Xilinx Adaptive Compute Acceleration Platform: Versal™ Architecture

Brian Gaide, Dinesh Gaitonde, Chirag Ravishankar, Trevor Bauer
bgaide@xilinx.com,dineshg@xilinx.com,chiragr@xilinx.com,trevor@xilinx.com
Xilinx Inc.

ABSTRACT

In this paper we describe Xilinx's Versal™ Adaptive Compute Acceleration Platform (ACAP). ACAP is a hybrid compute platform that tightly integrates traditional FPGA programmable fabric, software programmable processors and software programmable accelerator engines. ACAP improves over the programmability of traditional reconfigurable platforms by introducing newer compute models in the form of software programmable accelerators and by separating out the data movement architecture from the compute architecture. The Versal architecture includes a host of new capabilities, including a chip-pervasive programmable Network-on-Chip (NoC), Imux Registers, compute shell, more advanced SSIT, adaptive deskew of global clocks, faster configuration, and other new programmable elements as well as enhancements to the CLB and interconnect. We discuss these architectural developments and highlight their key motivations and differences in relation to traditional FPGA architectures.

KEYWORDS

ACAP, Versal, FPGA, Stacked Silicon, SSIT, Adaptable Compute Acceleration Platform, Math Engine, NoC, FPGA Architecture, FPGA CAD, Xilinx

ACM Reference Format:
Brian Gaide, Dinesh Gaitonde, Chirag Ravishankar, Trevor Bauer. 2019. Xilinx Adaptive Compute Acceleration Platform: Versal™ Architecture. In *Proceedings of The 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '19).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3289602.3293906

1 INTRODUCTION

It is well known that the benefits of process technology scaling are reducing [1]. The benefits of a new technology node alone are often insufficient to justify the development costs of a next generation device, forcing more aggressive innovations at the architectural and system levels [2, 3]. With the recent explosion of data and surge of machine learning and AI applications, the needs for compute have also been increasing. Due to the high costs of sub-16nm technology nodes and the continually changing requirements of these applications, developing ASICs for these markets is challenging. By virtue of their configurable nature, field-programmable gate arrays excel in applications with varying workloads and requirements, circumventing the economic challenges of heterogeneous compute platforms with reconfigurable hardware [4]. FPGA platforms have recently been deployed on the cloud to democratize these systems at a larger scale [5–8].

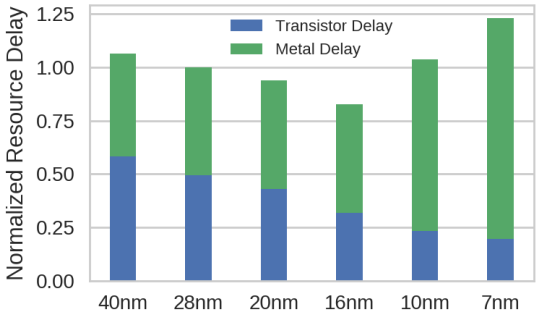


Figure 1: Metal and Transistor Delays For a Quad Routing Resource Across Different Technology Nodes (normalized to total delay at 28nm)

Many compute intensive solutions today operate in a thermal envelope and are thus power limited. Although power and delay per operation drop with technology scaling, they no longer drop at a rate that satisfies exponentially increasing compute demands. Metal resistance is another critical challenge that has worsened with technology scaling [9]. Although wire distances shrink with lithography, wire cross-sectional area shrinks quadratically, resulting in a net increase in resistance each generation. Hence, even though transistor delays continue to decrease with smaller transistors, total path delays may not. In Figure 1, we show the minimum wire pitch delay of an interconnect routing resource over several technology nodes assuming that the physical distance of a given logical span also scales. Despite the physical distance shrink and transistor delay speed up, total delay actually increases with more advanced process nodes. Hence, we are forced to use thicker metal with lower resistance to reduce wire delays. As technology scales, metal resources therefore become more expensive and architectural changes need to be made to use them more efficiently.

One of the hurdles to greater adoption of traditional FPGA architectures is ease of use. Recently, there has been a drive towards software solutions to improve the user abstraction level to interact with FPGAs [10]. However, wide-spread use of re-configurable hardware without the requirement for expertise remains elusive.

FPGA '19, February 24–26, 2019, Seaside, CA, USA

ACM ISBN 978-1-4503-6137-8/19/02…$15.00
https://doi.org/10.1145/3289602.3293906

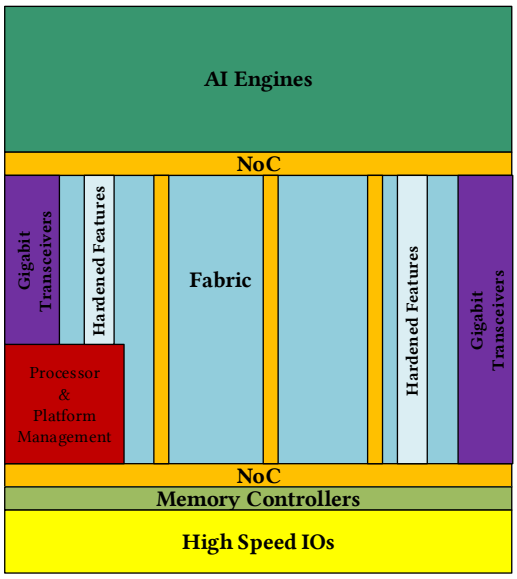


Figure 2: Versal Representative Device Floorplan

In this paper, we present a new class of re-configurable devices called the Adaptive Compute Acceleration Platform (ACAP), invented by Xilinx to provide a solution for the compute and communication needs of modern applications. We describe the 7nm based Versal™ Architecture, which is a new re-configurable platform architecture that solves the economic, technological, and ease of use challenges mentioned above. We provide a general overview of the architectural changes with some experimental results. We limit the scope of this paper to the re-configurable fabric and the related subsystems. In Section 2, we describe the features of the Versal Architecture at a high level and distinguish it from a traditional FPGA. In Section 3, we describe the programmable fabric and the various enhancements made to tackle the economic and technological challenges. Section 4 describes a 4th generation stacked silicon interposer technology with new capabilities for multi-die devices. We briefly describe the clocking structure, hardened Network-On-Chip (NoC) and the Configuration system in Section 5.

2 FLOORPLAN

In Figure 2, we show a representative device floorplan for Versal architecture based ACAP. The fabric portion of the device is similar to a traditional FPGA, including resources such as LUTs, flip-flops, DSPs, BRAMs, and the relatively recently introduced UltraRAMs [11], all arranged in a columnar topology. Changes to these blocks are described in Section 3.

What makes ACAP unique is that it hardens all the necessary platform management functions and separates them from the FPGA core logic. The processor and platform management controller occupy the lower left region of the chip. The adjacency of the Processor Subsystem (PS) to Gigabit Transceivers (GTs), memory controllers, and the NoC enables those blocks to be used together without any of the fabric being programmed.

GTs can occupy the left and right edges of the fabric regions. Note that high speed IOs in Versal now run along the bottom and, optionally, top edges of the die. Integrated with those IOs are hardened memory controllers to interface with off-chip memory such as DDR and HBM.

Across the top of this example Versal architecture based floorplan is an array of AI Engines designed to accelerate math intensive functions for applications including machine learning and wireless.

Finally, the chip pervasive hardened network-on-chip (NoC) augments the traditional fabric interconnect and enables a new class of high speed, system level communication between the various heterogeneous features, including the PS, DDR, AI Engines and FPGA fabric.

2.1 Hardened Features

A key benefit of FPGAs is adaptability provided by configurable fabric and other on-chip compute and memory blocks. One recent trend has been the use of FPGAs for accelerating various functions that were traditionally implemented on CPUs. In these environments, most of the peripherals are standardized. These standard application components like memory controllers and PCIE generally don't benefit from that adaptability enough to justify their cost in soft fabric.

A substantial portion of the FPGA fabric can be spent on this functionality as illustrated in Figure 3, which shows the placement of shell logic that performs static platform level functionality when an instance of AWS F1 FPGA [5] is invoked. Moreover, design effort is spent in ensuring that these platforms meet timing and do not interfere with the place and route of the actual accelerator cores. This logic implements memory controllers, PCIE connectivity and provides ports for rest of the compute to interface with the memory controller. Additionally, this solution requires loading a minimal state configuration prior to loading any user specific functionality.

In contrast, the memory controllers, PCIE and associated interfaces are all hardened and directly connectable in the Versal architecture. As a result, once powered up, the FPGA boots up in a state ready for use without the need to bootstrap any soft logic first. This reserves the FPGA fabric purely for user functionality

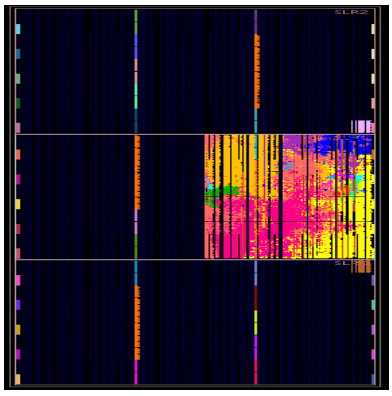


Figure 3: Static Platform Logic for AWS F1 Instance based on UltraScale™Architecture

and also helps non-traditional FPGA users productively use the ACAP without detailed device knowledge.

Some features are ubiquitous enough to include on all devices, such as hardened memory controllers and NoC infrastructure. However, other blocks are market specific and not always necessary. Therefore, Versal architecture comprises a framework that enables swapping different features in and out of different devices. For example, some devices have AIEs along their top edge. Other devices may have IO or fabric along their top edge. Similarly, some devices have HBM interfaces in place of DDR IO and controllers. A-to-D converters can replace GTs, and a variety of smaller hard IP blocks (forward error correction, MAC blocks, Interlaken, PCIE, etc.) can occupy slots within the fabric array. In this respect, the Versal architecture enables a platform that continues the trend towards enabling families of domain specific devices [12, 13].

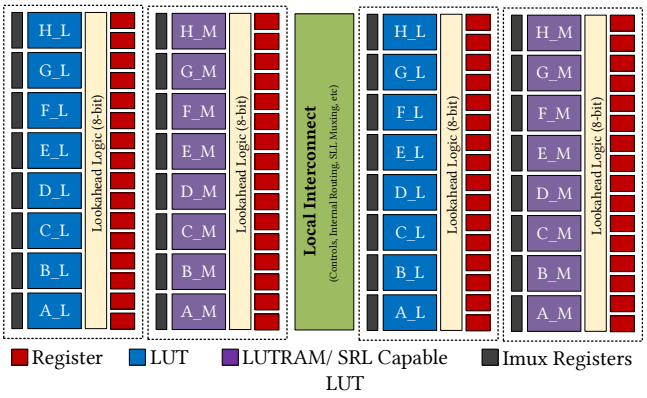


Figure 4: Versal CLB Block Diagram

2.2 Perimeter IO

Since Virtex-4, Xilinx FPGAs have had columnar IOs. There are several advantages to columnar IOs, including tight integration with the fabric and area efficiency. However, multiple technology trends have led to perimeter IO being more appropriate for the Versal architecture. IO cells don't tend to shrink with Moore's Law. Similarly, as noted, the cost of using long metal wires has increased. As a result, over the past generations of FPGAs, the increasing interconnect delays and clock skew incurred by metal crossing over large IO columns have led to software tools partitioning designs across these boundaries. Additionally, IO package trace breakouts from the die interior can be challenging and performance limiting. As a result, the implementation of perimeter IOs enables higher performance IOs and less fabric disruption.

2.3 Regularity

Overlays have become more powerful and popular among FPGA users [14–16]. Use of a domain specific overlay has substantial productivity benefits. Since overlays are very structured designs, using analytical techniques to implement them lets tools extract more performance from FPGAs rather than traditional generic RTL based implementation tools. One way to make overlays easier to implement is to have very regular patterns of fabric columns. However, in a traditional FPGA architecture, even perfectly repeated columns of fabric don't enable a perfect "stamp and repeat" of user IP, because communication interfaces to those blocks would not be identical. Each repeated IP would need to be uniquely configured to enable routing connections to the external environment.

Versal architecture based devices are the first class of modern devices from Xilinx that offer a high level of fabric regularity. The composition of clock regions repeats at regular intervals across each device. This permits two significant productivity improvements. It permits relocatability of IP in both the X and Y directions without having to do complete reimplementation. Moreover, if each IP communicates with the rest of the design over the NoC (discussed in Section 5.2), even global communication interfaces do not have to be reimplemented when an IP is relocated. Thus, Versal enables the possibility of importing pre-implemented placed and routed IP and replicating the same IP at different locations. Utilizing this regularity also can enable hierarchical place and route run times to be several times faster than flat implementation flows.

3 PROGRAMMABLE FABRIC

Compared to UltraScale™, the Versal architecture has some significant differences to its fabric (i.e. CLB, Interconnect, DSP, BRAMs, URAMs, etc.). Various design choices were made to increase device capacity and facilitate more complex designs in a technology with metal resistance and cost challenges. We briefly describe the design choices and present some experimental results in this section.

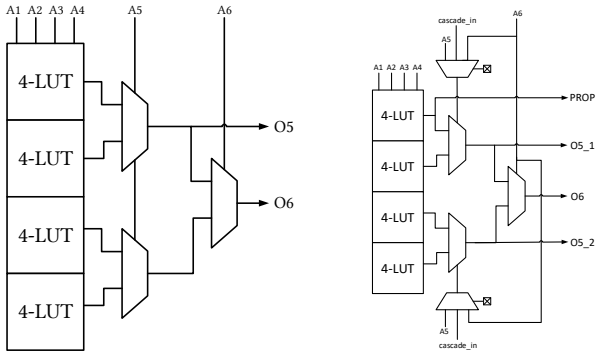
3.1 CLB

The CLB in the Versal architecture contains 4 times the number of LUTs and registers (32 LUTs and 64 registers) as the UltraScale CLB (8 LUTs and 16 registers). The components are noted in figure 4. Internals of the CLB, such as wide function muxes, carry chain, and internal connectivity were redesigned to increase total device capacity by reducing area per utilized logic function. A dedicated local interconnect structure resides within each CLB to support more versatile intra-CLB connectivity. By enlarging the CLB to include 4X the number of logical elements, we subsume a significant fraction of local nets internally, thereby reducing global track demand. Each of these enhancements are discussed in detail below.

3.1.1 Versal-based Look-Up Table. Compared to UltraScale, the look-up table (*LUT*) in Versal is enhanced to increase effective packing density and functionality. The UltraScale 6-input LUT has two outputs and it can implement either any 6-input function or two independent functions of up to 5 unique inputs. As shown in Figure 5, the 6-input LUT in Versal has an additional output, O5_2, and some circuitry on the second fastest input. This enables us to pack two independent functions of up to 6 unique inputs.

One goal of implementation tools is to maximize device utilization by packing logic into fewer logic elements. If two LUTs are placed close by, merging them into one physical LUT frees up additional resources while minimally perturbing the natural placement of the design. Figure 6a shows packing density improvements on a design suite of customer designs. It illustrates the number of legal LUT merging candidates within a given radius for Versal, normalized to UltraScale. For example, if we permit merging of LUTs

separated in placement by less than or equal to a distance of 5, in Versal we find 21.5% more candidates to merge than in UltraScale, thus increasing logic per unit area accordingly.



(a) UltraScale 6LUT

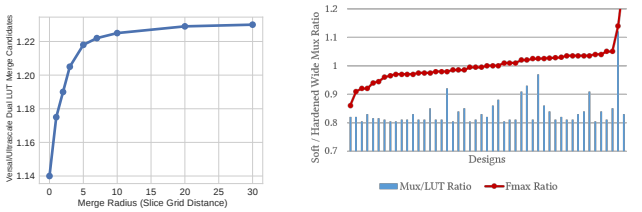
(b) Versal 6LUT

Figure 5: 6LUT Comparison between UltraScale and Versal

Also new to Versal is a dedicated, fast LUT to LUT cascade path which improves timing for paths with multiple levels of logic. The vertical cascade path daisy chains adjacent LUTs together by muxing into the second fastest LUT pin. Synthesis, placement, and packing tools can take advantage of the cascade connectivity to create fast and efficient macros implemented with the LUT and cascade path.

An additional output named "prop", in Figure 5b is added to implement carry lookahead functionality, which we describe in section 3.1.3

3.1.2 Wide Functions. We chose to remove the dedicated wide function muxes that existed in UltraScale, in favor of targeting more versatile LUTs for this function. Although dedicated wide function muxes enable fast, efficient, and compact implementations, they lack placement and routing flexibility. For designs with many wide functions, a synthesis tool has two options. It could use the dedicated wide function logic in the CLB or synthesize the same logic using regular LUTs. Wide functions implemented using dedicated logic result in large objects which have far more pins than a typical LUT - a hard 32 input mux will have around 37 inputs and one output. Placement algorithms do not perform well when they are asked to place instances with widely varying pin counts. As a result, it was observed that using the hardened wide function often features resulted in worse overall performance, worse wirelength, and tougher to route designs. By implementing wide functions in LUTs, tools can build muxes that span multiple slices, spread the mux out if necessary, and support a greater variety of topologies (priority muxes, unbalanced or sparse trees, etc.). The global speed advantages of a more flexible architecture offset the local speed advantages of the prior hardened solution. The effect of this modification on the worst case critical path of a suite of customer designs is shown in Figure 6b. The normalized geomean on this suite of designs does not change, with some outliers showing an increase in critical path up to 14%. We find that this is an acceptable tradeoff for reduced area usage in all designs



(a) Dual LUT Packing

(b) Soft vs Hard Wide Function Mux

Figure 6: Comparison between UltraScale and Versal

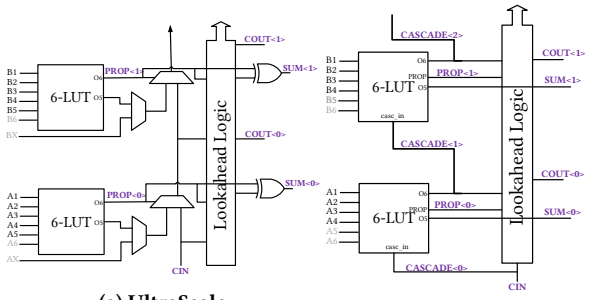
3.1.3 Carry Chains. A significant portion of UltraScale's dedicated carry logic is removed in Versal and absorbed into the LUT using the new cascade paths (see Figure 7). Dedicated carry logic area as a result reduced by a factor of 5 while keeping long carry chain speeds constant (comparing both at 7nm). Elimination of these dedicated carry signals also led to a reduction in CLB output muxing costs, since LUT outputs double as both generic LUT function and arithmetic function outputs.

3.1.4 Other CLB Changes. The Versal-based CLB has 25% fewer outputs per LUT compared to the UltraScale CLB. We converted that into additional connectivity for each CLB output at roughly cost parity and increased routabilty.

Each register in the CLB can be individually bypassed without affecting packing density. This also provides more output pin options for internal nets in the CLB to drive the interconnect, which improves routing flexibility.

Half of the LUTs in each CLE are capable of functioning as distributed memories or shift registers. Our analysis showed that deeper LUTRAM modes (128, 256 and 512 bit) were used in less than 1% of all instances, hence hardened support for these modes were removed in favor of a soft decoder based solution.

The combination of these changes make for a more streamlined and efficient CLB in the Versal architecture. Hard functions that were rarely used or whose use could be detrimental were removed and replaced by enhanced LUT connectivity.



(a) UltraScale

(b) Versal

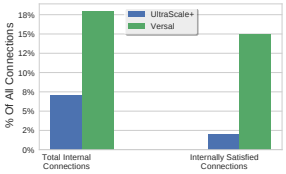
Figure 7: UltraScale vs Versal Carry Logic (8-bit structure, only 2 bits shown. Grayed inputs not used in carry mode.)

3.2 Local Interconnect

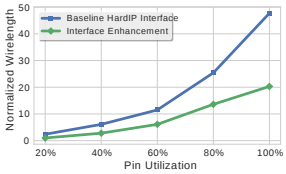
3.2.1 CLB Internal Routing. A significant fraction of nets have very localized sources and destinations. An additional layer of routing muxes exists within the Versal-based CLB to achieve more success connecting local internal nets without requiring the general interconnect. Since local routes are shorter and can be squeezed with tighter pitches onto fewer, lower level metal layers, the implementation cost of local routes is substantially less than global routes. With the internal CLB routing structure in Versal, almost every CLB output pin can drive every input pin on the same CLB using local routes.

We placed and routed a set of customer designs on both the UltraScale CLB and the Versal-based CLB and report the number of pin to pin connections contained within a single CLB. On average, we found that 18% of all pin to pin connections are theoretically intra-CLB connections for a CLB in Versal, contrasted to 7% within the smaller UltraScale CLB. In Figure 8a, this is captured as "Total Internal Connections."

In practice, not all theoretical connections are achievable. However, we demonstrate that roughly 83% of those theoretical connections are actually routed in Versal, compared to only about 28% of UltraScale theoretical connections. Figure 8a illustrates this. As the "Internally Satisfied Connections" shows, only 2% of all nets in UltraScale are successfully routed within a CLB compared to 15% in Versal, increasing internal net routing by a factor of almost 8X while only modestly increasing the cost of the CLB.



(a) Internal Routing Structure in Coarse CLB



(b) Enhanced Interface to Hard Macro Blocks

Figure 8: Benefit of Local Interconnect Enhancements

3.2.2 Interface Enhancement for Hard Macro Blocks. Similar to the Versal-based CLB, a local interconnect structure was added to the interface of every hard macro block (BRAM, DSP, PCIe, etc.) to enhance routability for highly utilized and congested designs. Every path from the general interconnect to a hard block goes through a layer of local interconnect muxing structures which reduces stress on the general interconnect.

To illustrate this, we experimented with a simple design consisting of registers driving the input pins of a hard macro block at various utilizations. We constrained the registers to be some distance away horizontally and let the placer choose their destinations within the constrained region. We routed the designs with and without the interface enhancements and report the normalized wirelength in Figure 8b. We saw decreased wirelength with the interface enhancements and the wirelength gap widens at higher pin utilizations, which indicates that the stress on global interconnect is reduced.

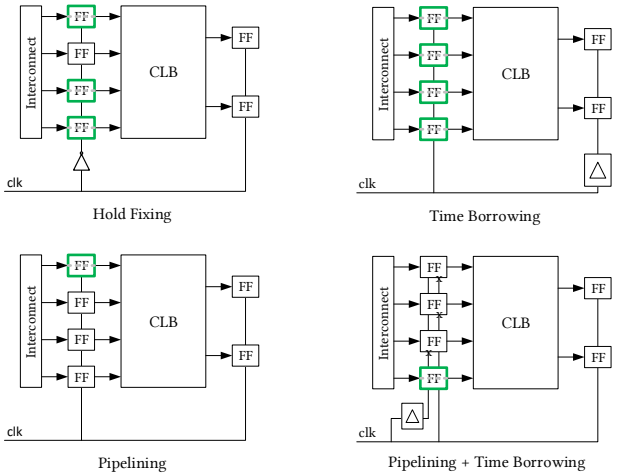


Figure 9: Imux Registers

3.3 Imux Registers

In order to facilitate easier implementation of high performance designs, we introduce a new feature to the fabric called the Imux Register Interface (IRI). Imuxes are the traditional name of input muxing to each block; Imux Registers are registers placed in-line with the Imuxes. These are flexible, bypassable registers on the input side of all blocks. Compared to an approach where registers exist on every interconnect resource [17] ("registers-everywhere approach"), Imux Registers are a more cost effective solution to increasing design speed while requiring far less design adaptation.

In the "registers-everywhere approach", there is a significant negative "up front performance tax" - a sea of interconnect registers increase the delay of every routing resource when not used in every route. Secondly, cost mandates that the resulting registers be a simple flop with no enable, reset or initialization functionality. Adding control sets increases the cost of registers themselves, plus the infrastructure necessary to connect control signals would be prohibitive. Thirdly, structural constraints such as the presence of sequential loops, pipeline balancing, and hold requirements typically constrain how aggressively one can use pipelining. Registers on every mux can simply not be used due to those constraints.

With IRI, the Versal architecture adds a substantial amount of registers but not beyond what are useable, and each register is far more capable. Each register is fully featured and supports clock enable, reset, and initialization, as well as time shifting capabilities. As with any added resource, the registers have a a cost in terms of area and delay. However, as we demonstrate, even without any user design modification, IRI delivers a performance increase (not decrease) for most designs.

The Versal architecture supports aggressive time borrowing techniques described in [18]. We support time borrowing at a much finer level than in [18]. UltraScale+™ enabled time shifting only for a clock shared by 30 CLB slices, whereas Versal has per CLB slice time shifting capabilities. The programmable delay lines per CLB can be used by itself or in concert with pipelining. Figure 9 shows the different cycle time reduction modes supported. Time borrowing enables the register to act as if it existed out in the interconnect

and still effectively bisect timing paths. Figure 12 illustrates this effect and other modes and how each one reduces cycle time.

Additionally, the IRIs support a "hold fixing" mode. Each Imux Register can be optionally clocked on the opposite clock edge, stalling data for half a cycle. Time borrowing is often limited in practice by how many registers share the same delay. Borrowing time on one register may cause hold violations for the other registers within the same cluster. By selectively using "hold fixing" mode, thus, more aggressive time borrowing is available. Figure 10 demonstrates how hold fixing mode works by shifting the data eye away from the clock edge to avoid indeterminacy.

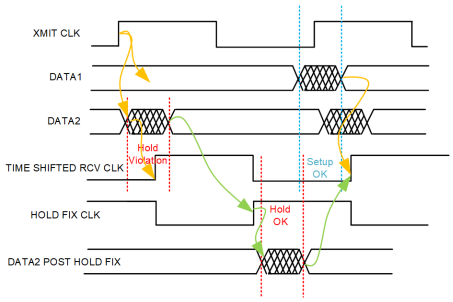


Figure 10: Hold Fixing Mode Timing Waveforms

Figure 11 illustrates the average timing impacts of the various modes across a suite of customer designs, limiting design modifications to pipeline insertion (no redesign of feedback loops). We used the same set of customer designs to also compare with our own “registers-everywhere approach”. Adding Imux Registers to the architecture incurs an initial penalty of about 4% due to (a) growth of block widths, which stretches horizontal routing resources, and (b) additional delay of bypassing the new register. However, the additional benefits to time borrowing more than offset the initial penalty. Note that these benefits do not require any design modification and has improved performance. Unlike our approach, a “registers-everywhere approach” incurs a larger initial penalty that cannot be recovered without pipeline insertion or retiming. If one assumes that designs were indeed allowed to be pipelined, and pipelining was the only technique applied, then the “registers-everywhere approach” has better performance than

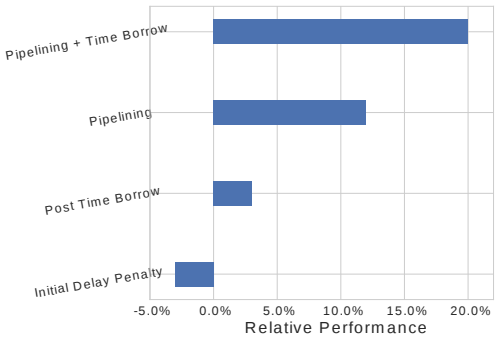


Figure 11: Imux Register Performance Gains

the IRI flops approach. However, when we use pipelining in conjunction with time-borrowing simultaneously - a feature that for all practical purposes can only exist in the IRI implementation, we are able to minimize the gap on the high end. In short, IRI register approach can deliver performance very close to that achieved by the “registers-everywhere approach” but at a fraction of the cost in area and power. Moreover, this approach also benefits traditional designs where design modification in the form of adding pipeline stages is not permitted.

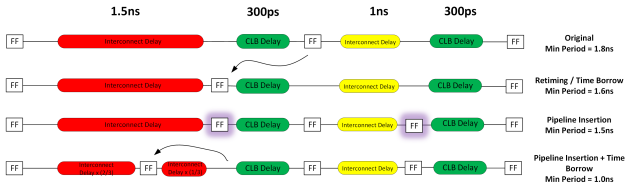
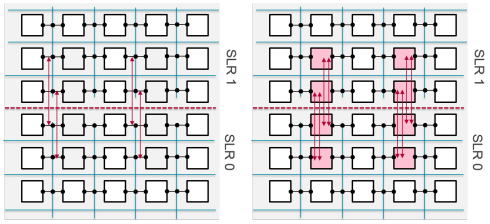


Figure 12: Cycle Time Reductions from Pipelining and Time Borrowng

4 SSIT

Versal uses a 4th generation stacked silicon interposer technology (SSIT) to construct ultra-large and heterogenous devices at reasonable costs. Multiple active silicon dice called Super Logic Regions (SLRs) are stacked on a passive interposer and connected together through microbumps and metal traces on the interposer. Prior works show that the total number of inter-SLR routing tracks (or SLLs) is about 25% of the routing tracks observed in an arbitrary horizontal cut within an SLR, therefore the place-and-route tools need to have awareness of the multi-SLR architecture and inter-die interfaces [19]. The design may be partitioned with a minimal number of connections between SLRs to reduce delays and routing congestion at the SLR boundaries.

4.1 SLL Interface Architecture



(a) Routing based **(b) Tile based**

Figure 13: SLL Channel Architecture

We classify the SLL interface architectures in previous SSIT based devices as "routing based" and "logic tile based". In 28nm based 7-series Xilinx FPGAs, the interfaces were "routing based." Connections to microbumps from the FPGA fabric were made directly on the routing channels. Tri-stated wires on each individual SLR were shorted on the interposer, which allowed the router to treat the SLLs just as another routing resource with drivers and

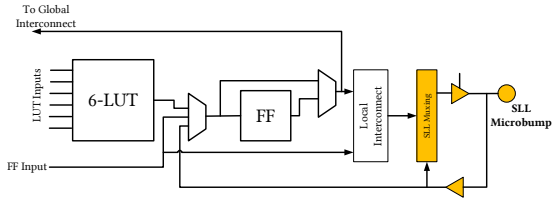
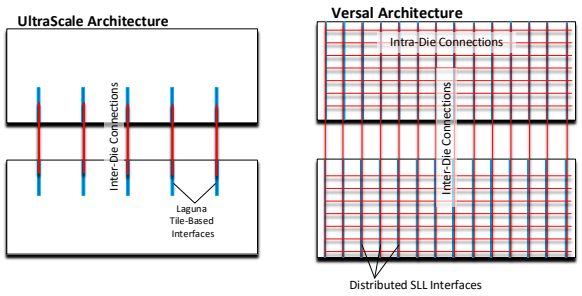


Figure 14: Access to/from FFs within the CLB from/to SLLs

loads in different SLRs. Since routing channels are ubiquitous on the FPGA fabric relative to logical tiles, this had the software effect of simply extending the existing routing infrastructure.

20/16nm based UltraScale FPGAs introduced the "Laguna" tiles, which were specialized logic tiles that displaced CLB tiles at each SLR boundary. Laguna tiles contain optional registers to create fast synchronous SLL connections across the SLR boundaries. This achieved inter-SLR frequencies of more than 500MHz in the -2 speedgrade [20]. However, to minimize total Laguna cost, each column of Laguna tiles appears relatively infrequently compared routing based approach and has much greater inter-SLR connections per channel. [19] showed that the concentration of inter-SLR connections at Laguna tile channels could result in routability hot spots.

The Versal SSIT architecture connecting SLRs includes a hybrid approach that maximizes the benefits of both routing based and logic tile based SLL interfaces. Instead of a standalone Laguna tile, we distribute and embed the SLL interface into each CLB. Costs are kept at a minimum by leveraging the CLB's interconnect and internal routing. We take advantage of the larger 4X CLB's internal routing structure as described in Section 3 to provide local access to each SLL. The internal routing structure also provides fast connectivity to and from registers within the CLB. As shown in Figure 14, the registers have optional bypass capability, which allows the SLL interface to operate synchronously or asynchronously. In addition, the same register type and control set granularity is used for intra-SLR and inter-SLR connections, giving more options on the placement of a given register. Similar to the routing based SLL interface, there are significantly more SLL channels and fewer tracks per channel.



(a) Laguna Tile Interface **(b) Distributed SLL Interface**

Figure 15: SLL Interface

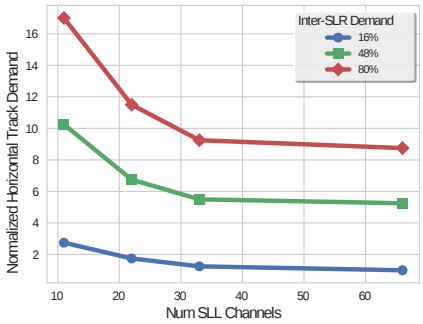


Figure 16: Horizontal Estimated Channel Demand. The sample UltraScale device contains 11 SLL channels while Versal contains 33.

Near the SLR edge, the pattern connects CLBs in one SLR to another. Towards the center of the SLR, the pattern creates a full mesh connecting CLBs within the same SLR. We discuss intra-SLR routing in the following section.

In Figure 16, we use the evaluation methodology described in [19] to implement multi-SLR synthetic designs with controlled inter-SLR connectivity. We compute the horizontal estimated channel demand as the number of SLL channels per device increases while keeping the total number of SLLs constant. We observe a decline in horizontal track demand as we increase the number of SLL channels. As the inter-SLR demand grows, the incremental benefit of adding more SLL channels is greater. When 80% of the SLLs are used, the horizontal congestion reduces by 50% in the Versal (hybrid) vs. UltraScale (tile-based) approach.

The reduction in horizontal congestion directly translates to better routability. We also observed a 40% reduction in horizontal wirelength and a 5% improvement in routability with the Versal hybrid SLL interface vs. UltraScale's tile-based Laguna interface.

4.2 Intra-SLR Routing

As shown in Figure 15, the Versal Architecture also takes advantage of the interposer to create a full mesh of long distance wires both between and within SLRs. This results in a more scalable routing architecture, since larger SSIT devices (which typically are targeted for designs of higher routing complexity) have an extra layer of routing that smaller devices do not have to pay for. This feature did not exist in any prior SSIT architectures. The long wires on the interposer are also about 30% faster than regular interconnect routes for similar logical distances. Long SLL wires on the interposer alleviate both horizontal and vertical routing congestion by freeing up local routing resources. Across our suite of designs, we observe an 8% reduction in horizontal wirelength and a 6% reduction in vertical wirelength within each SLR due to the interposer routing structure. The reduction is greater for designs that span a higher number of SLRs.

5 GLOBAL SUBSYSTEMS

5.1 Global Clocking

FPGA architectures have the unique problem of supporting many clock networks in spatially variable locations. Metal resistance is not scaling well, so as transistor delays continue to improve, clock delays associated with long wires do not. CPUs or ASICs often use thick metal layers to distribute clocks. FPGAs cannot afford this without sacrificing clocking capacity. Since clock load locations are not pre-defined, many clocks must run throughout the chip in parallel. In addition, FPGA architectures cannot afford to have per-device fully customized clocking solutions. The Versal architecture needed to be scaleable in order to support several device variants that span an order of magnitude in size. At the same time, clock frequencies continue to increase to keep up with external bandwidth demands. Clock skew as a percentage of cycle overhead thus increases too. We used a 3-prong approach to reduce clocking overhead: 1) adaptive clock deskew, 2) isolated clock supplies, and 3) local clock dividers.

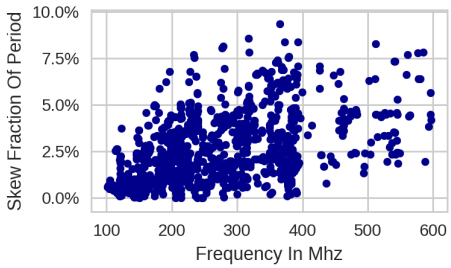
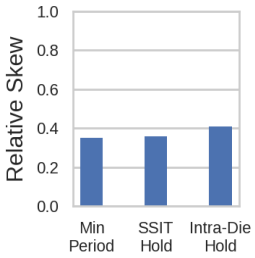


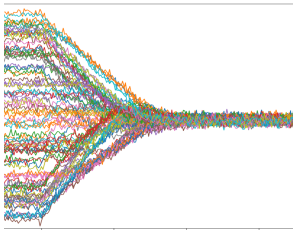
Figure 17: Skew as a fraction of clock period as a function of frequency

Even the most perfectly balanced clock tree incurs skew penalties primarily due to process mismatch along nominally matched paths. In order to reduce clock skew without sacrificing clocking capacity, we implemented an adaptive clock deskew scheme that actively modulates delays within the clock tree so that the process variation and even topological variations are tuned out. Traditionally, skew is computed based on a min/max spread of possible arrival times set by the range that the process can vary for a given speed grade, which becomes untenable for higher speed designs. Adaptive deskew tunes propagation delays so that process variation becomes a non-factor, increasing Fmax of all designs but especially the higher performance ones. Figure 17 shows a suite of UltraScale based customer designs that illustrate the trend of higher frequency designs incurring greater clock skew penalties.

Versal devices are broken into a grid of fabric regions, where each fabric region denotes clocking segmentation at its boundaries. The number of fabric regions varies based on device from less than 10 to over 100. The Versal adaptive deskew scheme has phase detectors at each fabric region boundary. These phase detectors send phase mismatch information back to a delay line and state machine within each fabric region. Based on the phase detector feedbacks, each fabric region auto-negotiates its own delay until the phase mismatch at all adjacent boundaries are minimized. Figure 18b shows one simulation result, a relative magnitude comparison where each line represents the delay to a given fabric region over time before and after adapative deskew is enabled. Each deskew state machine is initially programmed by software to nominally equalize delay from the clock source to each fabric region. Then, the adaptive deskew system is enabled to match actual silicon delays at the fabric region boundaries. To make the process transparent to the user, the entire deskew process occurs as a step during configuration, using a configuration clock, so that user clocks need not be active and continously running.



(a) Skew improvement due to adaptive deskew



(b) Clock injection delays per region as adaptive deskew is enabled

Figure 18: Impact Of Adaptive Deskew

Adaptive deskew directly minimizes region to region clock skew, but in turn also reduces global skew. The same scheme is also used across SSIT boundaries, so that the historically higher clock skew between SLRs is also critically reduced. We expect to enable registering of inter-die paths on both source and destination sides in a system synchronous environment without causing hold violations and needing special software support [20]. As shown in Figure 18a, global clock skew measured in terms of setup, hold, intra or inter die, are all expected to reduce by 60% or more.

In order to reduce clock jitter, the fabric region to region clock spines are implemented on a more isolated supply, where decoupling capacitors could be added more liberally. Although clock and data jitter are largely uncorrelated over long distances, we found at the local level that data and clock jitter are more correlated and thus kept the leaf level clocks on the local common supply.

Timing closure between related clocks generated from the same source can be challenging, since in most architectures the nearest common node from a timing perspective is at the PLL or DLL, which can be nanoseconds away from clock loads. Designs that operate different portions of the same logical design at frequencies which are divided from a master frequency is now very common. We added clock dividers to every clock leaf, which allows a single clock to be distributed through the clock network and then multiple frequencies generated locally at the leaf level. Inter clock skew reduces as a result by an order of magnitude assuming the related clock is a divided version (divide by 2,4, or 8) of the base clock. This not only improves clock skew for paths with related clock frequencies, but it also results in reduced global clock track demand. In previous architectures, every variant of the master clock had to be routed in parallel across the entire device. In Versal architecture,

we route only the single master clock and derive divided frequencies as demanded by the placement.

To reduce clocking power, we enable designs to use dual-edge clocking in a way that is transparent to the user. Each clock leaf can opportunistically use its clock divider to send the clock at half rate and then each fabric block multiplies the clock back to its original frequency. Dual edge clocking introduces a duty cycle distortion timing penalty overhead, so software would enable only clock leaves that have sufficient slack to use dual-edge. Since leaf clocks are further down the clock tree than clocks spines, there are many more of them, and thus they consume the bulk of global clock power (roughly 80%). Even though the clock spines send the clock at full rate and not all clock leaves use dual-edge, total clock power can still reduce substantially. For example, if 80% of clock leaves can take advantage of this feature, global clock power reduces by 40%.

5.2 Network on Chip

FPGAs have been very successful in providing users with a bit level configurable interconnect. This interconnect emulates the routing done during the design of an ASIC in the configurable fabric of an FPGA. The semantics that each routing resource presents to the user are very similar to that of a routing track in ASIC design. Such a model provides tremendous flexibility in the way users can map their designs. But this fine level of granularity suffers from significant loss of efficiency [21]. Secondly, increasingly larger portions of the device resources are now being spent in managing this communication. As was described in Section 1 this problem is worsened by interconnect technology trends as well.

To address the issue of bit level management it makes sense to organize data movement into wide standardized bussed interfaces. ASICs and SoCs faced a similar problem of moving many high bandwidth datastreams. These were initially addressed by busses. A next step was to increase bus pipelining. This was followed by a move to point-to-point interconnects with many independent parallel paths. An addition to these was to fully packetize the data and control information to more efficiently use the wires and buffers in the network [22]. These networks are now commonly referred to as NoC. In packet switched NoCs, the same physical resource is used to route communication between multiple ports, thus increasing area efficiency.

For FPGAs, researchers have similarly proposed various techniques to improve on the efficiency of bit level interconnect. These include requiring users to reason at the word level rather than at bit level [23], to implementing NoCs as hardened interconnect resources on the FPGA [24–26]. In the Versal architecture, we implement a hardened NoC as a separate level of interconnect augmenting the traditional FPGA interconnect. The traditional FPGA interconnect continues to provide bit level flexibility, but as more and more of the system level communication occurs in a structured fashion, the NoC is able to absorb much more of the interconnect demand. This separates system level communication implementation from compute implementation. In traditional FPGA implementations, since both communication and compute were implemented on the programmable fabric, designs with demanding interconnect resulted in reduced compute and vice versa. Moreover, it is now not important to co-locate where compute occurs with where communication needs of the compute are satisfied. Consider the concrete case of a compute IP requiring access to some memory controller. In order to close timing at high frequencies (required to support high bandwidths), the compute would have to be placed close to the memory controller. Alternately, the physical implementation tools would have to be smart enough to insert on-demand pipelining. On the other hand, with NoC, it is possible for the compute to be implemented anywhere on the FPGA. All it needs to do is hook up to the nearest NoC port for communication to occur at a guaranteed bandwidth.

Figure 19 shows the topology of the NoC in relation to rest of the device resources. The NoC topology is a compromise between increased routing flexibility and minimal perturbation to the rest of the fabric. Most academic NoC topologies focus on implementing a mesh. There are several advantages of a mesh topology in terms of routing flexibility. However, a pure mesh topology is expensive and would be underutilized. In Versal, NoCs appear in the main fabric as columns. In a columnar architecture, the NoC columns integrate with the rest of the fabric just like any other block - DSP, BRAM etc would. Each column is sized sufficiently to be able to saturate the bandwidth of a DDR4/LPDDR4 memory controller operating at the maximum supported speeds. At the bottom and top of the device there are a greater number of NoC channels that interface with the processor and the memory controllers. The reason for higher bandwidth topologies at the top and bottom edges is to provide some flexibility to resources that demand memory bandwidth from more than one controller. Topologies with higher cross sectional bandwidths at the top and bottom make compute kernel placement an easier problem than it otherwise would have been.

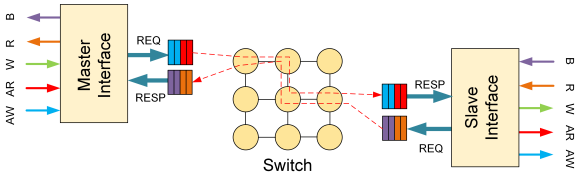


Figure 19: NoC Conceptual Diagram

Figure 19 shows the blocks that make up a NoC topology. The primary block is a four ported switch which routes traffic from any of the four ports to any of the other ports. Peripherals such as masters, slaves or memory controllers form the endpoints in this topology. Masters perform either a read or a write request addressing any of the slaves or memory controller which in turn fulfill these requests. Fabric ports can serve both as masters and slaves. The physical transport is managed by the routing tables within each switch which tell it how to route each packet. It is the task of the NoC software layer to ensure that bandwidth demands are met without deadlock. The NoC supports standard AXI4 memory mapped and streaming semantics and different classes of Quality of Service. It also supports multiple virtual channels to alleviate head of line blocking of traffic and to aid deadlock free routing.

Within the device, the NoC provides ports that permit communication from the processor to all the peripherals. There are also NoC ports at regular intervals in the fabric for soft implementations of

some compute to communicate either with each other or with the peripherals. The NoC integrates the entire memory address space of the device. Any master attached anywhere in the device, whether via the fabric or the processor subsystem can address any other slave in the device whether it is a port on the NoC or whether it is a memory controller. This level of integration is extended to SSI devices as well. All masters can address any slave in any die using a uniform address space.

5.3 Configuration

The configuration system in Versal architecture has increased both in terms of capabilities as well as configuration speed. By pipelining the configuration infrastructure and increasing the config bus width, we were able to achieve an 8X speedup in terms of configuration time per bit. Any design that reconfigures frequently will have less configuration overhead and more time for computation. Readback speedups are even more pronounced at 56-300X relative to UltraScale. This is due to a combination of configuration infrastructure speedup, concentrating flop state readback memory into fewer address frames, read pipeline efficiency gains, and enabling parallel readback of multiple dice in a device. For designs with lower Fmax (50Mhz or less), it is possible to take snapshots of the design state without stopping the clock.

The platform management controller (PMC) is a dedicated processor that handles device management control functions such as power sequencing, initialization, boot, configuration, security, power management, and health monitoring. It consists of a hardened microblaze core, boot ROM, peripheral interfaces, security accelerators, and power management units. Similar to UltraScale devices, fabric blocks are configured via a distributed array of memory cells controlled by a grid of address and data lines. Relying on the same system to program peripheral blocks however is problematic, primarily because these blocks have no fixed physical relationship with the interior fabric. An additional configuration mecahnism was added to service these blocks that runs in tandem with the NoC, known as the NoC Peripheral Interface (NPI). NPI is a streamlined packet based switch network that routes in a tree structure to the peripheral blocks, and is seamlessly integrated into the configuration sequencing. The PMC can selectively write to and poll status registers on specific endpoints.

From a partial reconfiguration standpoint, reconfigurable region granularity has become finer. Most reconfiguration regions are now the size of single blocks, and blocks that share common physical space are seperated logically to minimize loss in functionality when requiring specific regions to reconfigure.

6 CONCLUSION

Xilinx addresses current semiconductor technological, economical, and scalability challenges with the new 7nm ACAP heterogeneous compute platform. The Versal™ architecture tightly integrates programmable fabric, CPUs, and software-programmable acceleration engines into a single device that enables higher levels of software abstraction, enabling more rapid development of hardware accelerators that solve next generation problems.

REFERENCES

[1] E. Track, N. Forbes, and G. Strawn, "The End of Moore's Law," *Computing in Science Engineering*, vol. 19, no. 2, pp. 4–6, Mar 2017.

[2] D. Patterson. The Past is Prologue: A New Golden Age For Computer Architecture. [Online]. Available: https://cra.org/wp-content/uploads/2018/07/2018_CRA_Snowbird_Keynote_Patterson.pdf

[3] J. Hennessy. (2018, 03) The end of road for general purpose processors & the future of computing. [Accessed: 2018-09-12]. [Online]. Available: https://web.stanford.edu/~hennessy/Future%20of%20Computing.pdf

[4] Y. Li, X. Zhao, and T. Cheng, "Heterogeneous computing platform based on cpu+fpga and working modes," in *2016 12th International Conference on Computational Intelligence and Security (CIS)*, Dec 2016, pp. 669–672.

[5] Xilinx, "Xilinx FPGAs to be deployed in new Amazon EC2 F1 Instances," *Xilinx Press Releases*, 2016.

[6] ——, "Baidu deploys Xilinx FPGAs in new public cloud acceleration services," *Xilinx Press Releases*, 2017.

[7] ——, "Xilinx selected by Alibaba cloud for next-gen FPGA cloud acceleration," *Xilinx Press Releases*, 2017.

[8] A. M. Caulfield, E. S. Chung, A. Putnam *et al.*, "A cloud-scale acceleration architecture," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2016, pp. 1–13.

[9] L.-C. Lu, "Physical Design Challenges and Innovations to Meet Power, Speed, and Area Scaling Trend," in *Proceedings of the 2017 ACM on ISPD*. New York, NY, USA: ACM, 2017, pp. 63–63.

[10] SDAccel development environment. [Online]. Available: https://www.xilinx.com/products/design-tools/software-zone/sdaccel.html

[11] *UltraRAM: Breakthrough Embedded Memory Integration on Ultrascale+ Devices*, Xilinx.

[12] G. Singh and S. Ahmad, "Xilinx 16nm datacenter device family with in-package HBM and CCIX interconnect," 2017, HotChips,.

[13] B. Farley, J. McGrath, and C. Erdmann, "An all-programmable 16-nm RFSoC for Digital-RF communications," *IEEE Micro*, vol. 38, no. 2, pp. 61–71, Mar 2018.

[14] R. Nimaiyar *et al.*, "Xilinx DNN Processor An Inference Engine, Network Compiler + Runtime for Xilinx FPGAs," 2018, HotChips.

[15] A. K. Jain, D. L. Maskell *et al.*, "Throughput oriented FPGA overlays using DSP blocks," in *2016 DATE Conference Exhibition*, March 2016, pp. 1628–1633.

[16] "GRVI Phalanx on Xilinx Virtex Ultrascale+: A 1,680-core, 26 mb risc-v parallel processor overlay," in *3rd International Workshop on Overlay Architectures For FPGAs*, 2016.

[17] D. Lewis, G. Chiu, J. Chromczak *et al.*, "The Stratix™10 highly pipelined FPGA architecture," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '16. New York, NY, USA: ACM, 2016, pp. 159–168. [Online]. Available: http://doi.acm.org/10.1145/2847263.2847267

[18] I. Ganusov and B. Devlin, "Time-borrowing platform in the Xilinx Ultrascale+ family of FPGAs and MPSoCs," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, Aug 2016, pp. 1–9.

[19] C. Ravishankar, D. Gaitonde, and T. Bauer, "Placement strategies for 2.5D FPGA fabric architectures," in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, Sept 2018.

[20] C. Ravishankar, H. Fraisse, and D. Gaitonde, "SAT based Place-And-Route for High-Speed Designs on 2.5D FPGAs," in *2018 International Conference on Field-Programmable Technology*, Dec 2018.

[21] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," in *Proceedings of the 2006 ACM/SIGDA 14th International Symposium on FPGAs*, ser. FPGA '06. New York, NY, USA: ACM, 2006, pp. 21–30.

[22] W. J. Dally and B. Towles, "Route packets, not wires: On-chip inteconnection networks," in *Proceedings of the 38th Annual DAC*, ser. DAC '01. New York, NY, USA: ACM, 2001, pp. 684–689.

[23] A. Ye and J. Rose, "Using Bus-based Connections to Improve Field-Programmable Gate-Array Density for Implementing Datapath Circuits," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 14, no. 5, pp. 462–473, May 2006.

[24] R. Gindin, I. Cidon, and I. Keidar, "NoC-based FPGA: Architecture and routing," in *First International Symposium on Networks-on-Chip (NOCS'07)*, May 2007, pp. 253–264.

[25] G. Schelle and D. Grunwald, "Exploring FPGA network on chip implementations across various application and network loads," in *2008 International Conference on Field Programmable Logic and Applications*, Sept 2008, pp. 41–46.

[26] M. S. Abdelfattah and V. Betz, "Design tradeoffs for hard and soft FPGA-based networks-on-chip," in *2012 International Conference on Field-Programmable Technology*, Dec 2012, pp. 95–103.