High Precision, High Performance FPGA Adders

Martin Langhammer
Intel Corporation
martin.langhammer@intel.com

Bogdan Pasca
Intel Corporation
bogdan.pasca@intel.com

Gregg Baeckler
Intel Corporation
gregg.baeckler@intel.com

***Abstract*—FPGAs are now being commonly used in the datacenter as smart Network Interface Cards (NICs), with cryptography as one of the strategic application areas. Public key cryptography algorithms in particular require arithmetic with thousands of bits of precision. Even an operation as simple as addition can be difficult for the FPGA when dealing with large integers, because of the high resource count and high latency needed to achieve usable performance levels with known methods. This paper examines the architecture and implementation of high-performance integer adders on FPGAs for widths ranging from 1024 to 8192 bits, in both single-instance and many-core chip-filling configurations. For chip-filling designs the routing impact of these wide busses are assessed, as they often have an impact outside the immediate locality of the structures. The architectures presented in this work show 1 to 2 orders magnitude reduction in the area-latency product over commonly used approaches. Routing congestion is managed, with near 100% logic efficiency (packing) for the adder function. Performance for these largely automatically placed designs are approximately the same as for carefully floor-planned non-arithmetic applications. In one example design, we show a 2048 bit adder in 5021 ALMs, with a latency of 6 clock cycles, at 628 MHz in a Stratix 10 E-2 device.**

I. Introduction

FPGAs are increasingly expected to compete with ASICs. Designs such as Microsoft Catapult [1] and Project Brainwave [2] show that FPGAs can be used directly in datacenter. The flexibility of the FPGA has made it particular useful for the construction of Smart NICs [3], with the increased need for security driving the adoption of more complex encryption applications in the soft logic fabric.

Algorithms such as Elliptic-Curve Cryptography (ECC) and RSA require the processing of very large integers, typically orders of magnitude larger than those naturally found in traditional FPGA designs. Newer sieve methods have made smaller RSA widths such as 1024 bits vulnerable to attack, driving the adoption of higher precision such as 4096 bits. Current FPGA architectures perform well in the 32 to 64 bit range, typically with 8 bit [4] and 20 bit [5] granularity. Traditional FPGA pipelining approaches used for the implementation of high-throughput, larger integer addition often has a prohibitive cost, both in area and latency, with a knock-on effect of increased power consumption.

Two key metrics are derived for evaluating the performance of a given architecture: 1/ utility -or the need for many operations per FPGA- which a direct consequence of the logic resource utilization, and 2/ fitting - or the efficiency in the use of the FPGA. Complex designs, especially those using large amounts of arithmetic from the instantiation of numerous carry chains, often map poorly to the FPGA architecture. These designs use large amounts of both logic and routing; the sphere of influence of routing congestion introduced by the design affects logic utilization even far away from the adder. The impact to FPGA utilization is noticed in two ways: first, the low efficiency of the actual mapping, and second, the reduction in the availability of the rest of the FPGA.

These requirements and challenges give us two goals. We need to create an effective large-arithmetic methodology to make FPGAs a competitive platform for encryption, based on PPA (power, performance, area) metrics. At the same time, arithmetic mapping must use the FPGA efficiently, and allow other functions or portions of the application to easily coexist with the arithmetic datapaths.

Consequently, we make two main contributions with this work:

- We develop a way of generalizing the description and construction of very large adders for FPGAs, with a modest area increase over an equivalent ripple carry adder.
- We show a method for achieving a deterministic near 100% logic packing density, for any adder size, while maintaining a very short latency. These methods will leave the unused logic in the device completely untouched, along with the routing to/from, and internal to these regions.

Perhaps most significantly, the density and performance numbers are achieved without hand placement or relative placement directives; instead, implied relative placement within a small carry group is combined with a sparse network of combinatorial logic that can be completely randomly placed. We report on a number of architectures and approaches, allowing the user to choose from many different possibilities to realize their designs.

Our paper is organized as follows. We survey existing methods of adder implementation on current FPGAs, starting with smaller adder optimizations, followed by larger adders in the region of 100 to 1024 bits in Section II. We then introduce our proposed architectures and approaches, with six different structures in Section III. The results, comprising a sweep of several parameters for adder widths ranging from 1024 to 8192 bits, are presented in Section IV. Designs, consisting entirely of these adders, with the logic utilization of the device approaching 100% will be evaluated in Section IV-C. Finally, a summary of future work, which will improve these methods by maintaining relatively constant performance for many instance

designs will follow in Section V along with the conclusions in Section VI.

II. Prior work

Huge adders - that is to say 1000s of bits, rather than large adders of merely 100s of bits - appear to be poorly represented in the literature, at least at the application level. Many published works have focused on optimizing shorter adder density on existing devices.

Over the last several generations of FPGA, density has increased in line with process node scaling, but system speed on a full FPGA has hardly moved. Circuit and architecture improvements have made it possible to run some devices near 1 GHz for example designs, but achieving 500MHz for a complex, chip filling application is very challenging. The shorter adder latencies reviewed here will usually be faster, but the longer adders do not approach this performance level.

In [6], the LUT logic associated with the carry chain is used to compress a portion of the carry function of an adder, and the set of carry bits is used to generate the output of the adder with a second combinatorial layer. This gives a faster adder than a simple ripple carry adder of a given length, with up to a 30% decrease in propagation delay reported. The method is only effective for a relatively narrow set of precisions; typically 64 bits, sometimes more, and only for a single ripple carry adder. The authors state that hand placement is needed to realize the benefits of this approach.

In [7], an alternate approach is taken, which takes advantage of the ternary compression feature in the Xilinx FPGAs, and extends it to support 4:2 compressors. Very high speeds are reported, which appear to approach the clock tree limit of the target FPGAs, but are only given for relatively small precisions.

One of the first detailed studies of large adders on FPGAs was in [8], where the pipelined carry adder was explored. The classical decomposition of a longer adder into shorter segments was developed into a functionally equivalent operation with a prefix structure implemented as a $2n$ length ripple carry adder (RCA). The complexity of any carry forwarding was then contained entirely in the RCA, which occupied only a single level of logic. The latency, and therefore the number of balancing pipelines, was significantly reduced. Area was typically halved, although this was for a relatively small precision. There would likely have been a more significant savings for longer adders, where a deeper pipeline would have been required for the classical method.

The methods of these two publications is then improved in [9], and detailed in [10]. The $2n$ length ripple carry adder (RCA) prefix structure is replaced by a n length RCA, plus some combinatorial logic, potentially cutting the latency through this portion of the design in half. This structure is referred to as the CCC in the paper. Three different large adder architectures are presented. The first, AAM, creates $\{$generate,propagate$\}$ ($\{G, P\}$) pairs by adding segments of the inputs together. The G bit is the carry out of the segment sum, and P is the carry out of the sum plus one, implemented by an adder with the carry in tied to '1'. The $\{G, P\}$ pairs are processed by the CCC and produce a carry in per output segment. The $\{G, P\}$ segment adder results are then muxed by the output carry bits to create the output. The second, CAI, creates the P bits using a comparator. In the targeted Xilinx architecture, the 5LUT logic structure can perform a comparator in half the number of cells compared to an RCA. The output calculation must now be an addition instead of a mux operation, and is implemented by adding the prefix calculated carry bit to the delayed sum of the input segment generate calculation. The number of LUTs on the output segment are the same, although the ripple carry output could restrict placement flexibility slightly compared to the bitwise muxes. This could be offset, however, by the smaller comparator structure on the input side. The final architecture, CCA, calculates $\{G, P\}$ using a pair of comparators. Both input segment values must be routed to the output in this case.

The latencies for all these adders is short; there is little point in making the structure deeper as the RCA in the prefix will quickly become the limiting factor. This is reflected in the reported results, where an adder frequency of 250MHz was specified, which was stated as the likely achievable system speed.

A further development of these approaches is reported in [11], where the CCA architecture is improved, and a somewhat different method is introduced. Both of these, however, are built on the same principle; a carry lookahead structure is fed by a high radix $\{G, P\}$ pair, and the calculated carries are used to create a final binary result. In this work, larger adders up to 1024 bits are reported, but without pipelining - only a combinatorial delay.

Two limitations are evident in all the previous work. First, only a limited scaling is supported, with a maximum length of 1024 bits reported. Secondly - and commonly a trait across many FPGA design studies - is there is no system consideration analyzed. In all cases, only a single instance of described design is implemented, typically without packing efficiency or routing stress reported. In contrast, the goal of our work is that it can be scaled up to device level, with logic and routing stress curated to provide a very dense system at a high frequency.

III. Adder Architectures

We begin with the basic premise of [8], [10], [9], and [11]. We will implement large adders with a high radix $\{G, P\}$ pair generation feeding a prefix structure. We will target effective and efficient use of the FPGA from the start. In [8], [10], [9], considerable effort is spent on analysis of the optimal decomposition granularity of the adders. Our approach will be different. We notice that modern FPGAs have a natural arithmetic granularity, and we will fix our $\{G, P\}$ generation (and in most cases the output calculation as well) to these points. We will use six different adder topologies, and four types of prefix structures. By sweeping these parameters over a wide set of very large adders, the possibilities of FPGA mapping and limitations should become clear. Although

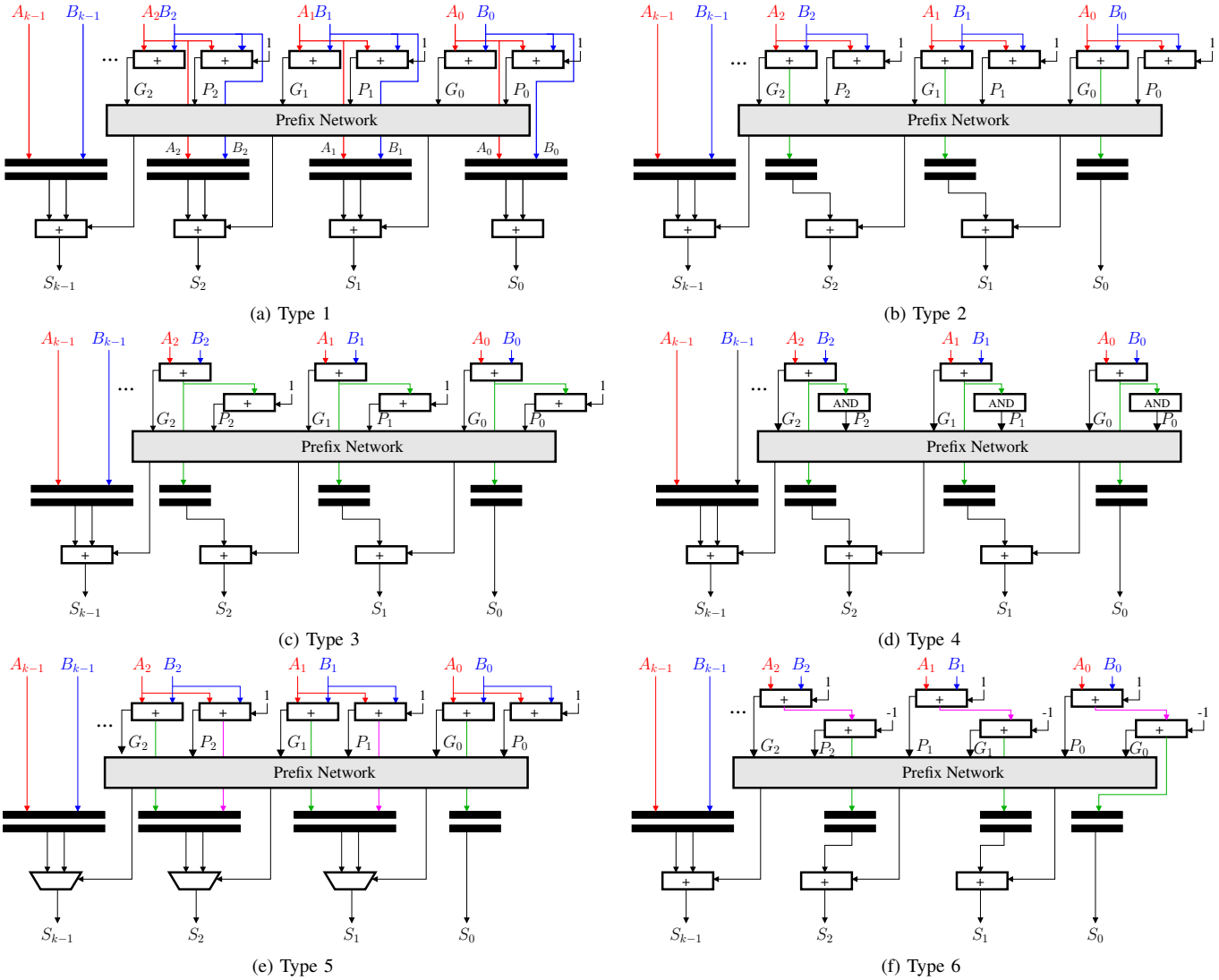

Fig. 1. Proposed wide adder architectures

we will report adder widths from 1024 to 8192 bits, our approach could be continued indefinitely. Performance can remain largely constant as precision increases, and can be finely adjusted using modest pipelining increments. We also show designs with many instances of adders.

Architecture one (Figure 1a) is a canonic representation of our approach; we use it as a baseline for comparing the other architectures. The input word is broken into segments that are the same size as the granularity of the carry chains (20 bits in the case of Stratix 10), minus a bit. This allows the G or P bit to output directly from the carry chain into the local routing with the same characteristics are the rest of the output bits of the LAB (assuming they are used for the architecture in question). Note that in all of the adder architecture figures, P_0 is drawn for regularity of the structure, although it is arithmetically redundant, and not found in the actual core. The two drawn pipeline stages in the figures are only logical representations of the pipelining found inside the core. In every case the input stage (prior to the first logic level) is not pipelined, but the output stage is. Depending on the architecture, the input stage (prior to the prefix) has one or two pipelines. The remaining pipelines are in the prefix stage, which will contain at least one.

The $\{G, P\}$ pair is fed to a prefix network - in this paper we use Brent-Kung [12], Han-Carlson [13], Kogge-Stone [14], and Sklansky [15] structures - which creates a carry vector for the output addition. The first output segment does not require a carry in, and the last input segment does not feed the prefix tree. A depth based re-timing algorithm distributes the number of pipeline stages allocated to the prefix structure to that network.

In architecture one, the input values are also used by the output addition. This requires the delay of two wide vectors through the circuit, and may be a significant contributor to the cost of this implementation. The independence of the output from the input calculation, however, may improve routability.

Architecture two (Figure 1b) is an improved version of architecture one. The sum of each segment (the first 19 bits of the G calculation for the most common 20 bit granularity) is routed to its counterpart output adder segment. This should reduce resource usage, as only a single adder precision word is delayed.

Architecture three (Figure 1c) is an improved version of architecture two. The P calculation is not based on the input segment, but the sum of the input segment, which is the output of the adder creating the G signal. This will hopefully have a small positive impact on routability of the input portion of the design.

Architecture four (Figure 1d) is a further improved version of architecture three. By inspection, a P can only be created when every bit position of the input segment addition (G) is a '1'. This can easily be implemented by a 19 input AND gate (in the case of a 20 bit granularity).

Architecture five (Figure 1e) uses a different output approach. The output segment is either the input segment G or P adder result. Both are delayed to the output, and selected by the respective carry vector bits. This adder should be about the size of the first adder, but the purely combinatorial output structure may give more routing flexibility in some cases.

We first generated a sweep of several hundred experiments of these five architectures. Architecture two was almost always the smallest, and usually had the best PPA metrics. We developed architecture six (Figure 1f) from different aspects of architectures two, three, and four. We wanted to use the idea of architectures three and four of the G or P adder following the other, to reduce routing stress. The problem with this is that the pipeline delay (and therefore the amount of logic) is one cycle greater, so we needed a way to reduce the pipeline delay of the G adder to the output stage. We implemented this by calculating P first, and then subtracting '1' to get the G bit, and it's associated segment addition value. The results, unfortunately, were usually slightly worse than architecture two.

IV. Results

In order to fully explore the performance of the different architectures, we ran a large matrix of compiles over a wide mix of parameters: adder precision, adder granularity, pipeline depth, and prefix tree type, totaling over 400 individual experiments. We cannot analyze more than a small subset of the results here, but will summarize what we have learned using these methods. A larger subset of the results are reported later in this paper as Table III. First, we present results in a tabular form for a single adder type, along with some comments. This tabulation can be explored by the reader so that they can draw their own conclusions. Next, we use plotted graphs a further subset matrix of the results, in order to compare the interaction of different elements of the experiments. Finally, we show and interpret a number of post-fitting floorplans, from individual adders to chip filling designs, and correlate these with our goals.

TABLE I
Performance results for the proposed adder architectures for a 2Kb adder, with a pipeline depth of 6 cycles on a Stratix 10 E-2 device.

Arch.	Prefix Type	Area (ALM)	Prefix Area	FMax (Mhz)
1	Brent-Kung	7659		575
2	Brent-Kung	5021		628
3	Brent-Kung	5558	210	644
4	Brent-Kung	5309		613
5	Brent-Kung	7000		606
1	Han-Carlson	7760		622
2	Han-Carlson	5135		631
3	Han-Carlson	5646	305	646
4	Han-Carlson	5401		578
5	Han-Carlson	7109		654
1	Kogge-Stone	7887		640
2	Kogge-Stone	5254		680
3	Kogge-Stone	5741	430	654
4	Kogge-Stone	5517		648
5	Kogge-Stone	7217		660
1	Sklansky	7677		605
2	Sklansky	5041		636
3	Sklansky	5584	230	631
4	Sklansky	5358		594
5	Sklansky	7004		651

A. Tabulated Results

Table I shows the performance of our proposed adder architectures for a 2Kb adder, having a total pipeline depth of 6 cycles. The lowest reported area was close to 5K ALMs for all prefix network topologies, and was achieved by architecture two. The frequencies of all these architectures are reliably over 600MHz on a Stratix 10 E-2 device. There was a small variation in performance and area with different prefix tree types. On an architectural basis, there appears to be an quasi linear relationship between prefix tree complexity and performance, with about a 5% variation across the experiments.

We can see that the performance is closely correlated to the prefix tree; as the per-segment operations (the creation of G, P, and the final output addition) are mapped to the physical carry chain boundaries in almost all cases, the performance of the majority of the circuit is known and reproducible for all adders implemented with these methods. Any performance reduction from the carry chain segment values is then due to two factors: the distance from the source of the input operands, and the critical paths inside the prefix tree. The prefix tree areas vary slightly from compile to compile, so to make it easier to interpret the table we have averaged out the resources for each tree group, and rounded them to the nearest 5 ALMs. Although the tree areas vary in size considerably, with Kogge-Stone requiring about twice the resources of Brent-Kung, the impact to the overall size of the adder is very small, typically about 5%.

B. Graphed Results

While detailed analysis of the many different options and features can partially done using the portion of the compiled results tabulated in the previous section, the comparison of some aspects of the large adders is better understood graphically.

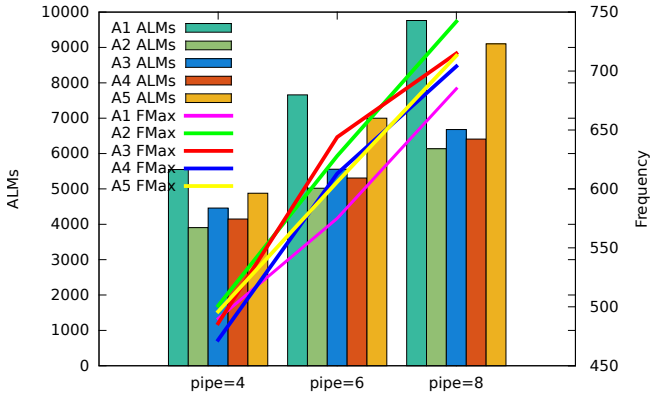

Fig. 2. Area and speed for the proposed architecture across 3 pipelining configurations: 4, 6 and 8 cycles. Adder width is 2Kb and prefix network is Brent-Kung. Target device is Stratix 10 E-2.

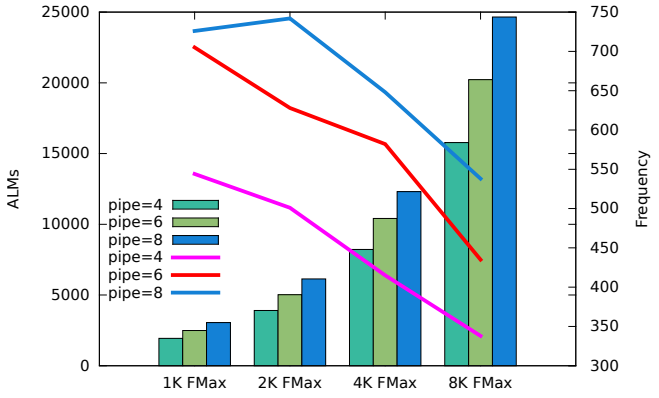

Fig. 4. Area and speed for varying adder lengths: 1Kb, 2Kb, 4Kb ad 8Kb across 3 pipeline depths: 4, 6 and 8 cycles. Adder architecture is type 2 and prefix type is Brent-Kung.

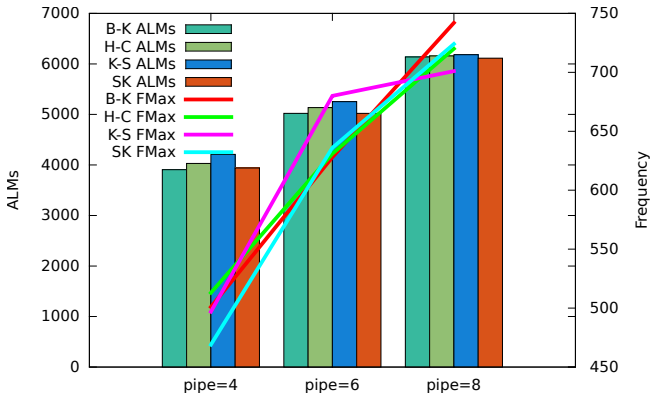

Fig. 3. Area and speed for varying prefix network topologies across 3 pipeline depths: 4, 6 and 8 cycles. The adder is type 2, width 2Kb and the target device is Stratix 10 E-2.

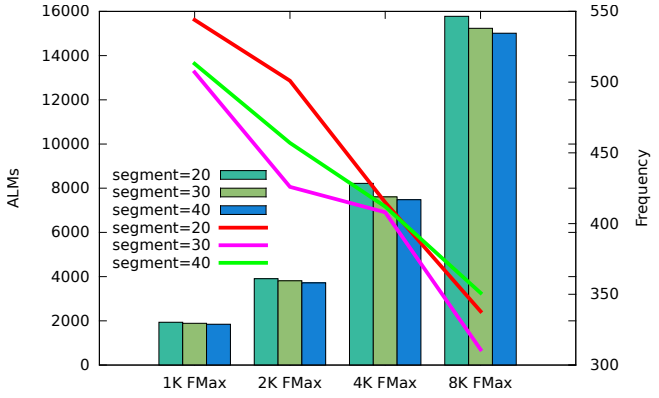

Fig. 5. Area and speed for varying segment lengths: 20, 30 and 40 bits. The adder is type 2 with a Brent-Kung prefix network and a pipeline depth of 4 cycles. Target device is Stratix 10 E-2.

In Figure 2 the five different architecture types are compared for a 2Kb adder, across pipeline depths of 4, 6, and 8 cycles; in this case, the prefix is fixed to Brent-Kung. The canonic type 1 is always the largest, followed closely by type 5. This is directly related to the cost of the delay lines of the input operands to the output stage. In types 1 and 5, two widths are forwarded to the output stage, whereas a single width is forwarded for other structures. Types 2,3, and 4 exhibit the same relative cost to each other independent of the overall pipeline depth of the adder. Types 2 and 3 appear to have the same functional structures, but type 3 is uniformly larger. This is because the propagate calculation follows the generate calculation in type 3, and this single cycle delay (register bank) increases the area accordingly. In type 4, the propagate calculation is combinatorial - and half the size of the carry chain calculation - which reduces the area compared to type 3. By inspection, we can see the effect of a single adder width operand delay, and the effect of removing the equivalent of half an operand calculation. The area of each of these architecture type can be estimated accurately in advance.

The speed of each architecture increases monotonically with latency. Type 1 is typically the slowest, with only a minor difference between the other architectures for this single instance study. Type 3 shows a slight non-linearity at pipeline = 6, but the overall effect in a system design between the different types is probably not significant at the system level.

In Figure 3, the different prefix types are compared over the three pipeline depths, again for a 2Kb adder. Architecture 2 is used for all instances. There is only a small variation in size, as expected; although the prefix trees vary greatly in relative size, their sizes are still quite small compared to the overall size of an adder. One observation from this graph is that as pipeline depth increases to 8, the performance is almost uniformly over 700MHz, which is approaching the expected speed of a single carry chain in the targeted Stratix 10 E-2 device. As a 2Kb type 2 adder has approximately 300 20bit carry chains, this also illustrates the effectiveness of decomposing a larger adder into smaller carry chains, all connected by a purely combinatorial ($i.e.$ not containing a carry chain) arithmetic structure such as a prefix tree.

Speed does vary by prefix type. At pipeline = 6 all trees have approximately the same performance except for the Kogge-

Stone tree, which is approximately 10% faster. At pipeline = 8, however, the Kogge-Stone tree is about 5% slower; all other trees are fairly closely matched in performance. Looking at all of the other performance variations per tree type across the entire dataset does not appear to show a significant difference. One conclusion from this is that the tree type could be varied during the timing closure phase of a project to give different fitting options.

In Figure 4, the effects of pipelining at different lengths is explored. Each area group (for adder widths 1Kb, 2Kb, 4Kb, and 8Kb) shows the impact of pipeline depths 4, 6, and 8, respectively. As expected, the total area of an adder at a given pipeline depth is directly proportional to the adder width, and the incremental cost of an adder is also linear to the pipelining. What is shown in the speed graph is how the performance degradation with increasing adder width can be compensated by increasing pipelining, almost deterministically. Each frequency line shows the approximately linear reduction in performance per level of pipelining: the lowest line is for pipeline = 4, the next for pipeline = 6, and the top line for pipeline = 8. A 2Kb adder with pipeline = 6 will run at around 650 MHz, but a 4Kb adder with this pipeline depth is about 15% slower. By increasing the pipeline depth to 8, however, the 4Kb adder will match the performance of the 2Kb adder. As the performance degradation follows an almost linear decrease, we can confidently map the performance of an individual instance vs. latency. We can expect that a pipeline depth of 5 would fall almost perfectly in the middle of the depth 4 and depth 6 curves.

We can also see that we cannot exceed 750MHz, no matter how deeply pipelined the adder is; as explained previously, this limit is set by the performance of a 20bit carry chain segment. Reducing the segment length will complicate the fitting - and in turn significantly increase the complexity of the prefix tree. The reduction in performance of the larger adders is likely caused by the complexity of the prefix trees, so any attempt to shorten the segment length below the natural hardware boundaries of the FPGA will likely be counter-productive.

Figure 5 shows the effect of increasing segment lengths, using virtual, rather than physical segments, but again aligned to natural hardware boundaries. Architecture 2 with a pipeline depth of 4 is used for all adders. There is a small, monotonic decrease in area with increasing segment length. There are two reasons for this: 1/the prefix tree complexity is reduced, and 2/ the overhead in routing out the calculated G and P bits. The bottom frequency line corresponds to the 30 bit segment line - aligning to multiples of hard boundaries (20 and 40 bits in the case of Stratix 10) results in a higher performance than a soft boundary alignment.

C. Floorplans - Single and Multiple Instances

We created a number of designs with increasing numbers of large adders, choosing a 2Kb, architecture 2, pipeline 6, with a Brent-Kung prefix tree for all cases. The fitting methodology used virtual pins (virtual pins occupy half ALM each), which can be used to represent physical source and sink locations

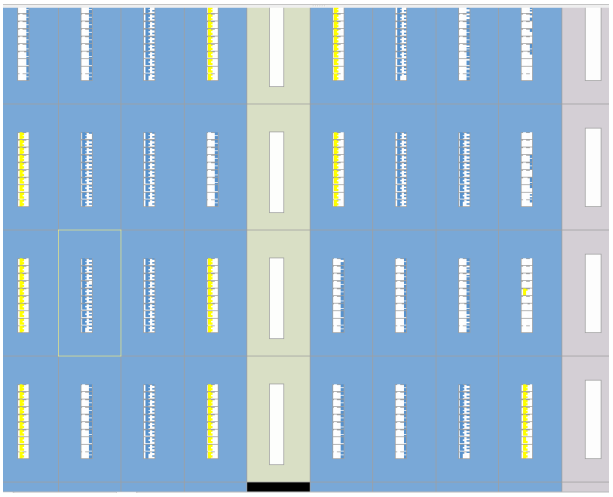

Fig. 6. Logic Packing for Adder

TABLE II
SINGLE AND MULTI INSTANCE ADDER RESULTS

Units	Area (ALM)	Area(%)	Speed (Mhz)	Time
1	8K	1%	628	-
50	404K	44%	478	4h39m
100	810K	87%	332	10h00m
110	886K	95%	295	12h01m

required to tie a larger design together. In a real-world application, a set of large adders would likely only occupy a small percentage of the device, as the application wrapped around the adders would require the majority of the logic and other resources. The results for our runs are summarized in Table II. Note that all of compiles were pushbutton, without any pre-compile floorplanning or fitting directives.

Figure 6 shows a portion of a single adder instance. Note that virtually every single ALM in every LAB is utilized - this may be only using the carry chain, only the LUT, or a combination of the two. Locations marked in yellow denote virtual pins.

A floorplan for a multi-core design containing 50 of the described adder instances is shown in Figure 7. We can see that a subset of the device is used, with the logic evenly distributed amongst the portion of the utilized resources. This will allow system logic to be inserted in between the adders, as well as a significant amount of the device being completely untouched.

As the number of instances increases, both the gaps between the adders and the untouched portions of the device reduced uniformly. The floorplan in Figure 8 shows an almost completely packed device, with 95% logic utilization. This is an exceptionally high density for a complex FPGA design, especially one consisting only of arithmetic functions. This shows that our carry chain granularity aligned methodology allows for a deterministic use of FPGA logic. Speed, however, declined, with an almost linear reduction with increasing density. Compile time also increased only linearly, even as utilization approached 100% - this again demonstrates the

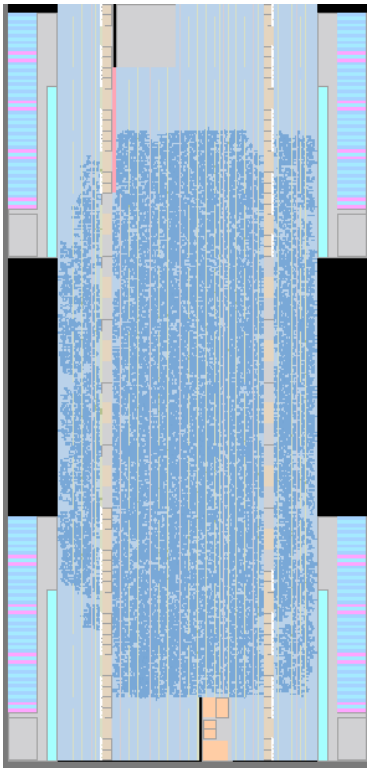

Fig. 7. 50 2Kb Adder Instances

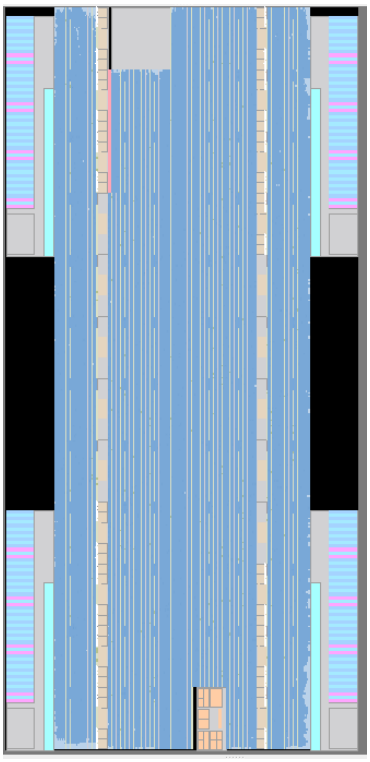

Fig. 8. 110 2Kb Adder Instances

effectiveness of the alignment design method.

We analysed the critical paths in the more fully packed designs and found that Quartus was allowing segments from different adders to mix. Although this did not impact the performance of the vast majority of the logic (which was all snapped to LAB boundaries), some of the paths in the prefix tree spanned a considerable portion of the device.

V. Future Work

The results of these adders, and in particular the ability to almost completely fill a large FPGA with arithmetic without any intervention, gives us several paths for future work.

We now have deterministic fitting, with only a linear degradation in performance. As the majority of the logic operates at a known speed, we can concentrate on the subset of the logic with the critical paths. To allow all users to take advantage of the potential for high performance arithmetic on large or huge numbers on FPGA, we will look for methods that do not require expert knowledge of FPGA tools. Our next step will be to develop tools that enforce a tighter physical grouping of logically related arithmetic operations during placement. We will also improve the register retiming of the prefix networks, to an predicted span-based method from the current logical depth-based method. These approaches will likely vary significantly from network to network because of the differing topologies.

Next, we will extend our arithmetic mapping techniques to large multipliers. The construction of larger multipliers from smaller multipliers, whether direct or via multi-way polynomial methods, requires large adders. Now that we have demonstrated the effective implementation of these adders, we can combine our planned research on virtual floorplanning with the use of the embedded multiplier components on the FPGA.

VI. Conclusions

In this paper, we have shown that any number of very large adders, often with many thousands of bits of precision, can be fit into current FPGAs, with deterministic results. Whether a single adder, or many instances, the effective utilization can be accurately estimated in advance, including up to almost 100% device use. Single instance large adder performance is near the maximum expected performance for small adders that are commonly implemented in FPGA, and an orderly performance reduction is observed even as the device approaches being completely full.

The analysis of the many experiments shows a clear path for new research towards improving the performance of single instances of these adders to the fastest of the current architectures, as well as flatten the speed reduction observed as many are used.

The enhanced arithmetic capability of the FPGA, using existing device architectures available to all users, will open up new application areas, as the use and implementation of large arithmetic can now be confidently specified and planned.

TABLE III

FULL TABLE: L - ADDER LENGHT, P - PIPELINE DEPTH, PRX - PREFIX TYPE, T - ADDER CONFIGURATION TYPE

L	P	PRX	T	CORE ALMs	TREE ALMs	FMax E-2	L	P	PRX	T	CORE ALMs	TREE ALMs	FMax E-2	L	P	PRX	T	CORE ALMs	TREE ALMs	FMax E-2	L	P	PRX	T	CORE ALMs	TREE ALMs	FMax E-2
			1	2754		493				1	5550		490				1	11546		419				1	22359		361
			2	1935		544				2	3907		501				2	8219		415				2	15776		338
		B-K	3	2211	40	589			B-K	3	4460	110	486			H-C	3	9328	635	447			B-K	3	17972	495	361
			4	2094		578				4	4184		472				4	8845		414				4	16933		349
			5	2420		526				5	4879		496				5	10108		424				5	19590		314
			6	2260		572				6	4548		476				1	11873		404				6	18378		349
			1	2809		546				1	5678		495				2	8551		424				1	23326		350
			2	1989		546				2	4030		513		4	K-S	3	9705	975	430				2	16607		324
		H-C	3	2254	100	578			H-C	3	4587	220	503				4	9237		429			H-C	3	18826	1735	356
			4	2134		572				4	4319		514				5	10436		453				4	17779		364
	4		5	2472		595		4		5	5004		482				1	11238		378		4		5	20420		324
			1	2865		556				1	5913		469				2	7970		413				1	24291		329
			2	2046		501				2	4209		497			SK	3	9088	350	431				2	17631		354
		K-S	3	2311	150	580			K-S	3	4715	440	523				4	8571		445			K-S	3	19729	2360	363
			4	2182		601				4	4486		512				5	9916		417				4	18739		382
			5	2524		600				5	5176		513											5	21159		329
			1	2789		563				1	5599		455				1	15684		546				1	22561	730	318
			2	1969		562				2	3941		469				2	10410		582				2	16007		349
		SK	3	2234	75	560			SK	3	4480	150	459			H-C	3	11377	730	584			SK	3	18246		337
			4	2111		574				4	4244		514				4	10858		527				4	17156		363
			5	2450		527				5	4930		467				5	14382		523				5	19833		328
																	1	16036		530							
			1	3811		693				1	7659		575				2	10770		575				1	30774		398
			2	2496		705				2	5021		628		6	K-S	3	11675	1045	582				2	20227		436
		B-K	3	2769	90	757			B-K	3	5558	210	644				4	11196		536			B-K	3	22368	970	443
			4	2639		665				4	5309		613				5	14721		552				4	21409		434
			5	3473		742				5	7000		606				1	15382		533				5	28191		431
			6	2810		670				6	5633		623			SK	2	10112		581				6	22709		494
			1	3841		704				1	7760		622				3	11189	485	593				1	31337		478
			2	2526		707				2	5135		631				4	10759		577				2	20791		467
		H-C	3	2786	120	706			H-C	3	5646	305	646				5	14071		553			H-C	3	22860	1475	477
			4	2683		697				4	5401		578											4	21892		471
1024	6		5	3497		711	**2048**	6		5	7109		654	**4096**			1	19575		570	**8192**	6		5	28760		453
			1	3848		690				1	7887		640				2	12310		648				1	32511		441
			2	2531		704				2	5254		680				3	13384		614				2	21954		444
		K-S	3	2807	120	696			K-S	3	5741	430	654			B-K	4	12835	600	624			K-S	3	23726	2715	478
			4	2684		702				4	5517		648				5	18252		651				4	22781		443
			5	3506		736				5	7217		660				6	13510		600				5	29950		447
			1	3806		702				1	7677		605				2ML	9559		566				1	30817		467
			2	2487		705				2	5041		636				1	19713		593				2	20254		456
		SK	3	2763	85	716			SK	3	5584	230	631				2	12448		651			SK	3	22425	990	496
			4	2651		701				4	5358		594			H-C	3	13523	700	609				4	21490		484
			5	3467		748				5	7004		651				4	12970		637				5	28198		440
															8		5	18413		632							
			1	4862		740				1	9762		685				1	19834		557				1	39167		529
			2	3055		726				2	6137		742				2	12567		673				2	24651		538
		B-K	3	3329	140	754			B-K	3	6679	290	715			K-S	3	13671	875	588			B-K	3	26789	1220	537
			4	3194		787				4	6409		704				4	13105		619				4	25737		504
			5	4525		750				5	9101		713				5	18552		589				5	36560		512
			6	3362		707				6	6742		696				1	19560		576				6	27083		518
			1	4855		745				1	9793		720				2	12298		651				1	39505		553
			2	3047		772				2	6158		720			SK	3	13409	625	588				2	24972		498
		H-C	3	3321	130	755			H-C	3	6705	335	708				4	12963		620		8	H-C	3	27165	1495	536
			4	3191		749				4	6436		716				5	18294		649				4	26103		534
	8		5	4518		745		8		5	9133		700											5	36934		555
			1	4842		722				1	9812		708											1	39343		517
			2	3029		700				2	6182		701											2	24794		574
		K-S	3	3306	140	723			K-S	3	6731	335	706											3	26949	1375	559
			4	3192		723				4	6467		681											4	25996		513
			5	4503		727				5	9157		701											5	27777		461
			1	4846		740				1	9739		702														
			2	3036		753				2	6114		724														
		SK	3	3310	120	704			SK	3	6655	240	719														
			4	3192		779				4	6443		725														
			5	4508		773				5	9091		701														

REFERENCES

[1] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Y. Xiao, and D. Burger, "A reconfigurable fabric for accelerating large-scale datacenter services," *IEEE Micro*, vol. 35, no. 3, pp. 10–22, May 2015.

[2] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi, S. Heil, P. Patel, A. Sapek, G. Weisz, L. Woods, S. Lanka, S. K. Reinhardt, A. M. Caulfield, E. S. Chung, and D. Burger, "A configurable cloud-scale DNN processor for real-time AI," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, June 2018, pp. 1–14.

[3] D. Firestone, A. Putnam, S. Mundkur, D. Chiou, A. Dabagh, M. Andrewartha, H. Angepat, V. Bhanu, A. Caulfield, E. Chung, H. K. Chandrappa, S. Chaturmohta, M. Humphrey, J. Lavier, N. Lam, F. Liu, K. Ovtcharov, J. Padhye, G. Popuri, S. Raindel, T. Sapre, M. Shaw, G. Silva, M. Sivakumar, N. Srivastava, A. Verma, Q. Zuhair, D. Bansal, D. Burger, K. Vaid, D. A. Maltz, and A. Greenberg, "Azure accelerated networking: SmartNICs in the public cloud," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018.

[4] *UltraScale Architecture and Product Overview - Advance Product Specification*, 2014, http://www.xilinx.com/support/documentation/data_sheets/ds890-ultrascale-overview.pdf.

[5] *Intel Stratix®10 GX/SX Device Overview*, 2018, https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/stratix-10/s10-overview.pdf.

[6] P. Kllstrm and O. Gustafsson, "Fast and area efficient adder for wide data in recent Xilinx FPGAs," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, Aug 2016, pp. 1–4.

[7] M. Kumm and P. Zipf, "Efficient high speed compression trees on Xilinx FPGAs." 2014.

[8] F. de Dinechin, H. D. Nguyen, and B. Pasca, "Pipelined FPGA adders," in *International Conference on Field Programmable Logic and Applications*. IEEE, Aug. 2010.

[9] H. D. Nguyen, B. Pasca, and T. B. Preußer, "Fpga-specific arithmetic optimizations of short-latency adders," in *International Conference on Field Programmable Logic and Applications*. IEEE, 2010.

[10] B. Pasca, "High-performance floating-point computing on reconfigurable circuits," Ph.D. dissertation, École Normale Supérieure de Lyon, Lyon, France, Sep. 2011. [Online]. Available: http://tel.archives-ouvertes.fr/docs/00/65/41/21/PDF/Bogdan_PASCA-Calcul_flottant_haute_performance_sur_circuits_reconfigurables_2011.pdf

[11] M. Rogawski, E. Homsirikamol, and K. Gaj, "A novel modular adder for one thousand bits and more using fast carry chains of modern FPGAs," in *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, Sep. 2014, pp. 1–8.

[12] H. Kung and R. Brent, "A regular layout for parallel adders," *IEEE Transactions on Computers*, vol. 31, pp. 260–264, 03 1982. [Online]. Available: doi.ieeecomputersociety.org/10.1109/TC.1982.1675982

[13] T. Han and D. A. Carlson, "Fast area-efficient VLSI adders," in *1987 IEEE 8th Symposium on Computer Arithmetic (ARITH)*, May 1987, pp. 49–56.

[14] P. M. Kogge and H. S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," *IEEE Transactions on Computers*, vol. C-22, no. 8, pp. 786–793, Aug 1973.

[15] J. Sklansky, "Conditional-sum addition logic," *IRE Transactions on Electronic Computers*, vol. EC-9, no. 2, pp. 226–231, June 1960.