGAs. During the routing phase, these cut nets get allocated to an inter-FPGA track. Because there are likely fewer available inter-FPGA tracks than cut nets, multiplexing of several of these cut nets into a single track must occur. Again, there are techniques that can be leveraged to make this process go as smoothly as possible and will be discussed later. There are still cases where no pin multiplexing is needed, and many designers prefer to create designs with this goal in order to run their designs at high speed even when partitioned across multiple FPGAs.

Place and Route is the last step, where the bitstream of each FPGA is generated and downloaded into the platform to model the design.

What About Debug?

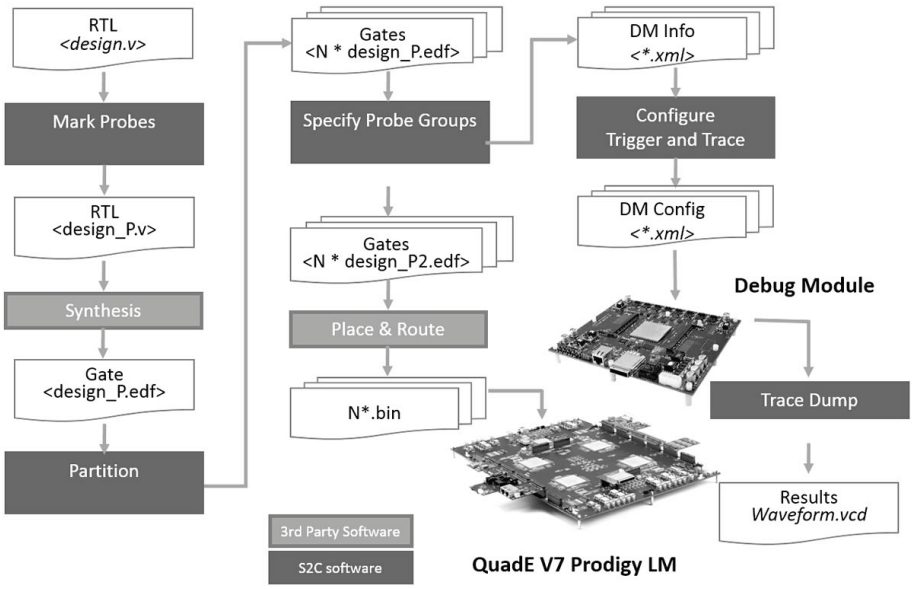
This typical FPGA prototyping flow puts less emphasis on debugging. There is a reason for this: debugging a single FPGA design is a relatively simple task. However, performing debug operations on a multi-FPGA platform is an extremely long and often labor intensive process.

Manual techniques only allow for debugging one FPGA at a time, and traditional tools such as an external logic analyzer or FPGA internal logic analyzer have limitations when it comes to multi-FPGA debug.

With manual processes, only gaining insight into the behavior of one FPGA at a time may result in missed design errors or misleading design behavior as it becomes difficult to test the functionality of the design as a whole. The part of a design that resides on a particular FPGA may be bug-free in its compartmentalized form, but when operated within the totality of the design may contain critical errors. External logic analyzers have a limited number of probes and require designers to pull their probes to the top level so they come out from the FPGA I/O pins.

Because of these issues, debug has been largely inadequate within the FPGA prototyping process thus leaving debug to be done only through simulation and/or emulation. But, hold on a minute. There have been significant advances in FPGA prototyping to deal with the very complex issue of multi-FPGA debug that augment the FPGA Prototyping Flow.

Image FG-13: Multi-FPGA debug flow



To make the debug of multiple FPGAs possible, probes must be set up in the RTL prior to synthesis so that they can be tested down the line. We'll explore the debug flow part of this flow in more depth during the Debug section of this guide.

Details of Implementing the FPGA Prototyping Flow

Now that you have a general grasp of the individual components of the FPGA prototyping flow let's explore how to implement them in detail.

Compiling and Partitioning the Design

Designs utilizing only one FPGA don't require partitioning and therefore designers can immediately enter place & route (P&R) using either Xilinx or Altera P&R tools. However, as we mentioned earlier, FPGA capacity has increased significantly but many of today's designs are still too big to fit on a single FPGA. Partitioning is a required step and partitioning a design incorrectly can have dire consequences in the functionality of the design within the FPGA prototyping environment.

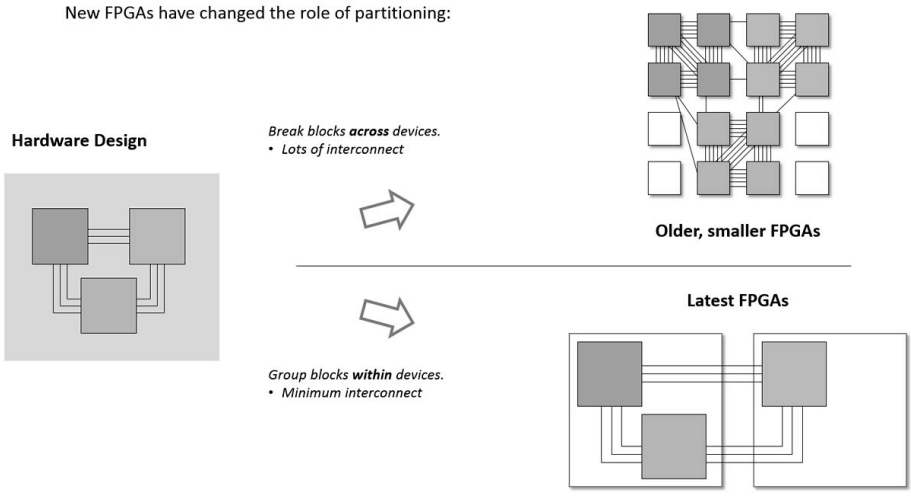
The increased capacity of newer FPGAs has changed the approach to partitioning. Partitioning a design across multiple FPGAs has been known to be a very difficult and time-consuming process that primarily took place at the granular gate-level in order to meet the correct parameters to partition the design correctly. Doing so resulted in the creation of a huge number of interconnects between the FPGAs. Today's high-capacity FPGAs, like Xilinx's Virtex UltraScale and Altera's Stratix 10, have allowed partitioning to come up a level and become more of a grouping exercise.

Because most designs today are IP-based (with functional blocks such as CPU, GPU, and peripherals) the individual blocks are often smaller than a single FPGA and can therefore be grouped at the IP level rather than having to go through more fine-grained gate-level partitioning. As an example one of the biggest ARM processor cores today, the ARM Cortex-A57, can fit into one Xilinx Virtex UltraScale FPGA. Most IP blocks have a manageable number of I/Os and partitioning algorithms should be able to find the best grouping to minimize the number of interconnects among FPGAs. The result is a much easier and smoother partitioning experience.

Partitioning can be done manually of course and still many designers are doing manual partition for smaller numbers of FPGAs. However, the work is tedious and error-prone and often results in long debug

cycles. Utilizing commercial partitioning tools can save both time and help improve the performance of your design.

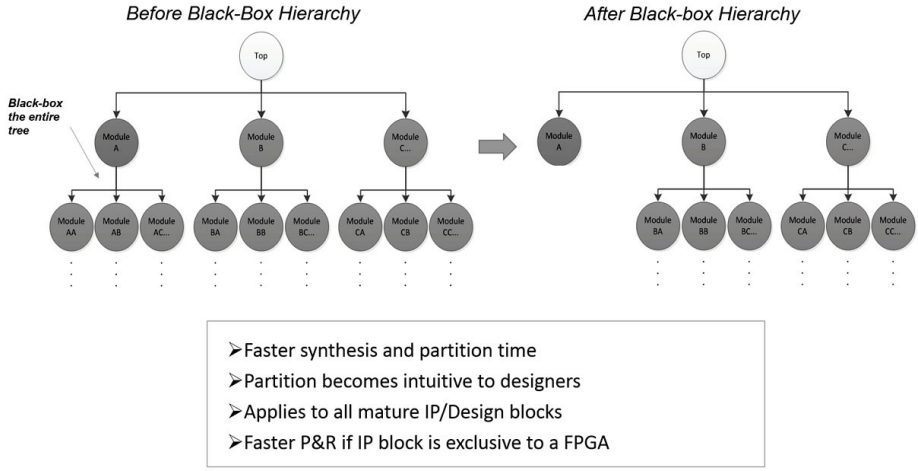
Image FG-14: Larger FPGAs mean reduced interconnect



Partition speed is always important. Traditionally a gate-level partition engine can take extremely long hours to partition today's large SoC designs. However, gate-level partitioning is no longer required today as mentioned earlier and therefore partition speed can significantly increase. In addition, by making some design hierarchies "black boxes" you can further increase partition performance and simplify the entire compile flow.

With black-box partitioning, designers can choose which design hierarchy tree should be black-boxed and therefore any lower level trees of that hierarchy are hidden from partitioning. The result is an increase in partitioning performance. The lower level hierarchy trees are not completely forgotten but simply merged at the FPGA P&R step of the flow.

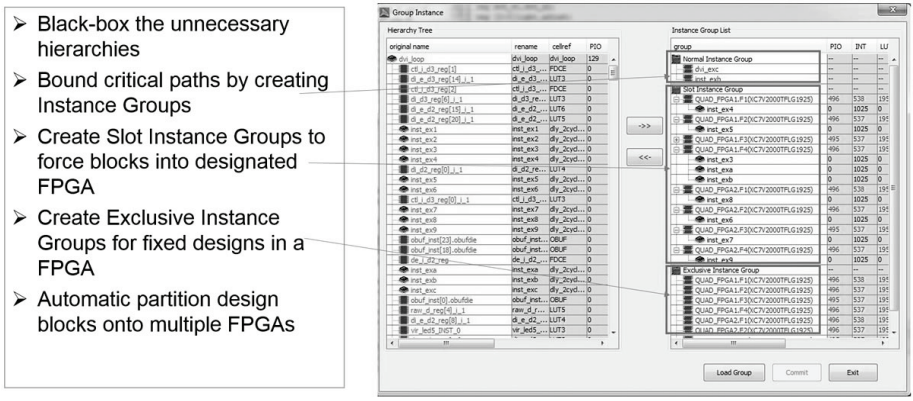
Image FG-15: Black-box partitioning



No matter how good the automatic partitioning algorithm is, it can always be better with some guidance from designers. Therefore, the following type of grouping constraints should be set by designers to guide the automatic partitioning engine.

- Normal Group – These are IP or design blocks that you know should be located in the same FPGA for increased performance or to minimize the interconnection nets.
- Slot Group – These are specific IP or blocks that should reside in a specific FPGA. Often there are specific external interfaces that are only available in one of the FPGAs on the board and you can use this feature to lock the block to the correct FPGA.
- Exclusive Group – Some portion of the designs might be fixed and you do not want to touch or re place and route again. Exclusive group means only the selected IP/blocks will be allowed in a specific FPGA.
- Global Group – some design blocks may need to be duplicated into multiple FPGAs to increase performance and/or minimize the interconnects among FPGAs. Examples are circuits that generate global clocks.

Image FG-16: Grouping items to guide partitioning



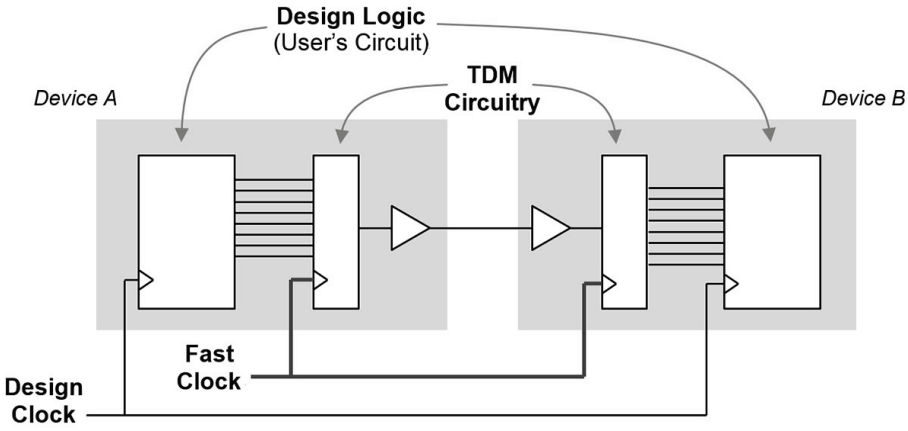
Splitting design blocks/IP into multiple FPGAs is half of the partitioning process. FPGAs have a limited number of pins and therefore the other half of partitioning involves making sure there are enough I/Os, either physically or through pin-multiplexing.

As newer generations of FPGAs become available, the physical I/O counts increase dramatically. A Xilinx Virtex UltraScale (VU) has 1,456 I/Os and an Altera Stratix 10 is planned to have 1,600, as compared to the 1200 I/Os available from a Xilinx Virtex 7 (V7). The I/Os themselves perform better and can handle pin-multiplexing schemes more efficiently. For example, in Xilinx's V7, LVDS can run at 1 to 1.2 GHz compared to its next generation VU that can run at 1.6GHz.

Most complex multi-FPGA designs will require the use of pin-multiplexing for efficiency. Therefore, it is a good idea to understand the different methods for conducting pin-multiplexing. The classic solution is to use a TDM (Time Domain Multiplexing) scheme that multiplexes two or more signals over a single wire or pin.

This solution is still widely employed and serves as the foundation for today's pin multiplexing. However, with advances in I/O technologies, the need to serve multiple clock domains, and the increasing reliability of pin multiplexing, many flavors of TDM have emerged to address different design requirements.

Image FG-17: Signals multiplexed with a fast clock



Flavors of TDM

There are many flavors of TDM methods. TDM can be either synchronous or asynchronous. TDM can be single-cycle or multiple-cycles. Finally, TDM can use different I/O standards such as using single-ended vs LVDS I/O.

Synchronous TDM

In synchronous TDM the multiplexing circuitry is driven by a fast clock that is synchronous with the (user's) design clock.

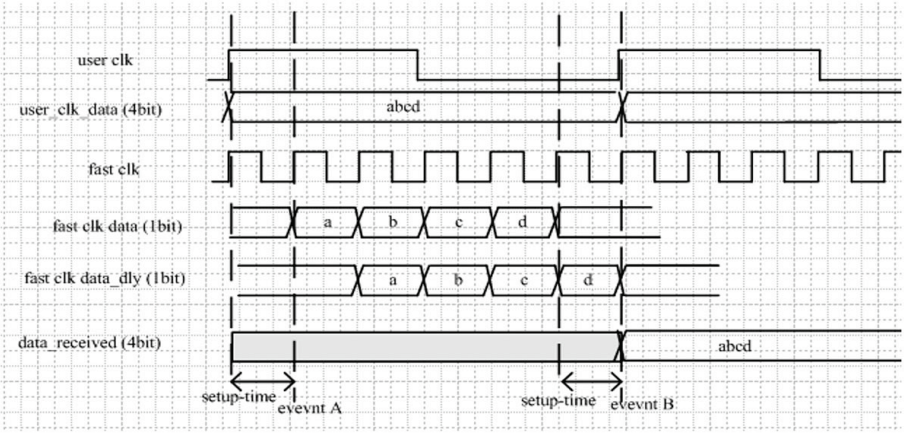
Synchronous mode is sufficient for many TDM implementations, but there are limitations. There must be no feed-through nets between FPGAs before inserting TDM (signals that pass through an FPGA without terminating at a register).

In addition, the difference between the fast clock and the design clock can introduce issues. The timing diagram below shows an example of this where event A is the sampling time for the fast clock, and event B is the sampling time for the design clock – the setup time for both needs to be the same as a single period of the fast clock.

And the interface between the two clock domains could contain a critical path, especially when the TDM ratio is quite large. (This is true

even where all inter-FPGA nets are registered input/output.) This path is often routed poorly inside the FPGA and usually suffers from timing violations due to limited FPGA routing resources. This in turn significantly decreases the speed of the fast clock which decreases the speed of the design.

Image FG-18: Synchronous TDM timing



Finally, synchronous TDM typically supports only one clock per one set of pins. Usually this requires stricter timing constraints that can be hard to meet with a lot of pins, making it difficult to automate.

Asynchronous TDM

In asynchronous mode, the TDM fast clock runs independently of the design clocks. Although asynchronous mode is slower, it supports multiple clocks so timing constraints are easier to meet.

Asynchronous TDM addresses the timing violations caused by synchronous mode, and does not require a timing constraint on the datapath between clock domains (usually equal to one-cycle of the fast clock). In fact, the fast clock can always run at its maximum speed. (For LVDS TDM, this is 1 Gbps for V7 and 1.6 Gbps for VU.) This means the design clock speed won't be affected by a potential reduction of the fast clock, as in synchronous mode.

An additional benefit is that asynchronous TDM is not sensitive to feed-through nets so these can be used with an asynchronous scheme.

However, the designer should be aware that feed-through nets transmitted over asynchronous TDM can impact system performance.

Single-cycle and Multi-cycle clocks

The majority of designs utilize a single-cycle clock. The bottleneck for pin multiplexing frequency becomes the latency rather than how fast signals can be transmitted between devices. Since LVDS has a longer latency, LVDS can actually be slower than single-ended signals when the TDM pin ratio is low. However, when the TDM pin ratio is high, the LVDS latency becomes less of a factor and therefore runs faster than single-ended signals.

As for designs that use multiple clock cycles, they can run at full transmission speed. However, since the data doesn't get to the destination in 1 design clock cycle, the designer must manually insure this is okay for their design. This issue is design dependent, and as result, can't be automated.

Single-ended signals versus LVDS

Single-ended TDM uses a single-ended signal which can transmit physical signals at a speed up to 290 MHz in VU. This is determined by dividing the TDM ratio (or signal multiplexing ratio) and taking into account setup, synchronization, and board delays.

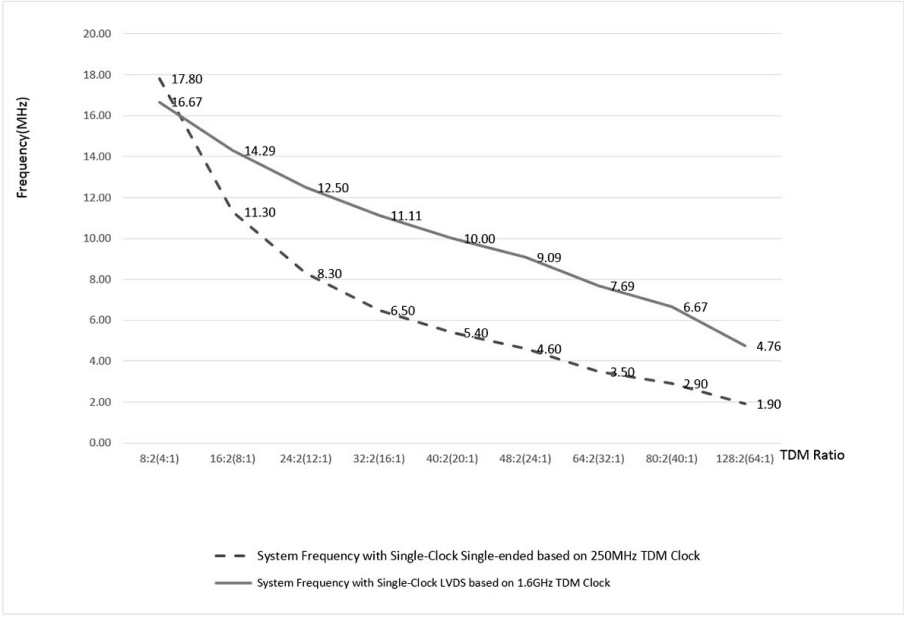
With a TDM ratio of 4:1, the system clock speed will be around 17.8 MHz. If the TDM ratio is increased to 16:1, the system clock speed will drop to less than 10 MHz. From this we can see that as the TDM ratio increases the performance drops linearly.

However, using the LVDS I/O standard supported by Xilinx FPGAs, the physical transmission data rate between FPGAs can achieve up to 1.6 Gbps. This offers tremendous advantages over single-ended transmission, even when considering that a single LVDS signal requires a pair of single-ended pins.

A comparison between single-ended TDM and LVDS TDM using Xilinx UltraScale devices shows the difference. (Note: performance for different FPGA families vary.) Performance of TDM implemented with

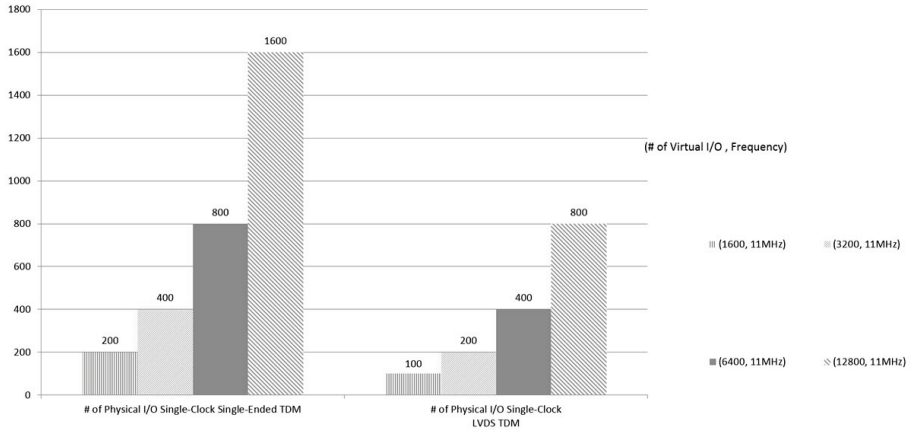
LVDS is typically better than single-ended TDM, especially for higher TDM ratios.

Image FG-19: Single-ended TDM and LVDS TDM performance with Asynchronous mode



A different view illustrates another comparison of Single-ended TDM and LVDS TDM. It shows the number of physical I/O needed to accommodate a given number of virtual I/O, assuming a system speed of 11 MHz. This shows that for a system with a clock speed of 11 MHz, if 12,800 virtual connections are needed, single-ended TDM consumes 1600 physical I/O. With LVDS TDM, this number is cut in half to 800.

Image FG-20: Number of physical interconnects needed for a system running at 11 MHz



Given the physical I/O limitation of FPGAs, partitioning becomes easier if less physical interconnections are needed. LVDS TDM has clear advantages over traditional single-ended TDM.

TDM Performance Comparison

A chart comparing asynchronous and synchronous modes with single-ended or LVDS TDM provides a good summary of the estimated performances using the various forms of pin-multiplexing. No single method is better than the other. Preference depends on your target performance, your design, and the amount of effort you are willing to put into partitioning. For example, LVDS may not be always be faster than using the single-ended method because of the long set up time required for LVDS. Therefore, for low pin ratios, the single-ended method may actually be faster.

However, if your design can afford data to get from one FPGA to another FPGA in multiple cycles, you can run at near full LVDS speed divided by the pin-multiplexing ratio. Of course, this is limited to just one clock domain and cannot accommodate mixing multiple clock domains without modifying the design.

Image FG-21: Comparison of TDM modes

Asynchronous mode				Synchronous mode					
LVDS @1.6Gbps		Single-ended @250Mbps		LVDS @1.6Gbps			Single-ended @250Mbps		
Pin ratio	System speed	Pin ratio	System speed	Pin ratio	One-cycle	Multi-cycle	Pin ratio	One-cycle	Multi-cycle
8:2	16.7M	4:1	17.8M	8:2	33.3 M	200M	4:1	41.6M	62.5M
16:2	14.3M	8:1	11.3 M	16:2	28.6 M	100M	8:1	25.0M	31.2M
24:2	12.5M	12:1	8.3M	24:2	25.0 M	66.7M	12:1	17.8M	20.8M
32:2	11.1M	16:1	6.5M	32:2	N/A	50.0M	16:1	13.9M	15.6M
40:2	10.0M	20:1	5.4M	40:2	N/A	40.0M	20:1	11.3M	12.5M
48:2	9.1M	24:1	4.6M	48:2	N/A	33.3M	24:1	9.6M	10.4M
64:2	7.7M	32:1	3.5M	64:2	N/A	25.0M	32:1	N/A	7.8M
80:2	6.7M	40:1	2.9M	80:2	N/A	20.0M	40:1	N/A	6.2M
128:2	4.8M	64:1	1.9M	128:2	N/A	12.5M	64:1	N/A	3.9M

Traditional FPGA Debugging Methods

Debugging in FPGAs has been difficult since day one. Unlike simulation where designers can see any signal at any time, signals when mapped to a FPGA may be difficult to locate or even worse optimized away. Even after you identify where the signal is, it may be difficult to capture the time period in which you would like to observe that signal as the FPGA runs at real speed and you cannot continuously capture and store the waveform of that signal. Therefore, some sort of triggering and waveform storage circuit is needed to perform debugging in an FPGA. There are two popular approaches today: external logic analyzer, and internal logic analyzer.

External Logic Analyzers

Let's first take a look at the use of external logic analyzers that have been in use for years. Popular external logic analyzers today are from Agilent and Tektronix and can sample at GHz frequency and store GBs of waveforms. External logic analyzers have the ability to store large amounts of trace data but for the data to be useable, the data needs to be taken off the chip, which can be a difficult task. The signals, or probes, designers want to observe need to be sent to FPGA I/O pins to connect to a logic analyzer. Since some probes may be buried deep

inside design hierarchy, it may be time-consuming to get the right probes to the top of the design.

Physically, you also need some kind of adapter card that connects the FPGA I/O pins to the logic analyzer header. For example, Agilent logic analyzers use a 38-pin Mictor connector. Most off-the-shelf FPGA boards do provide optional daughter cards that can connect the FPGA I/O pins to the 38-pin Mictor connector. If you are building your own (RYO) board, then you should reserve a set of pins to connect to the Mictor connectors if you choose to have the ability to observe through a logic analyzer.

The biggest drawback for the use of external logic analyzers is actually the limited number of probes you can observe at a time since there are only a limited number of FPGA I/O pins you can use for debug. In most designs, the majority of FPGA I/O pins are used for external target interfaces or used as interconnects to other FPGAs if more than one FPGA is used. Therefore, reserving a large amount of pins for debugging through an external logic analyzer may not be feasible. Multiplexing the probes to I/O pins can solve the limited pin issue but is almost never used since external logic analyzers need to capture data at real speed and also need to support de-multiplexing on the logic analyzer side.

Once connected, the external logic analyzer is used to set up triggering and data capture conditions. Triggering is typically done using a state-machine technique whereby values are specified for a signal and then either the data is captured or a different condition is sought after on another state. The signals remain static while the conditions can be altered at any time. Trace memory using an external logic analyzer is rather large therefore memory can afford to be wasted trying to find trigger conditions that are close to desired observation points. The advantage of an external logic analyzer is that it can sample at high frequency (in the GHz range), at high accuracy, and support very complex triggering conditions. Today, some designers still prefer to use an external logic analyzer because of these advantages as well as the feeling that debugging needs to be seen on real equipment, not just through a software tool.

Internal Logic Analyzers

Internal logic analyzers such as Altera's SignalTap or the integrated logic analyzer (ILA) in Xilinx's Vivado utilize cores embedded into the design whereby the trigger conditions are set using a GUI in software on a PC through a JTAG interface. The captured data is transferred to the PC where it can be viewed and analyzed. The internal logic analyzers provided by the FPGA vendors are tightly integrated with their FPGA place and route tool making them easy to learn and use.

However, trace data needs to be stored in the FPGA internal block memory before a triggering condition is met and therefore they can only achieve very limited width and depth. Often, you have to choose between limiting the amount of memory you can have for your design versus allocating some memory for debugging. When a triggering condition is met, the logic analyzer stops storing new waveforms in the memory and shifts out current memory content through JTAG. The process can be slow if trace data is large. The probes must be statically defined and trigger conditions can be dynamically changed during debug just like with external logic analyzers. Most internal logic analyzers only support probing at the gate level so signal names may have changed or may have even been optimized away. Since probes are static, to change probes you usually need to re-compile the design.

Some third party internal logic analyzers do support RTL probing which can improve the user experience. They also provide more advanced triggering and analytic features that allow you to get meaningful data from limited amount of waveform storage memories inside an FPGA.

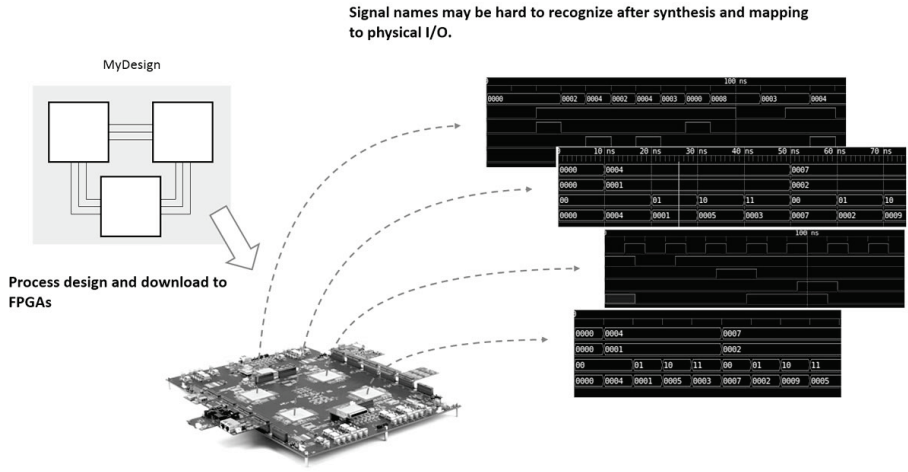
Even though internal logic analyzers supplied by the FPGA vendors have some limitations, they are still by far the most popular tools used for FPGA debugging today. This is due to their relative low-cost and tight integration with the FPGA vendors' own place & route tools.

Multi-FPGA Debugging Methods

External and FPGA Internal Logic Analyzers are better suited for debugging a single FPGA. Although external logic analyzers can probe signals simultaneously from a multi-FPGA environment, the limited

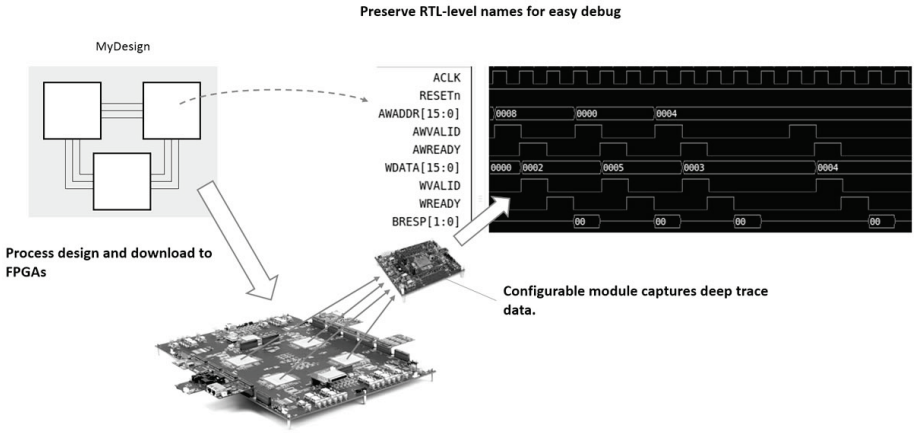
number of probes available makes debug inefficient. Debugging only one FPGA at a time in a multi-FPGA environment makes effective debug of the design significantly more difficult, time-consuming, and error-prone. These logic analyzers can only provide a subset of the picture in which to debug and don't have the trace depth for delving into the behavior of a multi-FPGA design. Debugging only a piece of the design at a time can lead to errors in other parts of the design as the bugs are fixed. The difficulties involved with this type of approach are illustrated in the diagram below.

Image FG-22: Debugging multi-FPGA prototypes means examining waveforms for each device separately



What's needed is a holistic approach to debug for multi-FPGA platforms to ensure design behavior is not affected as bugs are corrected because RTL-level signals and module names are maintained throughout. With the use of a configurable external module, multi-FPGA debug will also allow for the detection of very hard to find corner case bugs because of the deep trace depth that can be achieved.

Image FG-23: Benefits of using a multi-FPGA debug approach

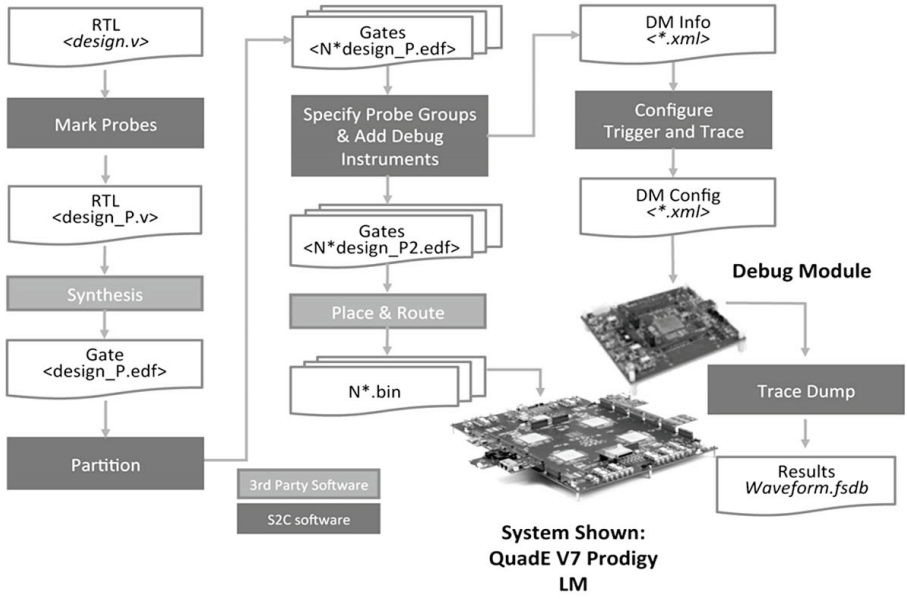


To understand this more, let's take a look at how multi-FPGA debug actually works. In the diagram below you can see that you must first mark probes at the RTL level so the probes are maintained throughout the compile flow. There should be no limit to how many probes you can mark as this simply tells the synthesis and partition tools to retain the RTL names for probing. After a design is partitioned to multiple FPGAs you can start selecting the signals you would like to probe in each FPGA. Multiple groups should be supported so you can see thousands of signals from any FPGA. Debug instrumentation is then added to each FPGA for FPGA place-and-route. Note that since the triggering logic and waveform storage are performed using an external module, the debug instrumentation in each FPGA consumes very little resources inside your design FPGA.

After the multi-FPGA design is compiled and downloaded to FPGAs, you can now set your trigger conditions and the information is uploaded into the dedicated debug module hardware. When you start running your design, the debug module will capture and store the waveforms continuously from multiple FPGAs in external DDR memory. The communication bandwidth between the debug module to each FPGA needs to be high in order to trace wide waveform at high speed. Then, when a trigger condition is detected by the debug module,

the DDR3 memory content is sent to the host computer for analysis via a high speed PC port such as Gigabit Ethernet. The waveforms in VCD or FSDB format can then be debugged using popular waveform debug tools such as Synopsys Verdi. Signals from multiple FPGAs can be viewed in a single waveform window.

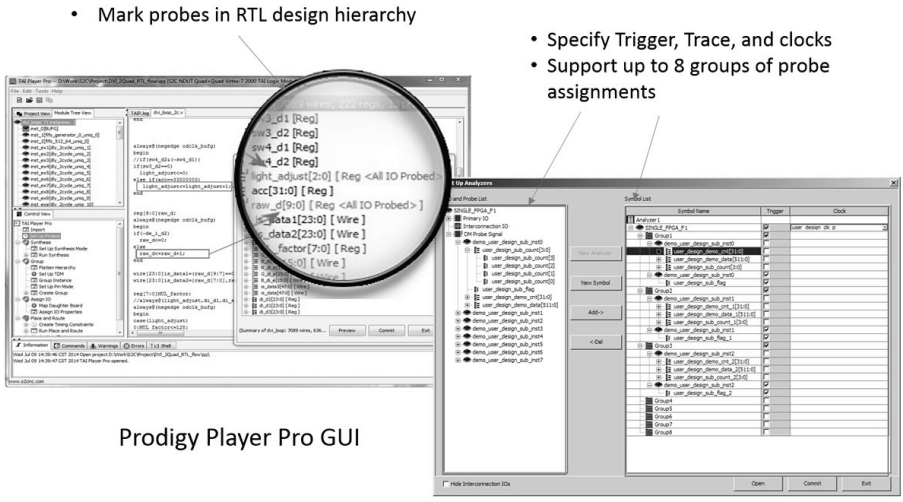
Image FG-24: Multi-debug flow



The use of a separate debug module of this nature allows for deep trace with a large number of RTL-level probes, the use of minimal FPGA resources to avoid design impact, and system-level debugging across the entire SoC design. An example of this device is the Prodigy Multi-Debug Module from S2C. The Prodigy Multi-Debug Module supports up to 32 FPGAs at a time with 16GB of DDR3 trace buffer and can utilize up to four 5GHz transceivers to capture waveforms from each FPGA to the debug module. The use of Gigabit Transceivers allows large amounts of data to be transmitted at high frequency. General purpose I/O pins are not occupied by debugging so they can be used for interconnecting between FPGAs and external interfaces. Deep trace is achieved with 16GB or trace memory with actual trace depth dependent on the number of signals that are probed. S2C also provides

an easy-to-use GUI (as shown below) that allows you to mark probes in RTL before synthesis, quickly locate probes after design partitioning, and select probes before FPGA place-and-route.

Image FG-25: Mark probes in Prodigy Player Pro



There are variations of the above multi-FPGA debug approach. There are solutions that use cascading instead of distributed topology to collect trace data from multiple FPGAs. Cascading topology means that trace data from multiple FPGAs needs to be collected to a single FPGA through potentially many FPGAs before transmitting to an external debug module for storage. Cascading topology is easier to implement in hardware but the large debugging data going from FPGA to FPGA can create a bottleneck that in turn reduces the amount of probes that can be seen at any one time and decreases the speed at which they can be captured. Distributed topology, on the other hand, sends debug data continuously from every FPGA directly to the external Debug Module. The hardware is more difficult to implement but this method can maximize the number of probes that can be seen at the same time as well as produce faster capture speed.

Other advanced FPGA Debug Techniques

One technique to increase trace depth is to compress the waveform being temporarily stored in the memory. The compression needs to be

lossless and meet the performance required to continuously store incoming waveforms from multiple FPGAs. This waveform compression technique has already been developed in some third party FPGA debug tools to address the trace depth issue.

Hardware assertions in FPGAs is another interesting area that can make debugging FPGAs easier. Instead of continuously capturing large amounts of data and looking for trigger conditions to shift out the waveform for analysis, you can embed the conditions that you are looking for together with your design in the FPGA. When such conditions occur, you receive high level messages such as: a memory is full, a bus has a contention, the CPU is at a specific state, and more. Nevertheless, most tools today do not generate synthesizable assertions that can be mapped in FPGAs so designers will have to write and embed assertions in the FPGAs themselves.

Finally, some newer FPGA families now support register and memory readbacks and even allow you to set the register and memory content. The readback feature enables you to access all nodes inside an FPGA at a given time. However, to access that information you would need to stop the design clock to shift out the register data. Therefore, this feature can only be used when the design is run in a controlled clock environment and not really useful when running FPGA prototypes in or close to real time speed. In addition, just by taking a snapshot of what's inside an FPGA cannot solve the issue/bug you are looking for. Are you taking the right snapshot and how many snap shots do you need to take? Readback data is often shifted out through a JTAG port which is also very slow when dataset is large. FPGA vendors do have plans to improve this feature by allowing shifting out the readback data without stopping the clock as well as using a faster protocol to shift out the data. We hope to see better support of this feature from Xilinx and Altera and also a complete environment that allows designers to quickly see what they are looking for.

Exercising the Design

Now that we've covered the components of an FPGA prototyping flow and how to maximize the available technologies within the flow, we

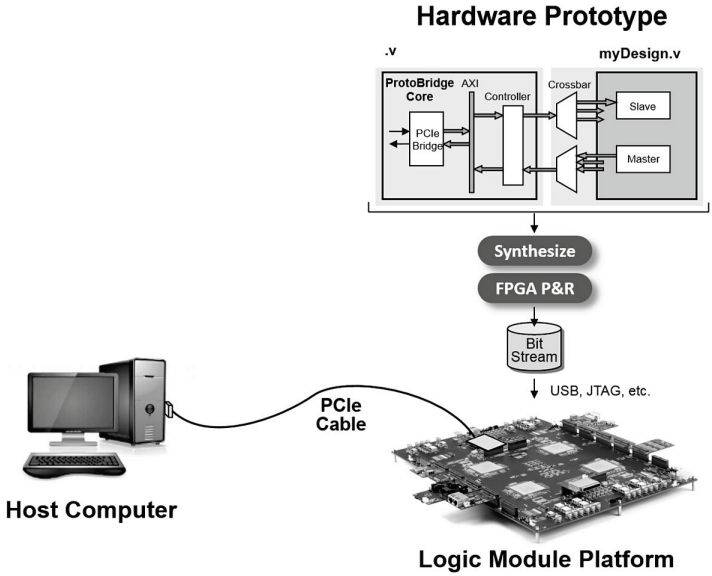
can move on to the final topic of exercising the design and the methods that are used to perform actual testing.

Often, simulation and/or emulation are the first things we think of when talking about testing the design. However, it is worth remembering that neither simulation and emulation can run at speed or close enough to actual speed for in-circuit testing. Although emulation can allow you to get up and running quickly, it is very costly. FPGA prototyping is the most practical method for doing complex and thorough pre-silicon tests as well as early software development. We will now cover two popular test methods using FPGA prototyping.

In-Circuit Testing

The common definition of In-Circuit testing is to connect your design to real targets intended by your final chip for real world tests before you have the silicon. FPGA prototyping, which can operate at or near final chip speed. allows you to simply build those system target interfaces either directly on the FPGA boards or through the use of daughter cards.

Image FG-26: In-circuit testing



The use of daughter cards allows an FPGA prototyping system to be flexible and scalable. The main FPGA prototyping system can be used to connect to different in-circuit test targets depending on the design requirements or can connect to other FPGAs for expansion with the unused I/O pins. You can either use third party daughter cards that are usually designed specifically for an application or build your own daughter cards that fit your application requirements exactly.

Building your own daughter card(s) for FPGA prototyping is actually not a bad idea since it is unlikely you'll find a daughter card that meets your various in-circuit test requirements exactly. Also, the complexity of building a daughter card is a lot less than building the main FPGA board and you may be able to reuse the daughter card for your next project if there are no significant changes in the chip interfaces. However, by using commercial daughter cards you can still save precious engineering time and resources in addition to reducing risks. Many of today's chip interfaces use industry standards such as USB, PCIe, Ethernet, DDR, and others, and there are usually commercial daughter cards available that will meet your requirements.

So what are the considerations for choosing daughter cards or the FPGA prototyping systems that will hold the daughter cards? Well, the FPGA prototyping system you select should have abundant unused I/O pins on I/O connectors in order to use daughter cards. Naturally, the most important methods for evaluating a prototyping system in this regard is to examine the type of I/O connectors the system uses, how the pins are defined, what features the FPGA board supports for using the daughter cards, and the availability of different types of daughter cards so you do not have to build everything on your own. The list below is a good starting point for you to evaluate if a system is ideal for doing in-circuit testing through reusable daughter cards:

- What type of connectors are being used on the FPGA board and how many daughter cards are available for that connector?
- Does the type of connector support high speed I/Os such as LVDS and multi-GHz transceivers?
- How many connectors are available on the FPGA board so you can connect to different targets at the same time?

- How many I/O pins are on the connector and how are the I/O pins/banks optimized on the connector?
- How will the daughter cards get power?
- What are the I/O voltages that are supported on the connector so you can pass power from the FPGA board to the daughter cards?
- Are there physical limitations on the size of the daughter card and how reliable is the daughter cards physically?
- Are the daughter cards testable?
- For an off-the-shelf daughter card, does the vendor provide tests?

One of the popular connector standards for daughter cards is FMC. It has been used by Xilinx evaluation boards for many years and comes in 2 flavors: HPC, which has a higher pin count with transceivers, and LPC, which has fewer pins and no transceiver support. There are many off-the-shelf daughter cards that are based on the FMC standard thanks to the popularity of the Xilinx evaluation boards. However, FMC is not an ideal choice for high-end prototyping systems because the connector is physically too big with too many I/O pins per connector so you will not have many I/O connectors on a single board. The I/O connectors are used for daughter cards as well as interconnecting multiple FPGAs together. Also, FMC connectors have poor interconnection cable support. As a result, most advanced prototyping systems define their own connector standards to solve the deficiency of the FMC connector.

As an example, S2C's Prodigy Connector is a compact, high-performance, 300 pin connector that can support the running of multi-GHz transceivers. It supports 3 full FPGA I/O banks and all traces from the same I/O connector have the same length. The Prodigy Connector has a matching Prodigy Cable that can connect 2 Prodigy Connectors with pin 1 matching pin 1. The Prodigy Connector supplies 2 voltages from the FPGA board to the daughter board: 3.3V and VCCIO voltages. In addition, there is an I/O voltage detection function so when the wrong voltage is input into the daughter card from the FPGA board, the power will be automatically shut off.

S2C has dozens of daughter cards including processors, embedded, multimedia, and memory, that can run on Prodigy logic modules, and

many have reference designs to run out-of-the-box. S2C also provides an FMC to Prodigy Adapter Module so FMC daughter cards can also be used on Prodigy logic modules.

Daughter card modules can be reused across multiple configurations of FPGA prototypes and among multiple projects/locations. Employing these daughter cards within a complex environment is simplified with the use of auto-detection technology that indicates the presence of a specific daughter card and allows for the configuration of design data related to the specified daughter card. S2C Prodigy Daughter Cards support this advanced feature.

Some vendors also provide daughter card customization services targeting special system interface requirements that cannot be met by off-the-shelf solutions. These services usually entail collaboration to understand your needs in creating a detailed specification of the daughter card. Once the specification is complete, the daughter card is designed, developed, and then thoroughly tested. A reference design is typically included that will help in the actual use of the daughter card on your design. These services can be extremely beneficial if you are working on a tight time-to-market schedule and cannot support the engineering resources internally to create your own.

In-circuit testing using FPGA prototyping allows you to obtain detailed system performance metrics and proof-of-concept results quickly and easily. Detecting issues such as functional definition errors or system-level timing errors are just a couple of examples of the system data that can be collected. FPGA prototyping offers uncompromised flexibility with support for industry bus standards such as PCIe, USB, Ethernet, and more, simplifying the testing associated with these standards.

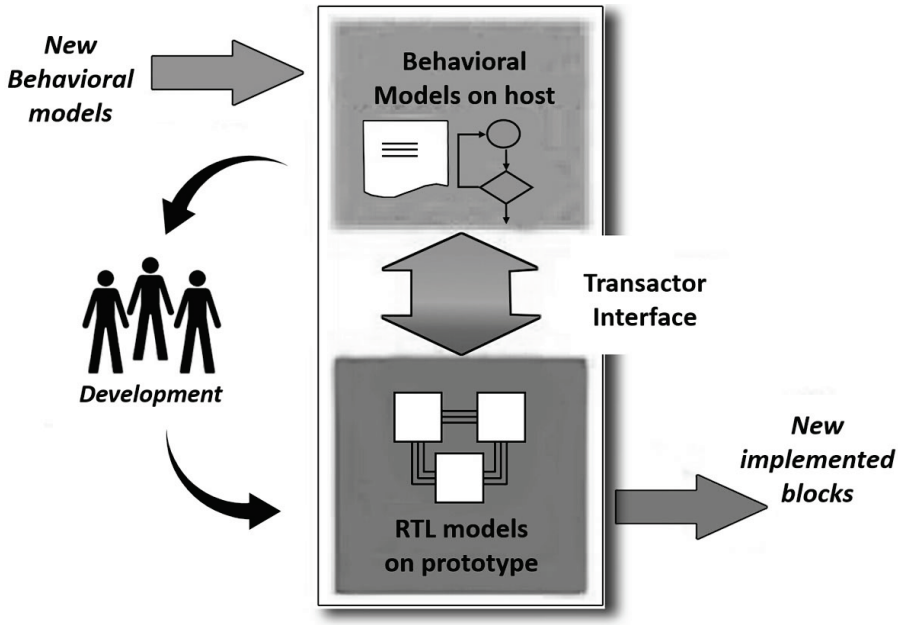
Hybrid Prototyping

In-circuit testing is probably the most important reason for doing prototyping today. But in-circuit tests are usually based on un-constrained random tests, which don't always ensure complete test coverage. Using a transactor interface allows test cases developed in simulation to be run directly on the prototype making these tests instantly available and insuring compliance. Moreover, these tests can

be easily extended to large data sets, providing coverage for corner cases and hard-to-find bugs.

The addition of a transactor interface to an FPGA-based prototype facilitates development of new systems in interesting ways. As behavioral models are introduced, architectures become refined and block functionality determined. These blocks are eventually defined and implemented as part of the new system. But blocks that are defined and rendered in RTL become IP for the next generation of systems, allowing the cycle of development to repeat. In this way, an FPGA prototyping platform becomes the engine of system advancement.

Image FG-27: Behavioral models and transactors

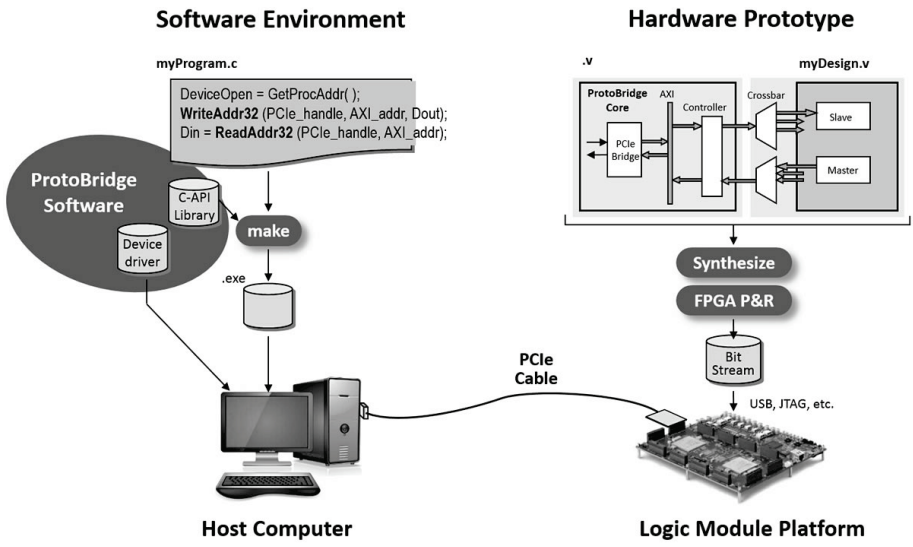


FPGA-based prototyping is well-suited for designs fully rendered in RTL and that can be mapped to an FPGA. However, many designs may not be completely mapped to an FPGA and may only be partially available as behavioral models in descriptions such as C++ or SystemC. In these cases, transaction-level interfaces play a critical role in being able to bridge the abstraction level between behavioral models and live hardware. These transactors offer a way to communicate between

software running on a host and an FPGA-based prototyping platform that often includes memories, processors, and high-speed interfaces.

S2C's unique patent-pending Prodigy ProtoBridge™ System is a solution that allows for just this type of high-speed communication. ProtoBridge supplies a transactor interface between a software program and the world of AXI-compliant hardware. There are two key parts to this: an AXI-to-PCIe bridge that connects to a host computer, and a C-API that communicates to the design through the bridge. The software-to-AXI transactor offers new flexibility to designers building ARM-based systems. Coupling this to a PCIe interface supporting transfer speeds up to 1000 Mbytes/sec provides a perfect development platform for data-intensive applications.

Image FG-28: S2C Prodigy ProtoBridge



A system like this allows designers to maximize the benefits of FPGA-based prototypes much earlier in the design project for algorithm validation, IP design, simulation acceleration, and corner case testing. A prototype combined with a transactor interface makes a range of interesting applications possible throughout the design flow.

Coding with FPGA-based Prototyping in Mind

We've walked you through the steps for effectively implementing an FPGA prototyping methodology. We discussed how to choose the prototyping system that fits your design requirements and how to extend and scale that system as your design needs change. You should also now have a good understanding of the best practices for partitioning, debugging, and exercising your design.

What about taking FPGA-based prototyping to the next level? How can you maximize your FPGA prototyping experience? The next step in the process is to code your design with FPGA prototyping in mind. This topic was explored briefly by Mon-Ren Chene in his foreword to "Prototypical" and will become a key launching point for the future of FPGA-based prototyping. We look forward to providing you with the knowledge you need to further your FPGA prototyping goals.

About the Authors

Daniel Nenni has worked in Silicon Valley for the past 30 years with computer manufacturers, electronic design automation software, and semiconductor intellectual property companies. He is the founder of SemiWiki.com (an open forum for semiconductor professionals) and the co-author of "Mobile Unleashed: The Origin and Evolution of ARM Processors in Our Devices" and "Fabless: The Transformation of the Semiconductor Industry." Daniel is an internationally recognized business development professional for companies involved with the fabless semiconductor ecosystem.

Don Dingee has been in the electronics industry since 1983, with experience spanning engineering, sales, marketing, and web development roles. He is the co-author of "Mobile Unleashed: The Origin and Evolution of ARM Processors in Our Devices." Currently Don is a product strategy consultant and content marketing freelancer working with firms on embedded, mobile, and IoT applications. Don also blogs on SemiWiki.com.

PROTOTYPICAL

“Our commitment to customers has been to push the limits of what FPGA prototyping can do to make designing easier, faster, and more efficient.”

- Mon-Ren Chene, CTO of S2C

PROTOTYPICAL looks at the history of FPGA-based prototyping and its role in SoC design, along with a look forward at how it can help application segments such as automotive, the IoT, and wearables. A practical Field Guide is included addressing questions designers typically have, the issues they often run into, and technical approaches to solve them.

S2C
FPGA PROTOTYPING SOLUTIONS

DAC 2016 | Austin, TX | June 5-9