Architectural Enhancements in Intel® Agilex™ FPGAs

Jeff Chromczak
jeff.chromczak@intel.com
Intel Corporation
Toronto, Canada

Mark Wheeler
mark.wheeler@intel.com
Intel Corporation
Toronto, Canada

Charles Chiasson
charles.chiasson@intel.com
Intel Corporation
Seattle, USA

Dana How
dana.how@intel.com
Intel Corporation
San Jose, USA

Martin Langhammer
martin.langhammer@intel.com
Intel Corporation
United Kingdom

Tim Vanderhoek
tim.vanderhoek@intel.com
Intel Corporation
Toronto, Canada

Grace Zgheib
grace.zgheib@intel.com
Intel Corporation
San Jose, USA

Ilya Ganusov
ilya.ganusov@intel.com
Intel Corporation
San Jose, USA

ABSTRACT

This paper describes architectural enhancements in Intel® Agilex™ FPGAs and SoCs. Agilex devices are built on Intel's 10nm process and feature next-generation programmable fabric, tightly coupled with a quad-core ARM processor subsystem, a secure device manager, IO and memory interfaces, and multiple companion transceiver tile choices. The Agilex fabric features multiple logic block enhancements that significantly improve propagation delays and integrate more effectively with the second-generation HyperFlex™ pipelined routing architecture. Routing connections are re-designed to be point-to-point, dropping intermediate connections featured in prior FPGA generations and replacing them with a wider variety of shorter wire types. Fine-grain programmable clock skew and time-borrowing were introduced throughout the fabric to augment the slack-balancing capabilities of HyperFlex registers. DSP capabilities are also extended to natively support new INT9/BFLOAT16/FP16 formats. Together, along with process and circuit enhancements, these changes support more than 40% performance improvement over the Stratix® 10 family of FPGAs.

ACM Reference Format:
Jeff Chromczak, Mark Wheeler, Charles Chiasson, Dana How, Martin Langhammer, Tim Vanderhoek, Grace Zgheib, and Ilya Ganusov. 2020. Architectural Enhancements in Intel® Agilex™ FPGAs. In *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '20), February 23–25, 2020, Seaside, CA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3373087.3375308

FPGA '20, February 23–25, 2020, Seaside, CA, USA

ACM ISBN 978-1-4503-7099-8/20/02…$15.00
https://doi.org/10.1145/3373087.3375308

1 INTRODUCTION

This paper describes key architectural enhancements in the programmable fabric of Intel® Agilex™ FPGAs and SoCs. Agilex FPGAs are built on Intel's 10-nanomenter chip manufacturing technology, which features 3rd generation FinFET transistor technology and is Intel's first high-volume manufacturing process to employ self-aligned Quad Patterning (SAQP), contact-over-active-gate (COAG) technology, as well as the use of cobalt in local interconnect, vias, and interconnect liners [2]. Compared to the previous Stratix® 10 generation of Intel's FPGAs, Agilex dramatically increases the performance and power efficiency of the programmable fabric, while retaining a quad-core ARM processor subsystem, expanding transceiver choices with lane speeds up to 112Gbps, adding hardened support for coherent attach to Xeon processors with UPI and CXL, and upgrading to PCIe Gen4/5 interfaces. Agilex FPGAs target a variety of applications spanning acceleration and compute in data centers, wireless 5G, Network Function Virtualization (NFV), automotive, as well as industrial and military/aerospace.

Stratix 10 was the first Intel product to leverage embedded interconnect bridge (EMIB) technology to disaggregate transceiver silicon from the rest of the FPGA SoC. Agilex continues to use EMIB to partition SoCs into two die types – (1) transceiver tiles

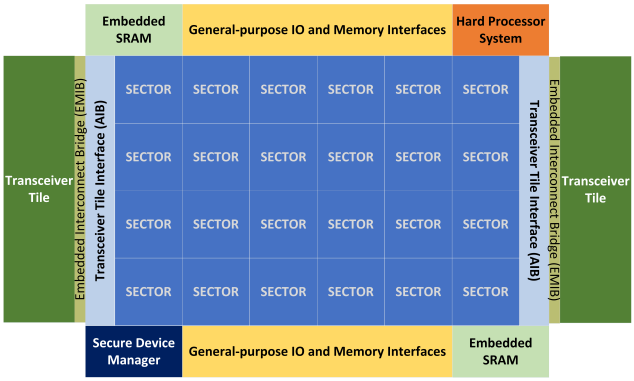


Figure 1: Representative device floorplan

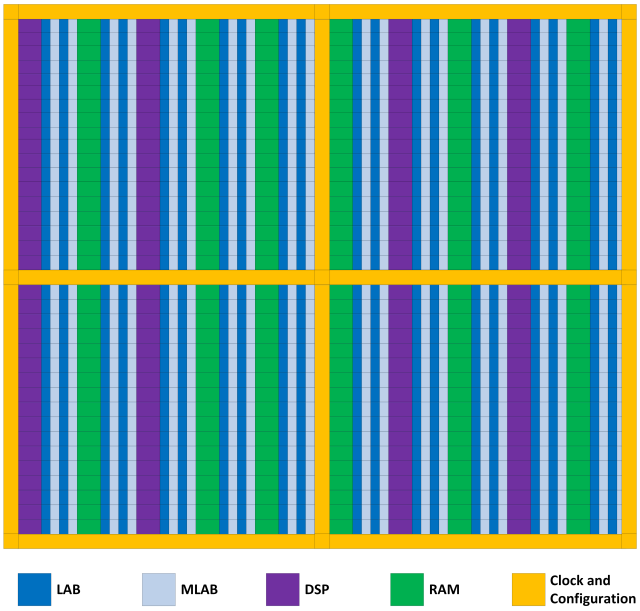


Figure 2: Representative fabric sector

comprising hard IP and (2) fabric die comprising programmable fabric, IO interfaces, processor subsystems, a secure device manager (SDM), and on-chip memories. This architecture enables flexible integration of heterogeneous technologies and differentiated connectivity solutions. Figure 1 shows a representative die floorplan from the Agilex family.

The programmable fabric die features transceiver interfaces on its left and right sides. The top of the fabric die contains off-chip memory and IO interfaces, as well as the hard processor system (HPS). The SDM and additional IO/memory interfaces are located on the bottom. The programmable fabric in the middle of the die is composed of a number of sectors with independent clocking and configuration. Agilex moves IO interfaces (and processors) from inside the fabric to outside its periphery at the top and bottom of the die, which represents a significant change from the previous two generations which had these embedded in the fabric. Column IO can simplify floorplan construction but it is very disruptive to the fabric, since IO columns block routing wires and do not scale as well as other components so fabric wire crossings incur a significant and growing delay penalty. Periphery IO eliminates these blockages for improved fabric performance and a better user experience. Moving IO to the periphery also improves signal integrity of IO routes and power integrity in the core.

Figure 2 shows the composition of a typical fabric sector. With the move of IO and processors to outside the fabric, fewer sector variants are required, and fabric regularity and translational consistency are improved. This leads to more predictable timing closure and IP portability.

Although the Agilex architecture is derived from the Stratix 10 FPGA family, it incorporates multiple architectural changes to reduce propagation delays and enable broad-based performance improvements. The configurable logic features multiple changes in

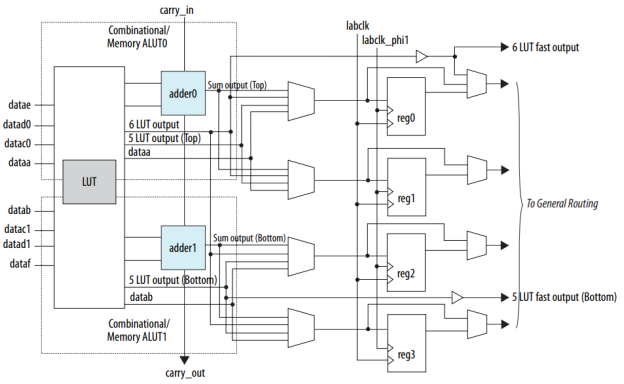


Figure 3: The ALM architecture

input and output connectivity which speed up combinational delays by 50% on average, as described in Section 2. The second-generation HyperFlex™ routing architecture has been optimized to reduce delay and area overheads of interconnect registers. The HyperFlex registers change from pulse latches to full edge-triggered flip-flops flops to reduce sensitivity to process variation and enable a wider voltage operating range. The general purpose routing has changed from multi-drop to single-drop, introducing new shorter wire types without HyperFlex registers while retaining them on the original longer wires. The Agilex clocking architecture has also been extended with a systematic approach to beneficial clock skew scheduling and time-borrowing, as detailed in Section 4. To support the needs of emerging compute-intensive applications, the Agilex DSP introduces native support for several new lower-precision numeric formats, as described in Section 5. Section 6 demonstrates that the Agilex programmable fabric is expected to deliver over a 40% performance improvement, on average, over a wide variety of applications.

2 LOGIC BLOCK

Agilex™ maintains the same general structure of the *Logic Array Block (LAB)* as in Stratix® 10. Each LAB consists of ten *Adaptive Logic Modules (ALMs)* with carry and clock logic. The ALM, depicted in Figure 3, can operate in three modes: (1) single 6-input LUT mode, (2) fracturable LUT mode with two 5-input LUTs, and (3) arithmetic mode. Agilex enhances this logic architecture with targeted architectural improvements that bring substantial speedups across the different modes.

Increased emphasis was placed on speeding up inputs to better support a wider range of designs. Remember that the inputs to a LUT drive the select signals in a tree of muxes as shown in Figure 4(a); thus pin *A* has the greatest delay to the output and *F* has the smallest delay. This Figure also illustrates the (b) dissimilar unbalanced path depths found in traditional designs, as well as the (c) similar nearly balanced path depths found in very high speed, highly pipelined designs. The latter designs increase the number of (near) critical input pins and thus the fastest LUT inputs best for them are often exhausted; speeding up the remaining slower input pins avoids performance degradation in this case. The former

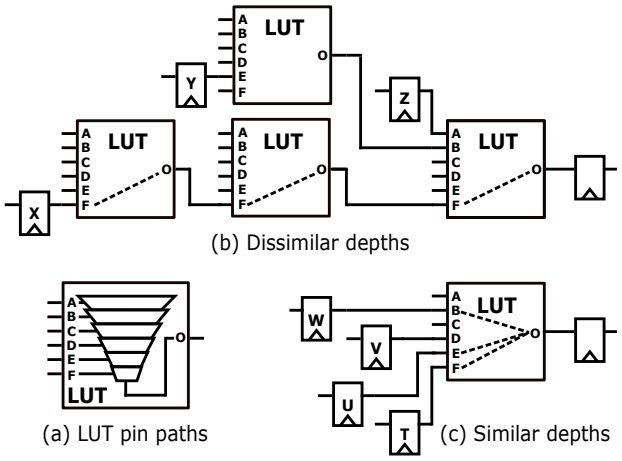


Figure 4: Design styles and (near) critical pin density

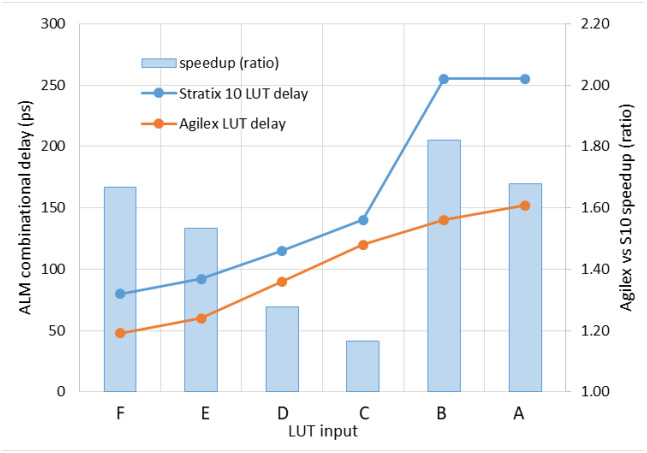


Figure 5: Agilex LUT delay improvement

designs are still very important and the critical path example shown benefits from lower *E*/*F* delay and direct LUT outputs, as well as the redesigned input network discussed in Section 3.

To further speed up the slow LUT inputs, the ALM register feedback path is removed. This improves the overall area and the combinational delay of inputs *A* and *B*. Removing this feedback path has a low negative impact, especially with HyperFlex, as registers tend to migrate from the ALM to the Hyper registers.

Furthermore, fast output pins are added, connecting the 6-LUT and one of the two 5-LUT outputs directly to the routing. These direct paths bypass the ALM registers and the output multiplexers. Their routing connectivity is optimized to speed up critical LUT chains and LUTs driving HyperFlex™ registers in routing.

These optimizations, combined, show substantial performance improvement across a wide variety of designs and use cases. Figure 5 shows the input-to-output 6-LUT delays for Agilex and Stratix® 10, measured from each input to the fastest output. The geomean speedup across all LUT inputs is 1.5×, with the largest improvement observed on the slowest and fastest inputs.

Agilex retains all four independent registers in the ALM, which helps improve the packing density in register-rich highly-pipelined designs. However, these ALM registers now share more extensively the control signals *CLK*, *CE*, *ACLR*, *SCLR*, and their associated HyperFlex registers. This reduces the area without significant loss to the performance or packing density. Agilex also supports more flexible LAB-level clock selection by providing two independent clocks per LAB. This added flexibility can be used to increase packing density and to boost the performance through beneficial clock skew, as discussed in Section 4.

As for the arithmetic mode, the ALM-level arithmetic-supporting logic is largely unchanged. However, LAB-level enhancements were introduced to significantly improve the mapping of very wide arithmetic functions typically used in cryptography applications. Traditional FPGA pipelined carry adder architectures are prohibitive in both area and latency at word widths used in cryptography (e.g., 2048 or even 4096 bits). However, this overhead can be addressed by creating both the generate (G) and propagate (P) values of several segments of the input words, and using a look ahead structure on these (G, P) bits to calculate a set of input carries for the output segments [10]. This approach was already largely supported by the carry skip adder structure inside the LAB [14]. Thus, minimal logic circuity was added to Agilex to output the (G, P) pair from the last ALM in each LAB, as segments are usually integer multiples of LAB lengths. These enhancements reduce the ALM count of wide pipelined lookahead adders by as much as 33%.

Logic improvements in Agilex targeted not only the architecture within the LAB and ALM, but also the density and distribution of the LABs and MLABs within the fabric. MLABs are LABs in which the ALMs have an additional LUTRAM mode. In Agilex, the MLAB density is doubled, when compared to Stratix 10, such that 50% of all LABs now support LUTRAM functionality. This improves the total LUTRAM bandwidth and the locality for high-speed memory-intensive IPs. Dedicated inputs for the LUTRAM write address are also added to reduce loading and eliminate extra gating logic on critical LUT control signals. The redesigned input network, discussed in Section 3, uses narrower multiplexers, so the area cost of adding these new input pins is roughly half that of previous architectures.

3 ROUTING

In FPGA routing, the conventional performance, power, and area (PPA) objectives are addressed by optimizing the routing network topology, the routing mux circuit designs, and the metal stack usage. In the routing network topology, we can simultaneously address all three PPA objectives by defining topologies which minimize the mux counts along critical paths, as well as the unused input pin counts on all paths. In this we repeatedly trade-off performance and routing flexibility. Optimizations in the mux circuit designs, the metal stack usage, and the wire planning provide further improvements.

3.1 Definitions

Agilex™ FPGAs and their predecessors contain a grid of fabric functions such as LABs, memories (M20Ks), DSPs, etc., as shown

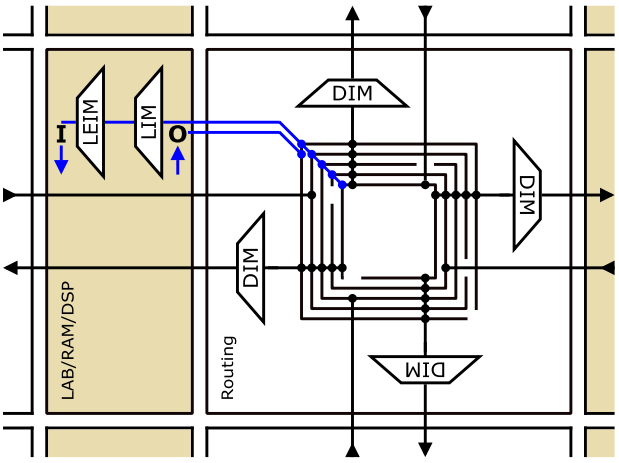


Figure 6: Routing block (without fly-overs) in fabric grid

in Figure 6. Each grid point contains one fabric function and one or more routing blocks. Each fabric function connects only with its associated routing blocks, and the routing blocks additionally connect with each other using Manhattan wires. Inside the routing blocks discussed in this section, each exiting wire driving another routing block is driven by a *Driver Input Mux (DIM)*, which selects between wires entering from other routing blocks or the outputs of the associated fabric function. The exiting wires are grouped into "bundles", which each consist of wires with the same direction and length; for example, a bundle comprising e.g. H4R0 through H4R9 is 10 horizontal wires traveling right 4 routing blocks. An additional two-stage set of muxes selects from the same sources as the DIMs and drives the input pins on the associated fabric function. Inside this second set, the first stage has *Logic Input Muxes (LIMs)* and the second stage has *Logic Element Input Muxes (LEIMs)* [3, 12, 13, 15].

3.2 Capacitance Reduction

Reducing mux input pins is a key objective. Such reductions decrease capacitance and increase performance, but excessive reductions damage routing flexibility. Agilex routing includes three related improvements.

First, fabric functions (LABs, M20Ks, etc.) now drive only one destination routing block, instead of a block to the left and another to the right. Besides reducing unused input pins, this simplified the integration of routing blocks with their fabric functions since these connections are now one-to-one. Furthermore, each wire between routing blocks now drives only the one farthest destination routing block at its end, and no longer drives the closer interior non-terminal routing blocks. This avoids unused inputs on unrelated crossed routing blocks. Both these reductions in unused input pins enabled the input pin counts of DIMs, LIMs, and LEIMs to be halved. This delivered faster muxes using less power and area, and was a prerequisite to using CMOS muxes as discussed later.

Second, the removals of sinks just described required the compensating introduction of a greater variety of wire lengths. Among those, unit length horizontal wires were needed *also* to compensate

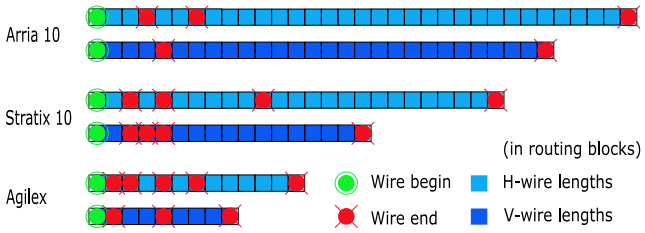


Figure 7: Wire lengths in blocks across generations

for fabric functions now driving only one routing block. These more numerous length options, and thus greater wire counts, needed additional metal layers. This restructuring also reduced power since only the needed metal length and routing block were driven and no more. Finally, the overall shift to shorter wires follows recent trends as shown in Figure 7, which is the result of multiple effects.

Third, adjacent muxes aggressively share input pins. This does not reduce input pin counts, but it does reduce via stacks and their capacitance. Sharing is a requirement satisfied either in the initial bundle network design or later in optimizations determining detailed connectivity.

3.3 Connectivity in and across Planes and Lanes

In the Stratix® 10 clock architecture [4], each user clock is usually distributed within a subset of one "plane" of the 32-plane clock network. Interconnections between planes are rare. This plane-oriented approach was an inspiration for the Agilex signal routing network. Consequently, the routing network is now consistently structured in terms of identical planes, with one plane per ALM in the LAB, and each bundle having one or two wires in each plane; previously bundle sizes were not multiples of the ALM count. This structure, with planes, ALMs, and bundles all integrally related, yields significant implementation simplifications, reduces the complexity of the design space we must consider, and makes visible wiring delays more regular and predictable.

Furthermore, in Stratix 10, when a function block's output was routed to other function blocks' LIMs, for each such connection only a few LIMs were reachable through the inter-block routing. This meant each and every function block needed a highly connected and thus expensive LIM/LEIM input subnetwork, which would provide "lumped" connectivity flexibility to get the incoming signal to the desired input LEIM. Agilex moves away from Stratix 10's *lumped* input connection network, to inter-plane connectivity *distributed* across multiple locations in three ways.

First, the LIM/LEIM network is now broken into 4 "lanes" which each service 2.5 ALMs, newly omitting much connectivity between the lanes. In previous architectures, all LIMs drove LEIMs on all 10 ALMs, but in Agilex, most LIMs drive LEIMs on ALMs in only one lane, allowing shorter wires for the remaining connectivity. Shorter LIM to LEIM wires can also move to a better place in the metal stack. Figure 8 shows the lanes and the horizontal planes; the vertical planes launch and land in the same mux subgroups as the horizontal planes. Again, the redesigned input network balances flexibility and speed. Fanout and span are limited to improve delay between nearby ALMs, while still retaining sufficiently flexible access to

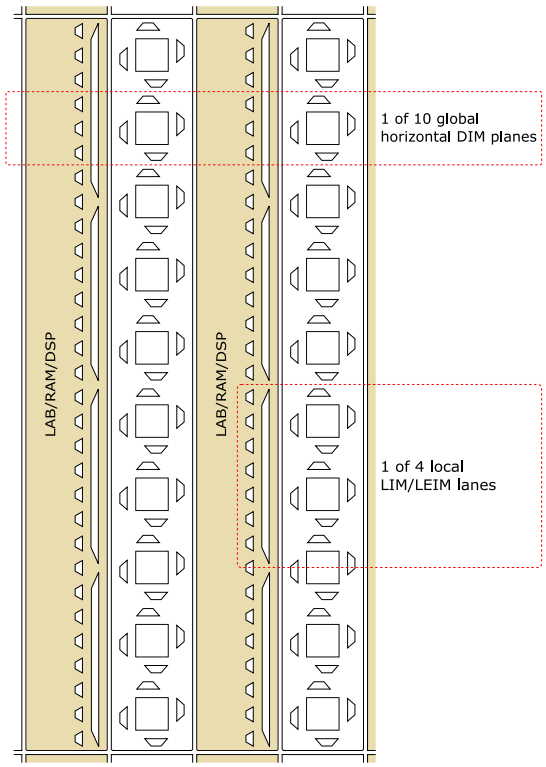


Figure 8: Planes and Lanes in one row of route blocks

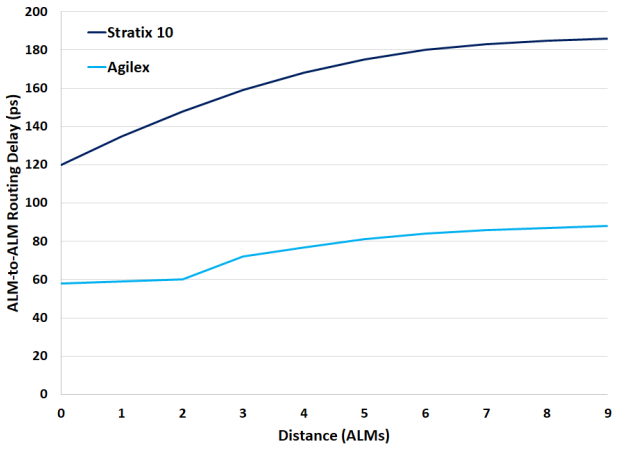
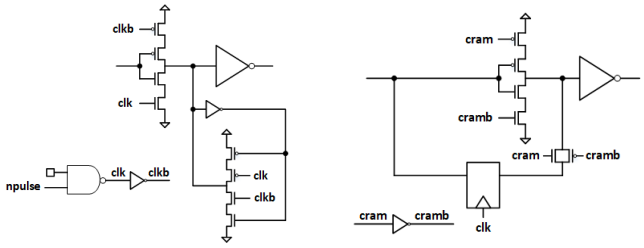


Figure 9: ALM-to-ALM routing delay improvement

multiple inputs of multiple ALMs. Even the few remaining longer nets with greater reach still offer significant delay reduction. The resulting improvement in routing delays to connect between ALMs in the same LAB is shown in Figure 9.

Second, additional DIM-to-LIM mux connections distribute the inter-lane connectivity previously in the LIM/LEIM network one mux stage backwards.



(a) Stratix 10 **(b) Agilex**

Figure 10: HyperFlex register and driver improvements

Third, additional DIM-to-DIM mux connections further distribute the inter-plane connectivity even farther backwards towards the signal driver. Not only can multiple plane jumps effect a lane jump, but this provides additional general routing flexibility.

The last two categories of additional mux connections (not shown in Figure 8) not only compensate for the reduced routability of the narrower muxes but were indeed one of the changes making narrower muxes possible. Increased routing choice decreases performance, but distributing the additional choices across upstream muxes reduces this cost while retaining its flexibility.

3.4 Circuit Improvements

Beyond these broad changes, Agilex routing also features some coordinated circuit design improvements. A general change was switching all muxes from over-driven NMOS to CMOS with an elevated V_{dd}. The latter works with more flexible voltage scaling and VID, which support a wider power-delay trade-off range, all leading to net power savings.

Stratix 10 introduced HyperFlex™ registers [15], which were provided in the second stage of the LIM/LEIM networks, and all the DIMs. Agilex alters these in multiple ways as shown in Figure 10.

- Changing registers from pulse latches to edge-triggered flip-flops improved hold time and reduced variation sensitivity.
- A restructured driver uses static configuration signals with boosted voltage for path select (**cram/b** instead of **clk/b**) to improve performance and reduce clock load.
- As we added shorter wires, we found they did not need registers. In this more diverse wire set, only 1/3 of DIMs have registers, for an area and power advantage.

3.5 Physical Design

Agilex is implemented using a custom variant of the Intel® 10nm process optimized for FPGAs. Extra metal layers are added and the pitch and thickness of each layer is tuned for improved fabric performance and routability. Reduced wire fanout decreases the number of tracks consumed by mux input routing and lets us allocate available pitch more effectively. Together these changes allow us to route critical wires with up to 50% larger pitch while maintaining or increasing track count compared to the previous generation.

Increased emphasis was placed on delay uniformity in architecture and implementation. Prior FPGAs have often included deliberate architectural resource differentiation [14, 15]; ideally fitting

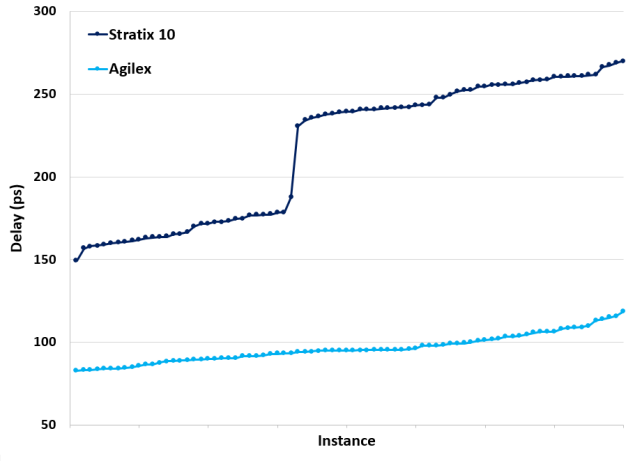


Figure 11: Vertical wire delay distribution

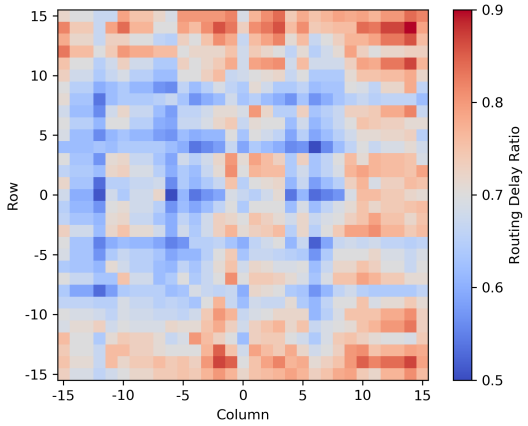


Figure 12: Routing delay reduction heat-map vs Stratix 10

would avoid slow instances on critical routes and best use the less common fast instances, but this is not always possible in congested regions or in high speed designs with many near critical paths, and it becomes less probable as connectivity becomes sparser for speed and cost. Agilex uses consistent resource assignments, and new specialized route implementation automation delivered further delay uniformity. Figure 11 shows the distribution of mid-length vertical wire delays in a single routing channel. Stratix 10 wire delays vary significantly, with an apparent jump as some wires are routed on thinner metal layers using a tighter pitch. Agilex wire delay is faster in all cases and much less variable.

3.6 Results

Together these Agilex architecture and circuit improvements deliver a 30% reduction in average routing delay.

Figure 12 shows a heatmap of routing delay reduction as a function of distance. Delay reduction is non-uniform due to changes in wire length and fanout locations, as well as the removal of prior delay non-uniformity. The largest improvement is seen on short connections which are most common on critical paths. The longest routing wires in Agilex are shorter than in Stratix 10 so a single long route will have more mux and buffer delay. In real designs we find that the Agilex long wires provide better system-level performance due to improved accessibility through more frequent entry and switching points.

Furthermore, fitting the GNL designs [16] confirms these improvements have also delivered greater routability. The new routing network can implement most individual designs at a higher Rent-like factor, by 0.05 to 0.10, and no other designs needed a lower factor.

4 CLOCKING AND SLACK BALANCING

On top of the logic and routing enhancements, the Agilex™ architecture offers additional performance improvements through modifications to the clocking architecture and through slack balancing techniques.

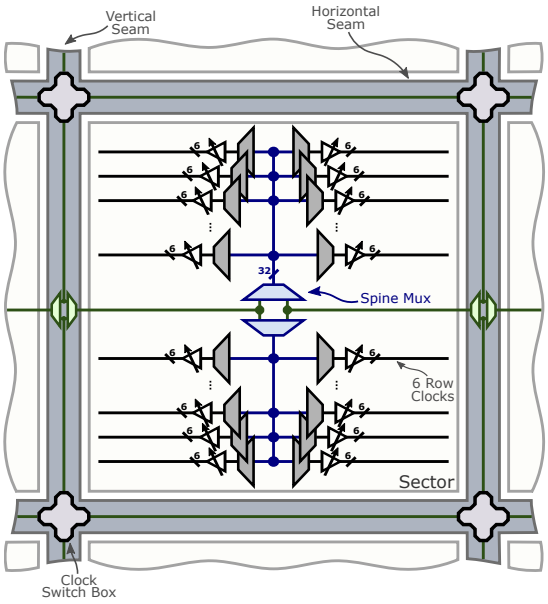


Figure 13: Sector-level clocking

4.1 Clocking Architecture

In Agilex™, the fabric is divided into sectors spanning multiple rows and columns of LABs, M20Ks, and DSPs. The global clocks are delivered to the center of each sector through horizontal and vertical clock seams. From the center, multiplexers drive the clocks up and down the spine of the sector, as shown in Figure 13. Each sector row then selects six of the clocks to drive the right side of the spine and another six to drive the left side of the spine. Any logic, memory, or DSP block that sits in that row and within the half-sector range gets to pick its routing and logic clocks from these six row clocks. Each row clock is equipped with a configurable delay chain. It is a set of delays that can be programmably added to

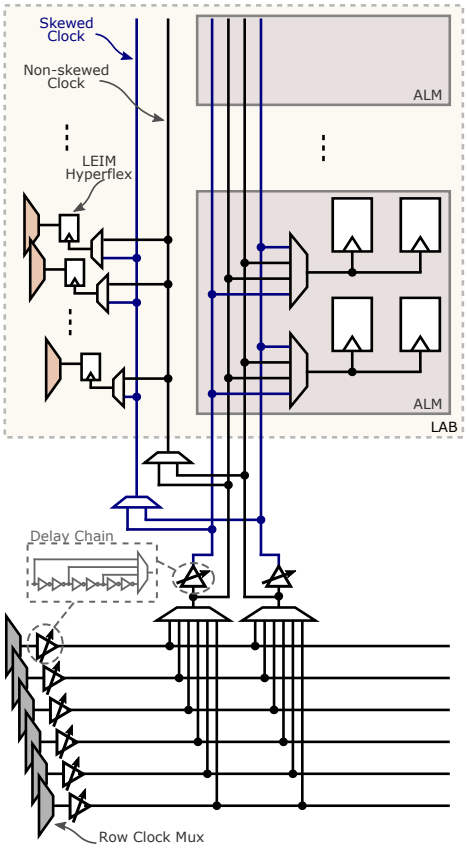


Figure 14: Agilex's LAB level clocking

the clock path to inject intentional skews. In Agilex, the row clock delay chain can be configured into one of three delays: 100*ps*, 200*ps*, and 500*ps*.

Similar to Stratix® 10, the LAB is allocated two clocks that are shared between the ALM flip-flops and the LEIM HyperFlex™ registers. Each of the two clocks has its own delay chain with configurable delays that are independent from the row clock delay chain, as shown in Figure 14. The LAB clocks can each pick one of four possible delays in their delay chains: 50*ps*, 100*ps*, 150*ps*, and 250*ps*. As mentioned in Section 3.4, only a subset of the DIMs have HyperFlex registers in Agilex. The hyper registers on the DIMs driving medium wires share one clock, while the ones on the DIMs driving long wires share a second clock. These two DIM clocks do not have dedicated delay chains, so any needed clock skewing must be done through the row clocks.

4.2 Slack Balancing Techniques

The performance of a design is dictated by the delay of the worst path(s) between registers and/or inputs/outputs. *Slack Balancing* is a set of optimization techniques aiming at improving the performance by minimizing the delay of such critical path(s). It does so by transferring slack from non-critical fast paths to adjacent critical paths. In Agilex, slacks can be balanced using three different techniques: (a) retiming, (b) clock skewing, and (c) time borrowing.

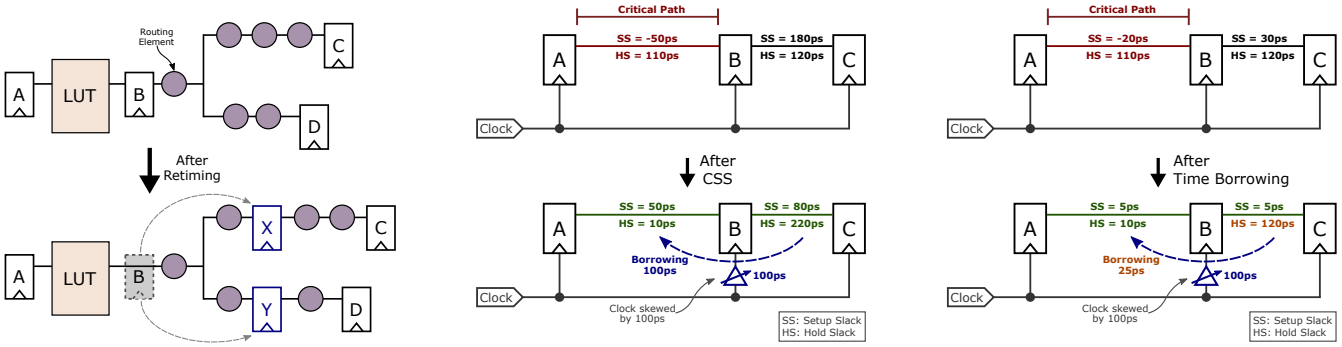
Retiming. Retiming is an optimization technique that changes the location of the design registers, to improve performance. By repositioning the registers, it balances the timing paths and reduces the critical path delay, without changing the design functionality [11]. Figure 15a shows a retiming example. Register B is retimed into X and Y, in order to balance the slacks of B's incoming and outgoing edges, and to minimize the critical path delay. Similar to Stratix® 10, the HyperFlex architecture of Agilex enables hyper pipelining through retiming. Retiming is performed at different stages of the CAD flow.

Clock Skewing. As mentioned earlier, Agilex introduced programmable delay chains on the LAB clocks and at the row clock level. These chains can be used to inject intentional skews on the clock path, in what is known as *Clock Skew Scheduling (CSS)* [5]. Figure 15b shows how skewing the clock of the critical register B can allocate more slack to the critical path AB, by borrowing some from the adjacent and faster path BC. The amount by which a critical register (B in this example) is skewed depends on the preset values of the configurable delays in the delay chain and the setup/hold slacks of the paths affected by the skew.

The DIM HyperFlex registers do not have dedicated delay chains on their clocks, so they can only be skewed using the delay chain of the row clocks. The same applies to DSPs and M20Ks. The ALM flip-flops and the LEIM HyperFlex registers, on the other hand, can be skewed through the dedicated programmable delay chains on the LAB clocks, and/or the delay chain on the row clock(s) driving them. Having two delay chains on the clock path gives more flexibility and, when combined, can create much larger skews, if needed.

Time Borrowing. Clock skew scheduling can only use the predefined set of discrete delays, equivalent to the configurable delays of the delay chain(s) on the clock path. Having finer-grain steps in the delay chain adds flexibility when balancing slacks, but at a higher area and power cost. *Time Borrowing (TB)* offers a compromise by maintaining the same configurable delays but allowing the register to borrow any continuous amount it needs within those preset delays. Continuous time-borrowing was used extensively in past architectures [14, 15] where many registers were implemented as pulse-latches. Agilex, on the other hand, eliminates pulse latches to improve hold closure and reduce variation sensitivity. However, the ALM flip-flops are designed as Master-Slave latches with time borrowing capabilities. Clock skew scheduling can be applied to such registers by skewing both the master and the slave clocks. However, skewing only the master clock creates a *borrow window* between the master and the slave, allowing the ALM flip-flop to borrow any amount it needs, up to a maximum value equal to the skew of the master clock. Using time borrowing, different registers can re-distribute different amounts of slack even if they share a common delay chain.

Time borrowing is favorable in tight setup constraints but can have a negative impact on hold, as shown in Figure 15c. Assuming that the minimum delay in the delay chain is 100*ps*, path BC has very little setup slack to share with the critical path AB. And, skewing register B by 100*ps* will create a worse critical path, as BC will have a setup slack of $-70ps$. Time borrowing can help in such situations where clock skew scheduling is constrained by setup. Still skewing the clock by 100*ps* but allowing B to borrow any amount it needs



(a) Retiming **(b) Clock Skew Scheduling** **(c) Time Borrowing**

Figure 15: Examples of the three slack balancing techniques used in Agilex

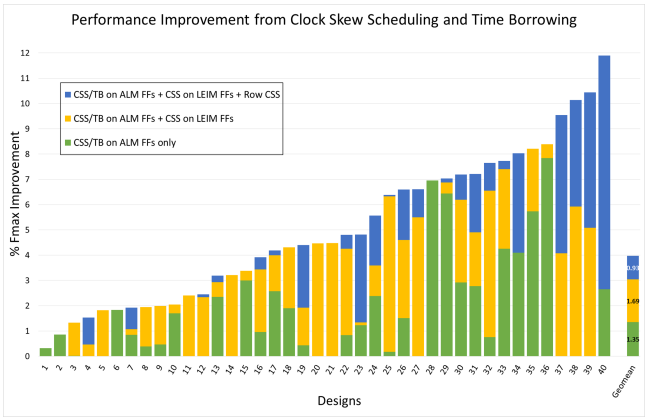


Figure 16: Performance gain with CSS and TB

within that 100*ps* delay, *AB* can now borrow 25*ps* from *BC*, relaxing the critical path. This, however, has a negative impact on hold as *AB* loses the full 100*ps* in hold but *BC* does not gain any, due to the nature of the master-slave latches.

4.3 Slack Balancing Performance Improvement

Quartus uses the three slack balancing techniques in its flow, to optimize the performance on Agilex. Similar to Stratix® 10, the retiming-aware CAD flow of Quartus estimates early on the changes introduced by the post-routing retimer, for a better overall performance [15]. Retiming on Agilex brings, on average, about 15% improvement in maximum frequency (F_{max}), when tested on a set of customer designs.

Clock skew scheduling and time borrowing are performed post-routing and post-retiming, as a last stage of the CAD flow. Thus, these techniques try to balance the slacks on paths that the retimer could not fully balance, and, as such, any performance improvement they bring depends on the retiming solution. For instance, efficiently retimed and well balanced designs might not benefit from any clock skewing. We evaluate the CSS/TB techniques on a set of customer designs and show in Figure 16 their impact on the design frequency. We report the F_{max} improvement for each of the designs that take advantage of these techniques (about 80% of

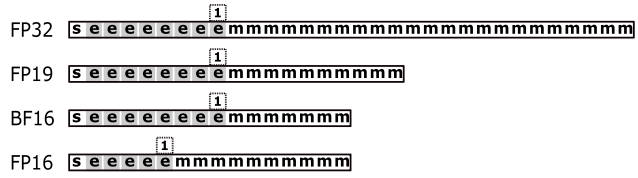


Figure 17: Floating point sign/exp/mant formats

all designs), and the geometric mean across all designs, even those that do not benefit from CSS/TB. The experiments are designed to evaluate the different architectural features that support CSS/TB, by gradually enabling each. First, we allow only the ALM flip-flops to skew/borrow, depending on the setup and hold slacks of their surrounding paths. At this stage, skewing of the row clocks and all HyperFlex registers is disabled, and the frequency gets a 1.35% boost. We then enable LEIM skewing, on top of the ALM CSS/TB, and get an extra 1.69% F_{max} improvement. Finally, we enable row clock skewing, which allows the DIM HyperFlex, DSP, and M20K registers to skew, and extends the skewing range of the ALM and LEIM registers. This adds almost 1% F_{max} improvement, to bring the total to about 4% on average, and up to 12% for individual designs.

5 DSP

The Agilex™ DSP block is an evolution of the Arria 10 [17] and Stratix® 10 [15] DSP architectures, each built on a pair of 18b integer multipliers. These two multipliers can be combined into a single 27b integer multiplier, which forms the basis of an IEEE754 single precision (FP32) multiplier [7]. A FP32 adder is also provided in these blocks. Recently, AI tasks, and in particular deep learning methods, have become high profile applications, and this has driven many of the architectural innovations in the Agilex DSP block. Just as in the introduction of FP32 in Arria 10, new functionality has been carefully constructed to maximally reuse the pre-existing fixed and floating point (FP) datapath elements to minimize area and power impacts. Three new lower precision floating point representations as shown in Figure 17 have been added for AI training and inference, doubling the floating point operation density as shown in Table 1. Embedded performance requirements,

Table 1: Number of multipliers/adders per DSP block

	Stratix 10	Agilex
FP32	1	1
FP16	-	2
FP19	-	2
BFLOAT16	-	2
INT18x19	2	2
INT9	2	4

TFLOPs or TOPs, are continuously increasing overall, so the ratio of DSP blocks to soft logic has been increased by around 50% over Stratix 10.

Figure 18 presents a high-level diagram of the floating-point datapaths in the Agilex DSP block. The left hand side of the diagram shows a logical dataflow, supporting multiplication, addition, and multiply-add. (Multiply-accumulate is also supported, but the feedback paths are not shown in the diagram.) The depicted muxing and the horizontal ingress and egress paths are used to configure the block into a vector mode supporting a low latency recursive reduction, which is beneficial for BLAS-type operation such as vector and matrix multiplication. The multiplier shown comprises two 18×19 multipliers. In floating point mode, these can be combined into the 24b mantissa multiplier of IEEE754 single precision, or the two smaller mantissa multipliers of two FP16 or BF16 operations. In the smaller precisions, a 19b floating point adder sums the two multipliers and casts the result to FP32. The final addition in each block, as well as all subsequent levels in vector mode, is always FP32. The right side of the diagram shows how the CPA of the integer multipliers is actually a compound adder, which performs all of the possible multiplier normalization and rounding combinations in parallel [8].

We will now summarize the key arithmetic features and enhancements for AI, although they can be used for many other applications as well.

5.1 FP16 and FP19

Recent trends in ML/AI research include growing interest in reducing AI training numerics from FP32 to a lower precision, such as FP16. Agilex natively supports two FP16 floating-point operations per DSP block. As in FP32, we chose again not to implement subnormal support for FP16, but we introduce a new variant to specifically address low precision AI training. This inputs FP16 but casts to an FP19 format which has the exponent of FP32 and the mantissa of FP16, allowing the same dynamic range as FP32. (Both FP19 and IEEE754 FP16 with flush to zero can be configured per DSP block.) The following adder is in FP19 format, which is then cast to FP32 for all subsequent additions. The mantissa multiplies of all of the lower precision floating point numbers are extracted from the two 18b integer multipliers, with rounding support implemented in a similar way to the FP32 multiplier [9].

5.2 BFLOAT 16

BFLOAT16 (BF16) was introduced by Google in 2016 [1, 6]. BF16 is not an IEEE standard, but the de-facto industry use model comprises a 16b multiplier input format {1,8,7}, with all additions in FP32. Our BF16 variant specification preceded the Google version, and although the input {1,8,7} and output (FP32) formats are the same as BF16, the first adder immediately following the first pair of multipliers uses FP19. All FP formats can use the low latency reduction circuitry, so all additions in the tree following the first adder level are FP32.

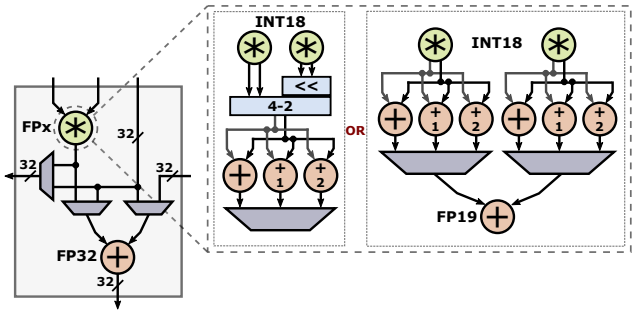


Figure 18: FP32 and FP16/FP19/BF16 datapath flows

In all cases, the 16b floating point formats are only supported as the sum of a pair. If a single independent result is needed, one of the multipliers must be zeroed by zeroing at least one of its inputs. Multiply-add and multiply-accumulate, both with 32b output, are supported on a per DSP block basis.

5.3 INT9

Our previous DSP block architectures included 9×9 multipliers [12–14, 17], primarily used for video and image processing. To make DSP block datapaths more efficient, this DSP-saving feature was later dropped as DSP counts increased.

AI needs many smaller precision multipliers, which have been re-introduced in the Agilex DSP block. Four 9b multipliers (each possibly 8×8 unsigned) are summed together. The result can be output directly from the DSP block, or chained from block to block in cascade. Two 9b multipliers are extracted from each 18b multiplier.

6 BENCHMARK SUITE RESULTS

The overall performance improvement of the Agilex™ programmable fabric over the Stratix® 10 generation of Intel® FPGAs is evaluated on a set of representative customer designs and IP blocks with varying characteristics, as shown in Table 2. These designs represent kernels from a wide variety of FPGA applications — from infrastructure acceleration to wired and wireless communications to test and measurement. They also vary in size from 24k to 427k ALMs, with the average design utilizing approximately 140k ALMs.

All the benchmark designs are implemented (synthesized, optimized, placed, and routed) on an Intel® Agilex engineering sample device (AGFB014R24A2E2VR0) using Quartus Design Suite version 2020.1, which is a preliminary version not yet fully optimized for the new Agilex architecture and does not yet support some of its new features. Figure 19 shows Agilex Fmax performance relative to Stratix 10 (1SG110HN1F43E2VG), measured across the benchmark designs. The geomean speedup across this set is more than 40%, with additional speedup expected with production software.

Table 2: Benchmark characteristics

	Minimum	Median	Maximum
Fmax (MHz)	284	566	951
ALM count	24k	136k	427k
Register count	71k	291k	1.05M
M20K block count	0	732	2600
DSP block count	0	0	1292

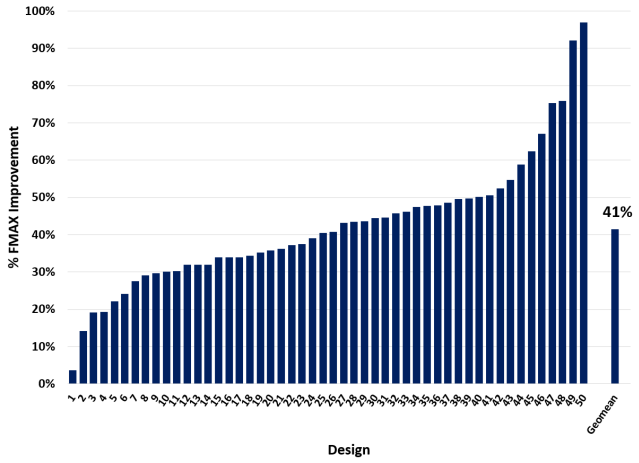


Figure 19: Benchmarked performance improvement

As expected, the results vary design-by-design but every circuit in this benchmark set sees improved performance. This is enabled by the broad-based delay reduction across key circuit elements and fundamental design building blocks. Since Agilex uses significantly lower nominal supply voltage compared to Stratix 10 to improve power efficiency, only a minority of this Fmax increase is due to process improvements.

7 CONCLUSION

Intel® Agilex™ FPGAs feature multiple architectural enhancements to significantly improve the performance of the programmable fabric. Both the inputs and the outputs of the ALM were re-designed and optimized to speed up combinational delays through LUT logic by 50% on average. ALM-level arithmetic was also enhanced with dedicated connectivity to enable efficient mapping of very large adders for emerging encryption workloads. The second-generation HyperFlex™ pipelined routing architecture has been optimized to reduce delay and area overheads of interconnect registers. All routing connections were also recast to be point-to-point, dropping intermediate connections featured in prior FPGA generations and replacing them with a wider variety of shorter wire types. Fine-grain programmable clock skew and time-borrowing were introduced throughout the fabric to augment the slack-balancing capabilities of HyperFlex registers. The Agilex fabric also increases the density of DSP blocks to better address emerging AI/ML/DL applications and augments DSP slices with native support for new low-precision floating-point formats, such as FP16 and BFLOAT16.

Overall, the Agilex fabric is expected to deliver over 40% performance improvement, on average, over a wide variety of applications. Coupled with other new system-level features and capabilities, Agilex FPGAs and SoCs are ideally suited to address a variety of applications spanning acceleration and compute in data centers, wireless 5G, Network Function Virtualization (NFV), automotive, as well as industrial and military/aerospace.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their helpful feedback and suggestions. We would also like to thank Scott Weber, Scott Whitty, Christopher Ebeling, Usman Ahmed, Arun Jangity, Ravi Thiruveedhula, Dustin Do, Hee Kong Phoon, Bee Yee Ng, Wei Yee Koay, Luis Hau, and Ning Cheng for their contributions to the evaluation and definition of the Agilex™ architecture.

REFERENCES

[1] M. Abadi, A. Agarwal, *et al.* TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *CoRR*, abs/1603.04467, 2016.
[2] C. Auth, A. Aliyarukunju, *et al.* A 10nm high performance and low-power CMOS technology featuring 3rd generation FinFET transistors, Self-Aligned Quad Patterning, contact over active gate and cobalt local interconnects. In *2017 IEEE International Electron Devices Meeting (IEDM)*, pages 29.1.1–29.1.4, Dec 2017. doi:10.1109/IEDM.2017.8268472.
[3] V. Betz and J. Rose. FPGA Routing Architecture: Segmentation and Buffering to Optimize Speed and Density. In *Proceedings of the 1999 ACM/SIGDA Seventh International Symposium on Field Programmable Gate Arrays*, FPGA '99, pages 59–68, 1999. doi:10.1145/296399.296428.
[4] C. Ebeling, D. How, D. Lewis, and H. Schmit. Stratix™ 10 High Performance Routable Clock Networks. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '16, pages 64–73, 2016.
[5] J. P. Fishburn. Clock Skew Optimization. *IEEE Trans. Comput.*, 39(7):pages 945–951, July 1990.
[6] Google. Using bfloat16 with TensorFlow Models, 2019. https://cloud.google.com/tpu/docs/bfloat16.
[7] IEEE. IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2008*, pages 1–70, Aug 2008. doi:10.1109/IEEESTD.2008.4610935.
[8] M. Langhammer and B. Pasca. Design and Implementation of an Embedded FPGA Floating Point DSP Block. In *2015 IEEE 22nd Symposium on Computer Arithmetic*, 2015. doi:10.1109/ARITH.2015.18.
[9] M. Langhammer and B. Pasca. Floating-Point DSP Block Architecture for FPGAs. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '15, pages 117–125, 2015. doi:10.1145/2684746.2689071.
[10] M. Langhammer, B. Pasca, and G. Baeckler. High Precision, High Performance FPGA Adders. In *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 298–306, April 2019. doi:10.1109/FCCM.2019.00047.
[11] C. E. Leiserson, F. M. Rose, and J. B. Saxe. Optimizing Synchronous Circuitry by Retiming (Preliminary Version). In R. Bryant, editor, *Third Caltech Conference on Very Large Scale Integration*, pages 87–116. Springer Berlin Heidelberg, Berlin, Heidelberg, 1983.
[12] D. Lewis, E. Ahmed, *et al.* The Stratix II Logic and Routing Architecture. In *Proceedings of the 2005 ACM/SIGDA 13th International Symposium on Field-programmable Gate Arrays*, FPGA '05, pages 14–20, 2005. doi:10.1145/1046192.1046195.
[13] D. Lewis, V. Betz, *et al.* The Stratix Routing and Logic Architecture. In *Proceedings of the 2003 ACM/SIGDA Eleventh International Symposium on Field Programmable Gate Arrays*, FPGA '03, pages 12–20, 2003. doi:10.1145/611817.611821.
[14] D. Lewis, D. Cashman, *et al.* Architectural Enhancements in Stratix™ V. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '13, pages 147–156, 2013. doi:10.1145/2435264.2435292.
[15] D. Lewis, G. Chiu, *et al.* The Stratix™ 10 Highly Pipelined FPGA architecture. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '16, pages 159–168, 2016.
[16] D. Stroobandt, P. Verplaetse, and J. Van Campenhout. Generating Synthetic Benchmark Circuits for Evaluating CAD Tools. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(9):pages 1011–1022, 2000.
[17] B. Vest. Design of a High-Density SoC FPGA at 20nm. In *2014 IEEE Hot Chips 26 Symposium*, August 2014.