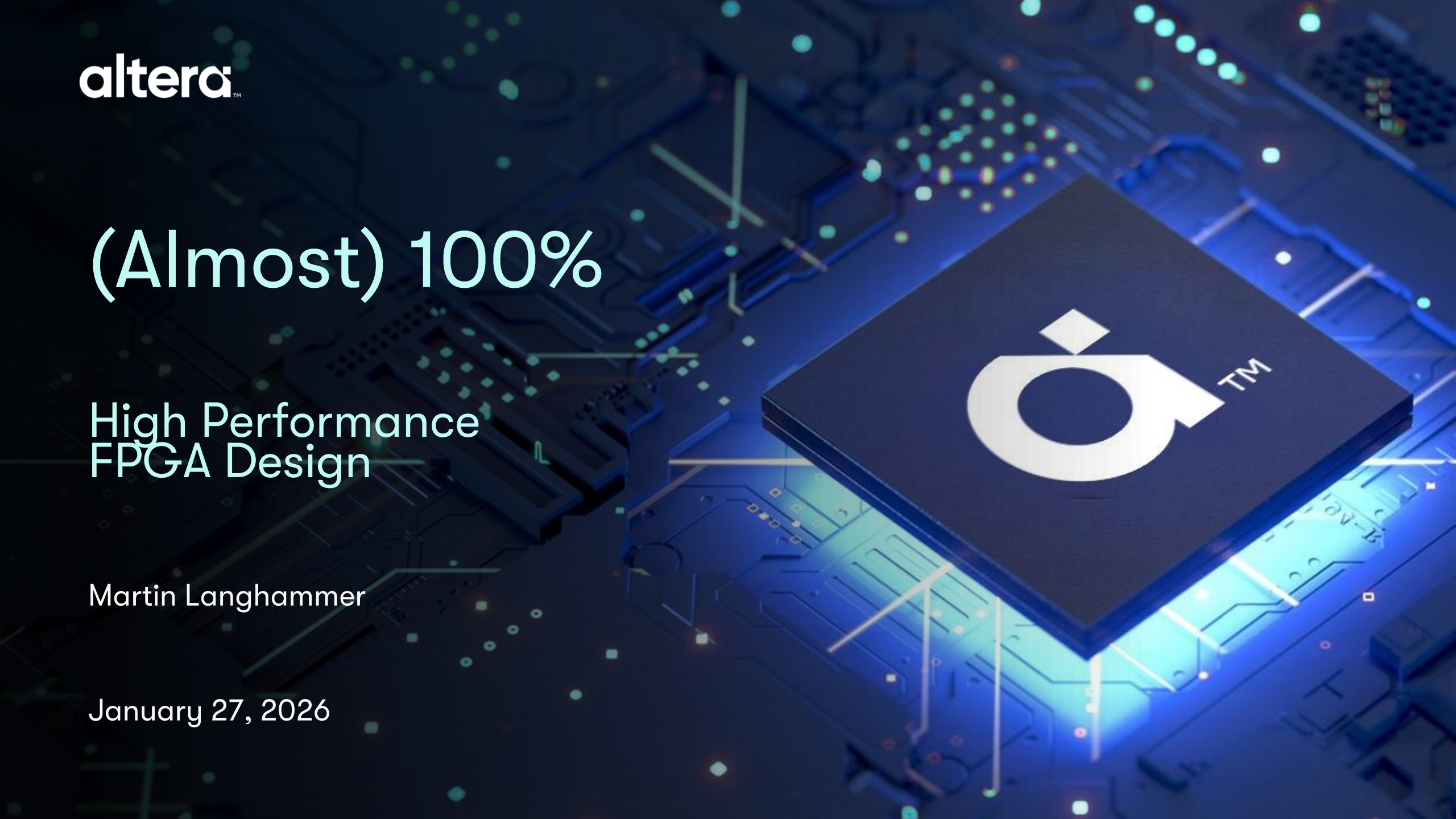
altera™

(Almost) 100%

High Performance FPGA Design

Martin Langhammer

January 27, 2026



Mindset

Method

Micro

Midi

Macro

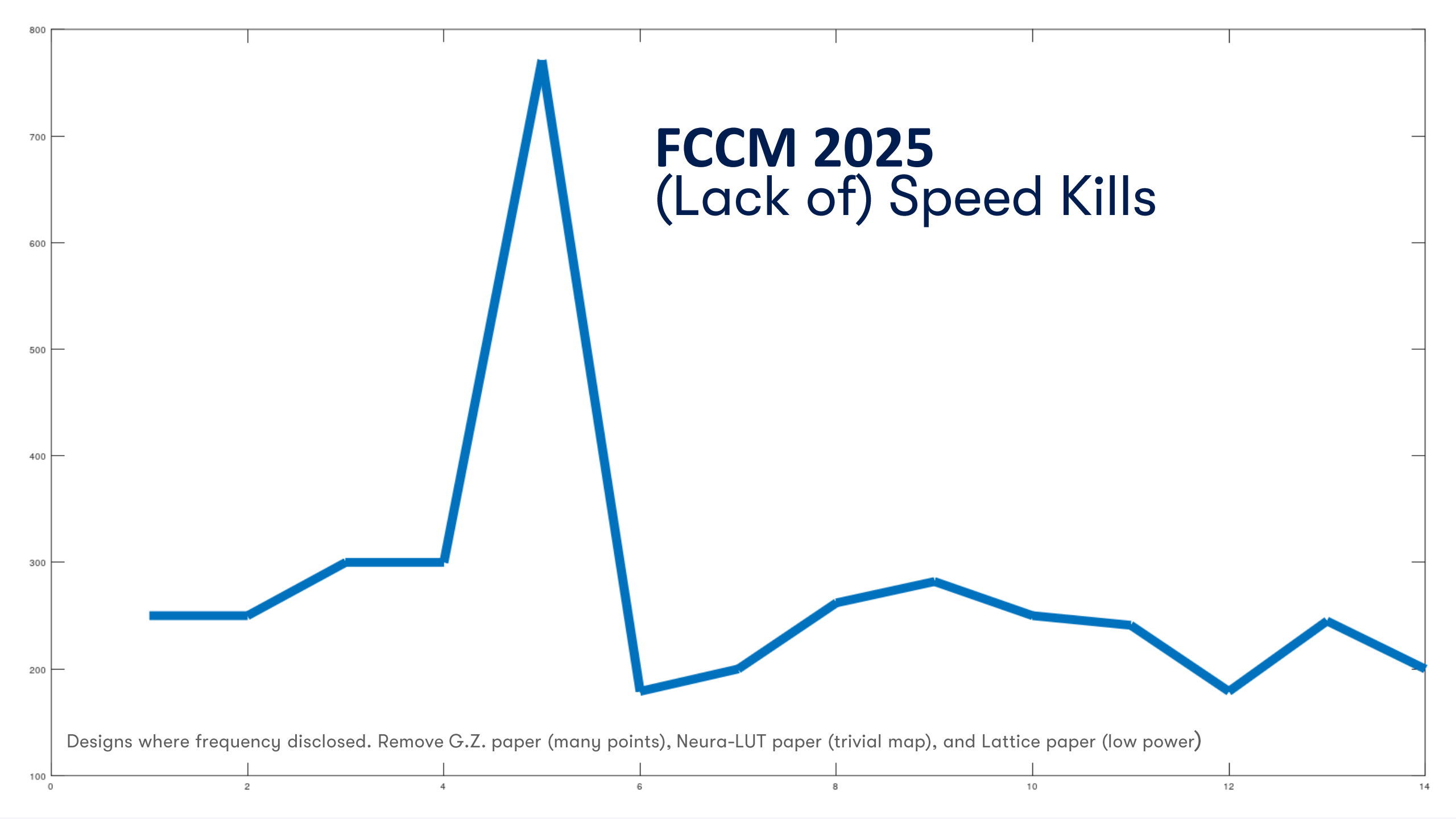
~1M LUT

~1 GHz


41

Clock Network: 1GHz
Memory: 1GHz/600MHz
DSP: 771 MHz




FCCM 2025
(Lack of) Speed Kills
800
700
600
500
400
300
200
100
0
2
4
6
8
10
12
14
Designs where frequency disclosed. Remove G.Z. paper (many points), Neura-LUT paper (trivial map), and Lattice paper (low power)



FPGA is a bad ASIC but a pretty good SoC

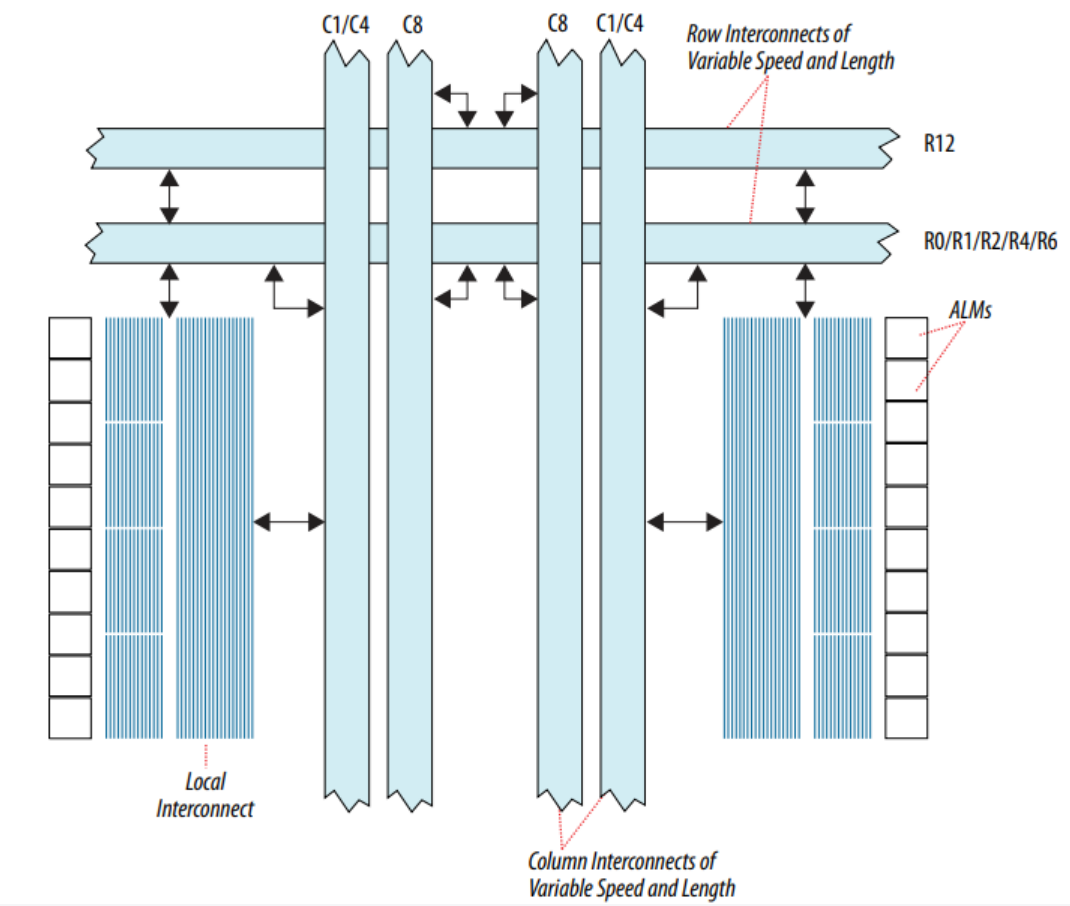
amorphus /əˈmɔːr.fəs/ *without physical shape or form*

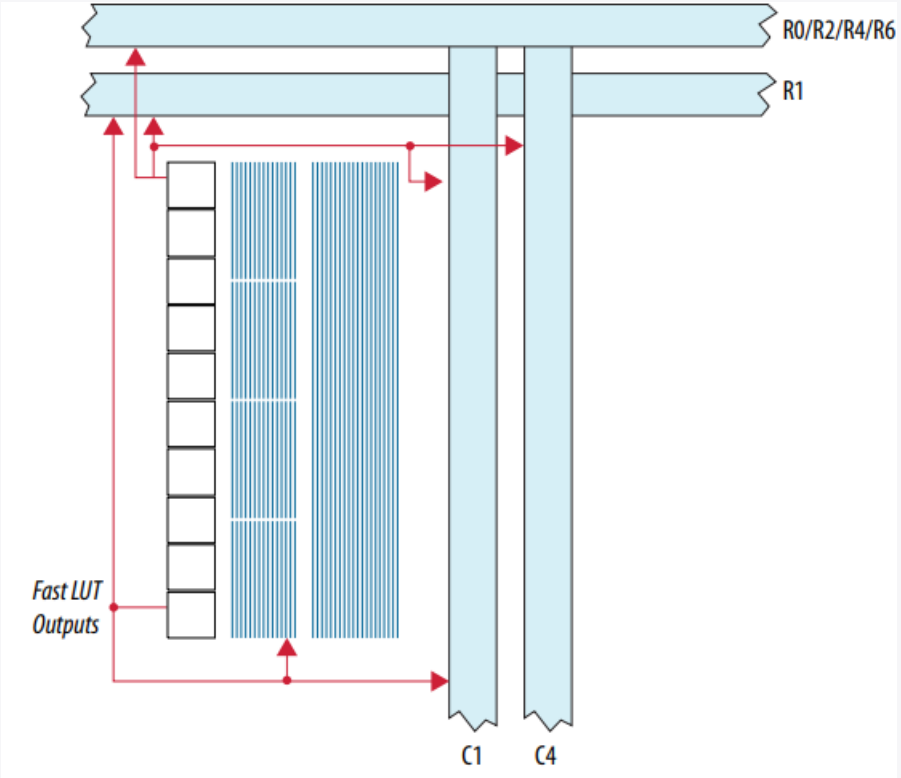
fungible /ˈfʌn.dʒ.ə.bᵊl/ *can be easily exchanged for something else*


Customers buy Logic, but they pay for Routing

The FPGA is not about Logic

It is, however, all about routing







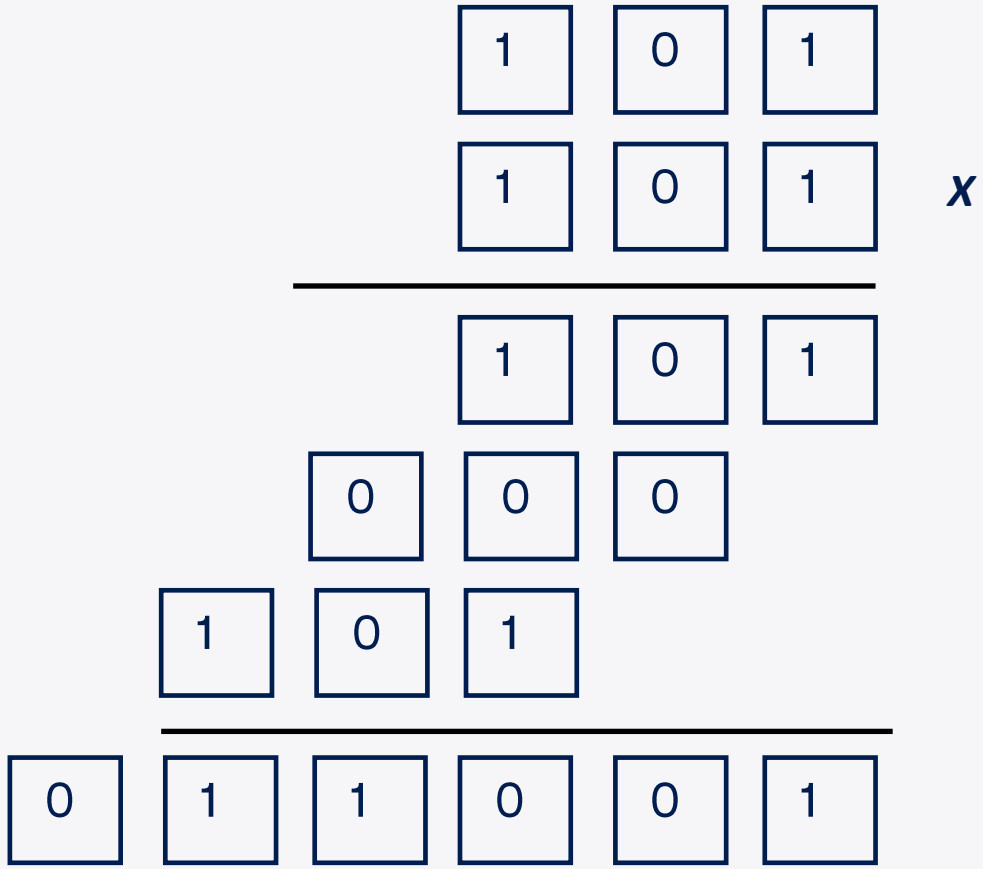
This would never be tolerated in the SW world

- Hey, let's write some code and see what happens
- Great SW engineers don't just understand (for example) C
  - Memory hierarchy and cache algorithms considered
  - Maximize execution time, minimize memory time
  - Heap, stack, and recursion depth
- Typical FPGA approach
  - Write (i) code (ii) compile (iii) try to fix fit problems
  - How about (i) ***work out*** fit problems (ii) code (iii) compile
- SW is different – performance is based on an average. FPGA - performance is based on the worst path

Method - Micro

Trivial example – with non-trivial implications

- Pencil and paper 3x3 multiplier
- 5 x 5 = 25



Pedantic 3x3 Multiplier

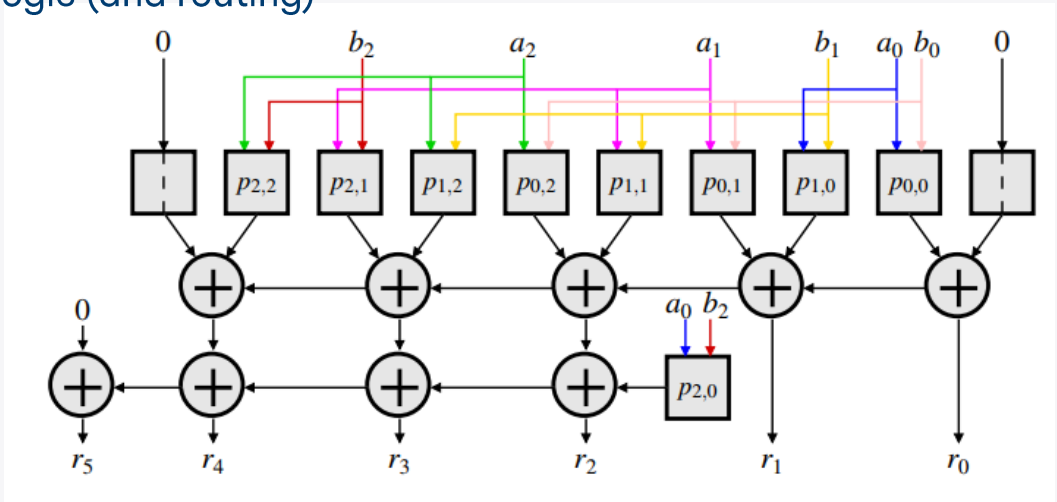
- 3x3 multiplier useful for AI inference
- A trivial case – or is it?
- Saving 1 LUT significant if you have 50K multipliers
- Logic savings overshadowed by routing optimization and simplification
- Second level carry chain creates device placement restriction

Doesn't depend on carry chain

3 high column affects rest of multiplier

Column	5	4	3	2	1	0
PP0	0	0	0	$p_{0,2}$	$p_{0,1}$	$p_{0,0}$
PP1	0	0	$p_{1,2}$	$p_{1,1}$	$p_{1,0}$	0
PP2	0	$p_{2,2}$	$p_{2,1}$	$p_{2,0}$	0	0

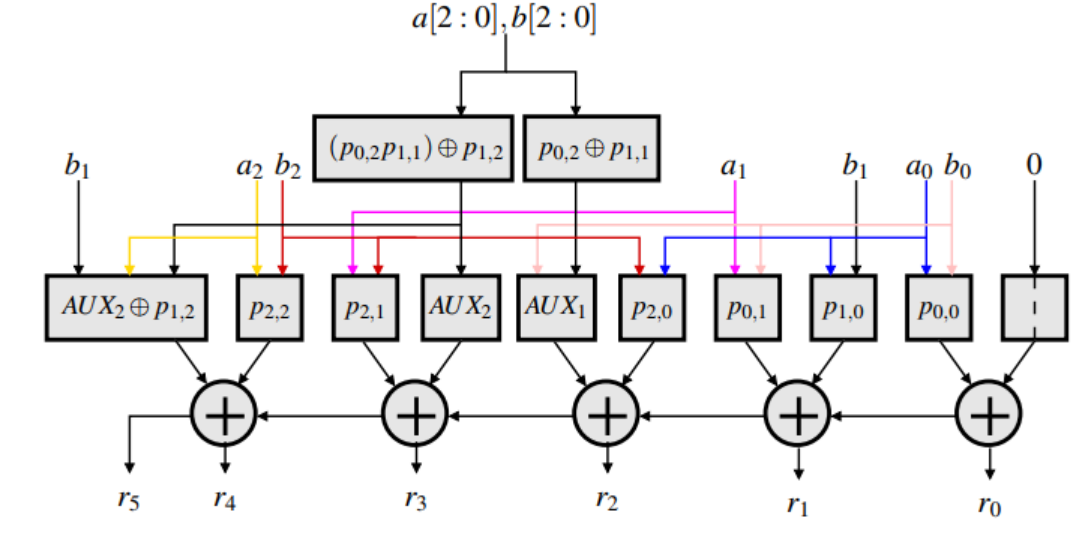
Unused logic (and routing)



Regularized 3x3 Multiplier

- Refactor logic in any column to more than 100% if required
- Refactor any logic over 100% to a function of 1 bit + 100% of remaining capability
- Use out of band functions for other partially used columns

Column	5	4	3	2	1	0
PP0	0	$p_{2,2}$	$p_{2,1}$	$p_{2,0}$	$p_{0,1}$	$p_{0,0}$
PP1	0	$AUX_2 \oplus p_{1,2}$	AUX_2	AUX_1	$p_{1,0}$	0

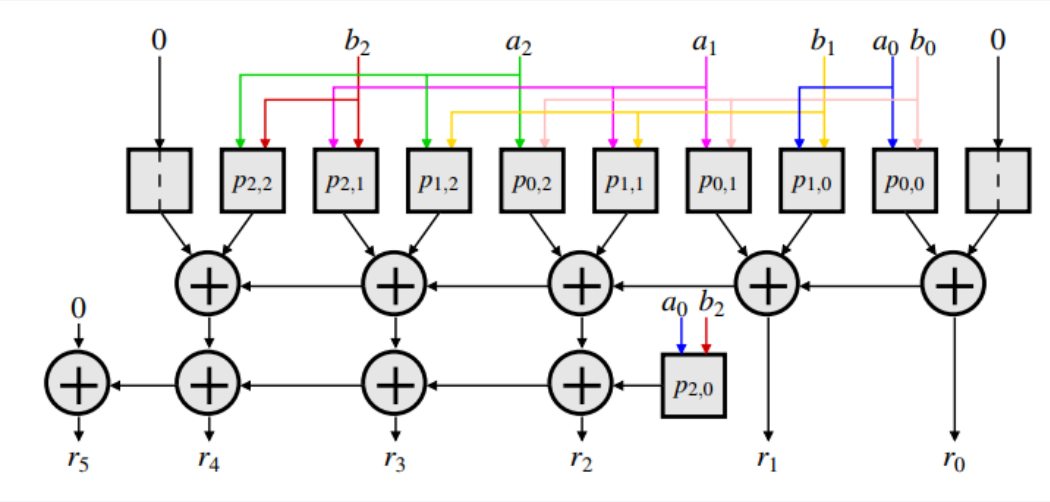


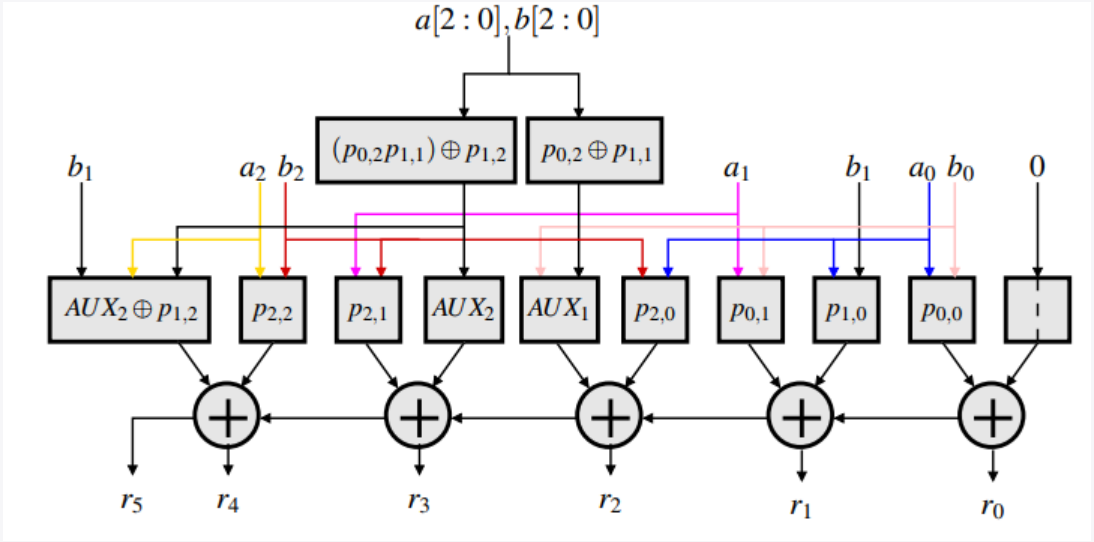
Method - Midi

True Cost of Bad Design

- "I am only getting 55% device utilization on my AI design"
  - It is actually worse than that
- Carry chain granularity is 10 bits
  - Your 3x3 is *really* taking 10 LUTs
  - It should take 3 LUTs
  - You are actually only getting 30%!

(additional techniques for getting around the carry boundaries in the next section)

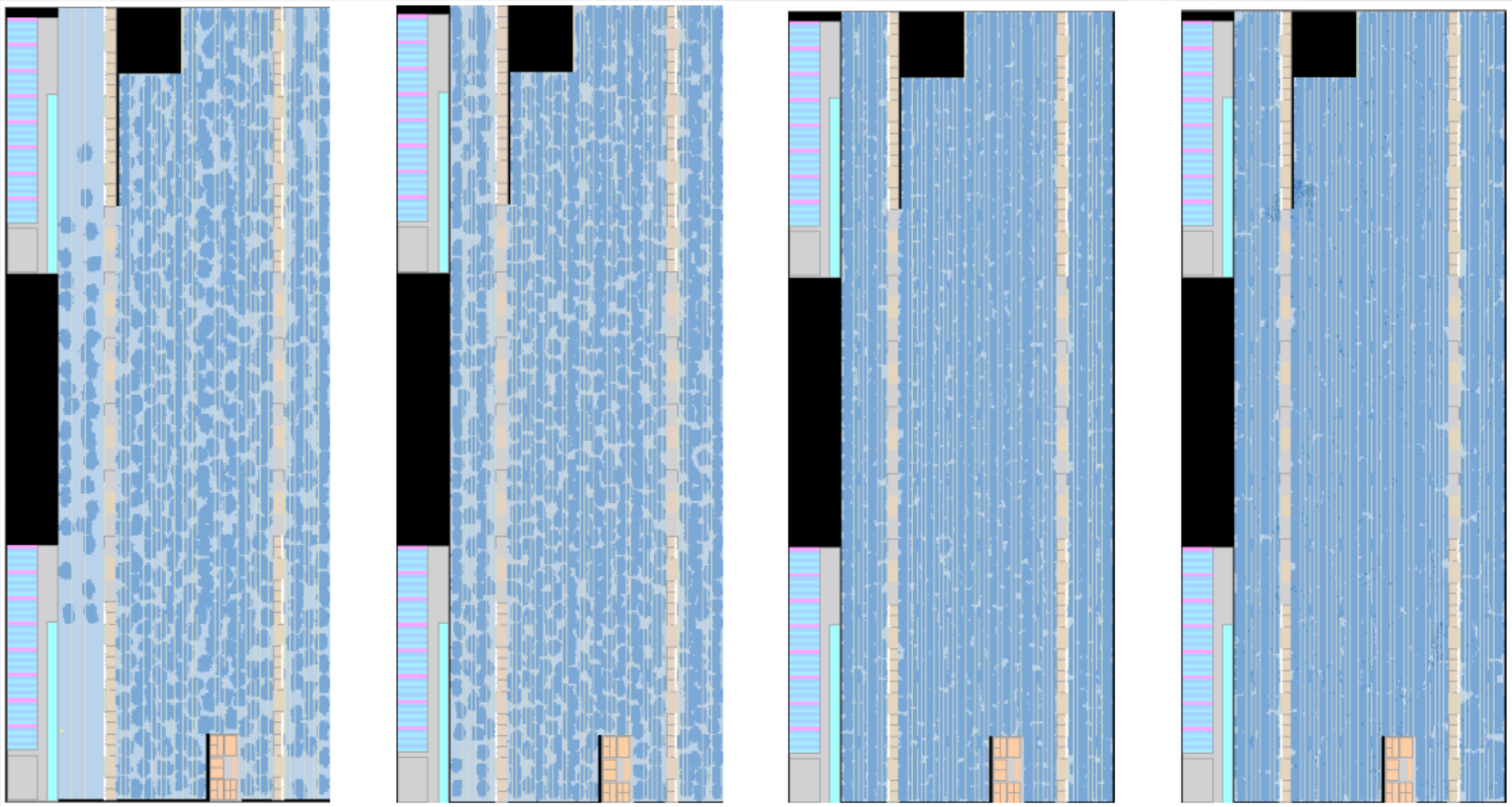




Continuous Arithmetic

- All arithmetic components decomposed into associated and non-associated blocks
- Associated blocks are assembled on a LAB-by-LAB basis
- Timing and locality for non-associated blocks not critical
- Hard depopulation performs useful (non) function
- Achieves 97% typical logic packing

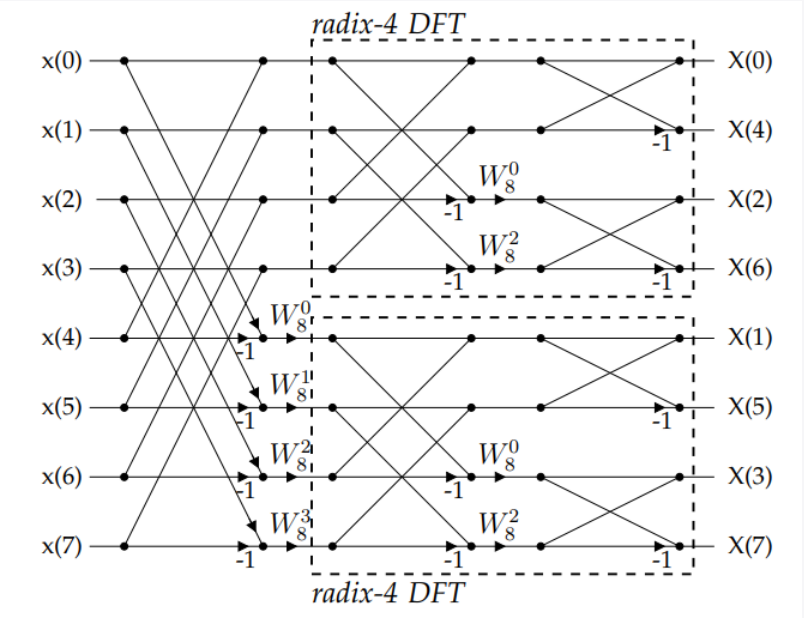


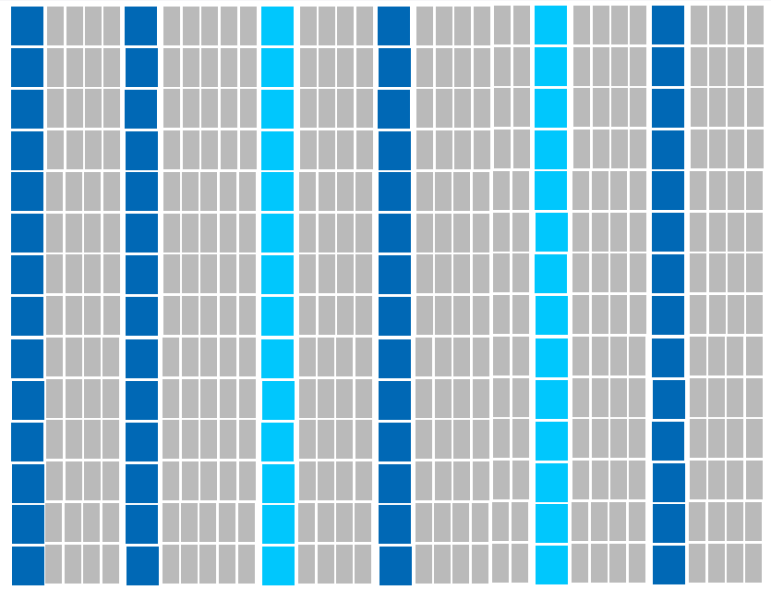


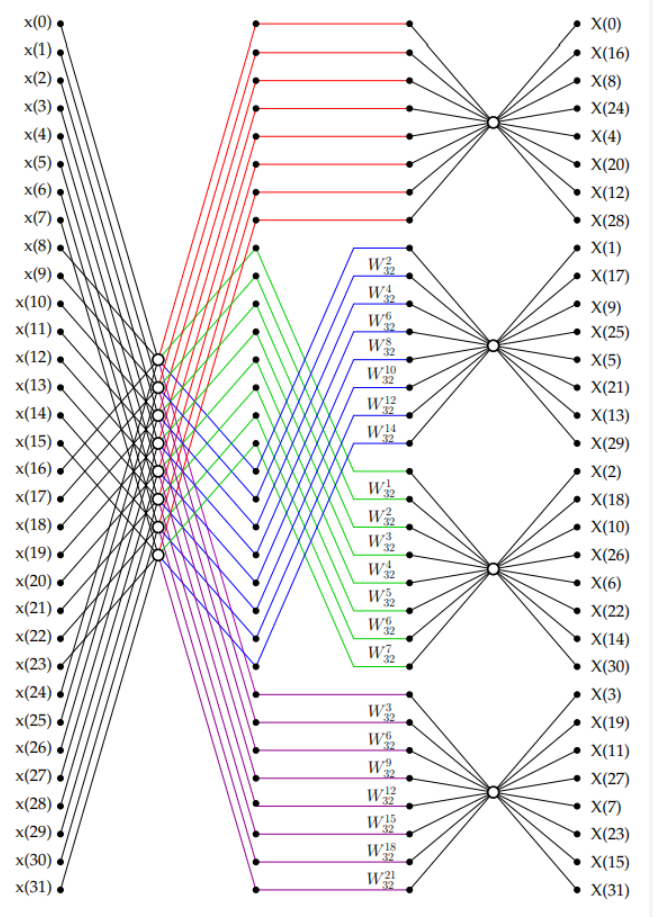
Method - Macro

2D dataflow

1D structure







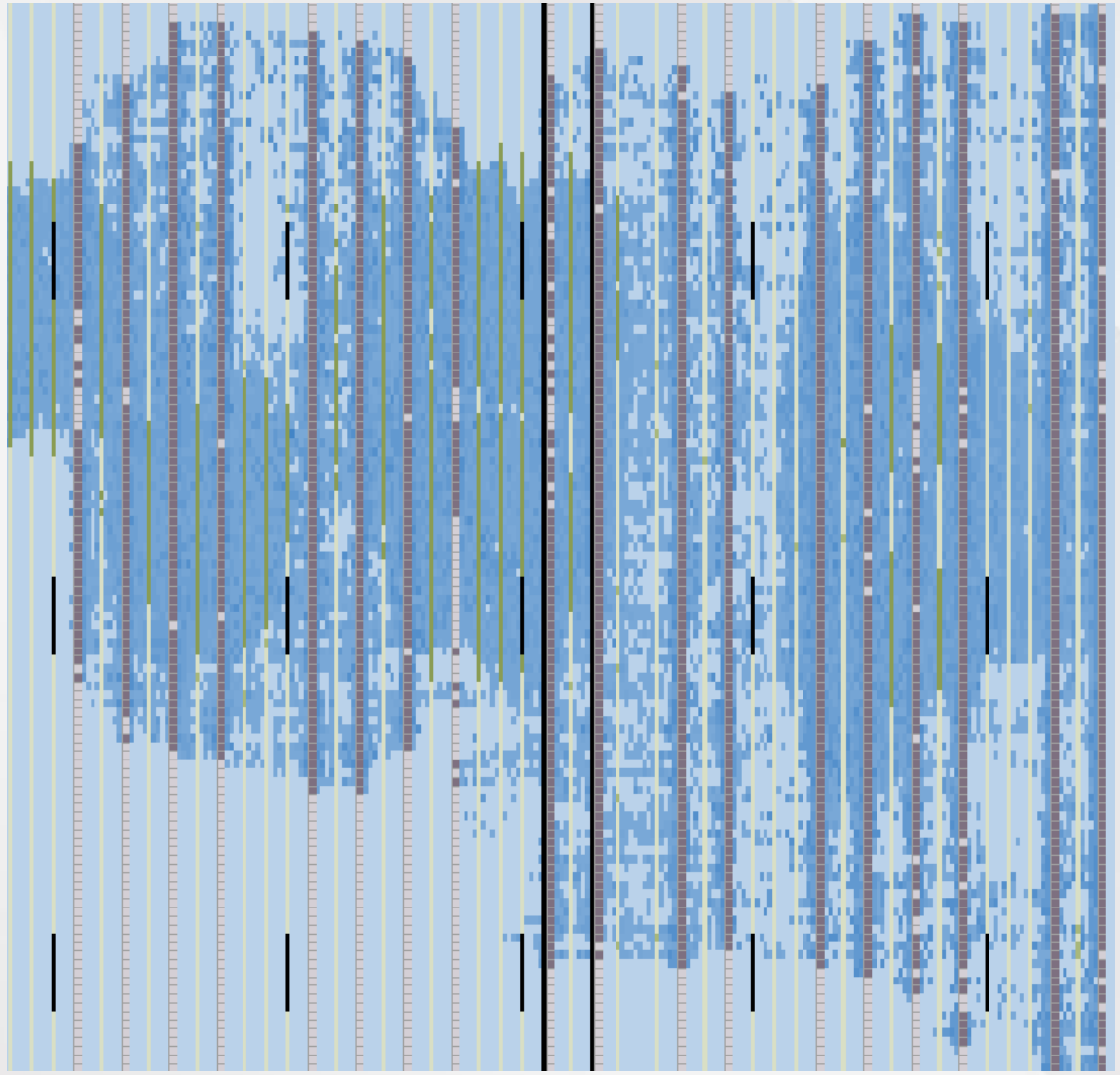
56 DSP Blocks

40 DSP Blocks

436-560 DSP Blocks

Footprint – Not Resources

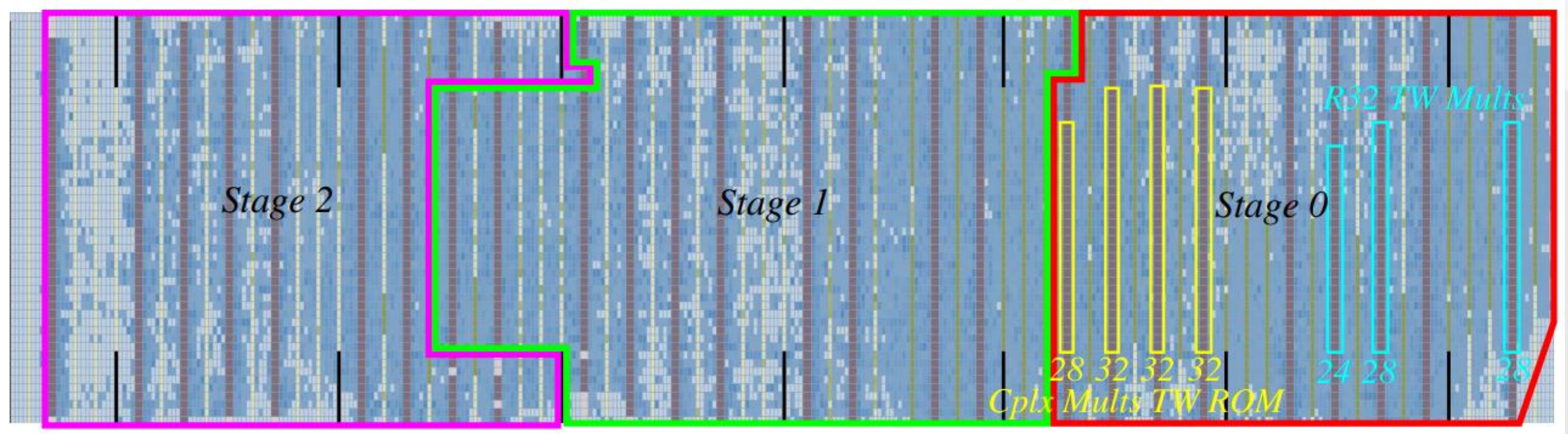
- High Performance FP32 FFT (Radix 32)
- 32 parallel channels – 2048 bit busses
- 771 MHz
- Resources: ~1600 DSPs, ~66K ALMs, ~330 M20Ks
- How big is it? 66K ALMs?
- No – the limiting factor is the DSP Blocks
  - In the AGX*x*027 the ratio is about 100:1 ALM to DSP
  - More accurate to say 160K ALMs!
- Is this an inefficient design?
  - NO!
  - Notice DSPs are packed ~100% density



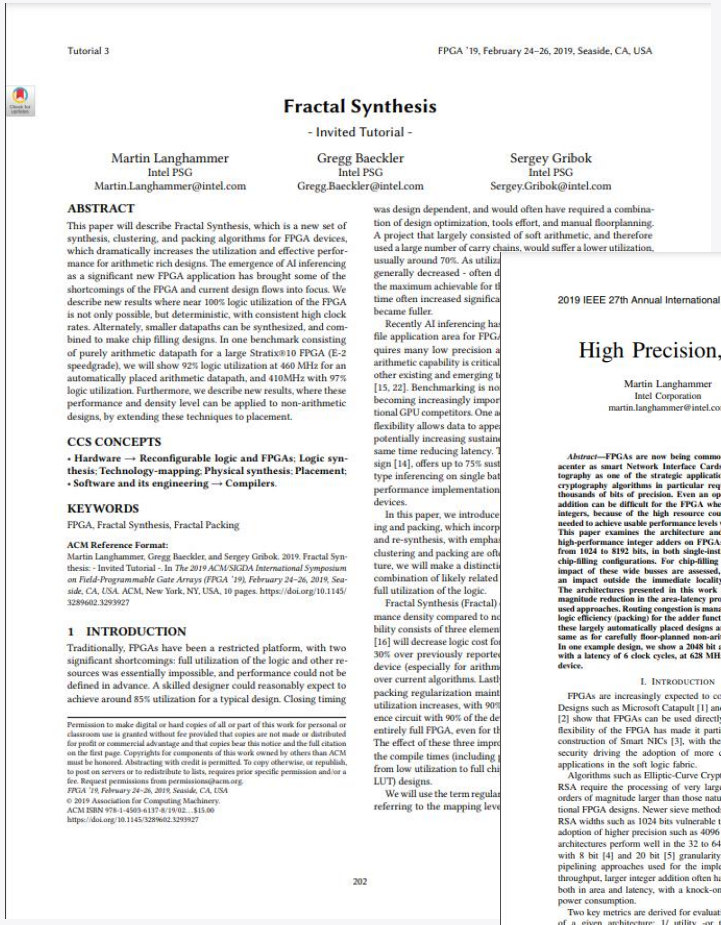
Optimized FFT - ~1300 DSPs

Still 100% packing, can extend this to full chip

And at 771MHz



References (check dblp.org)



Tutorial 3 FPGA '19, February 24–26, 2019, Seaside, CA, USA

Fractal Synthesis

- Invited Tutorial -

Martin Langhammer, Intel PSG, Martin.Langhammer@intel.com
Gregg Baeckler, Intel PSG, Gregg.Baeckler@intel.com
Sergey Gribok, Intel PSG, Sergey.Gribok@intel.com

ABSTRACT

This paper will describe Fractal Synthesis, which is a new set of synthesis, clustering, and packing algorithms for FPGA devices, which dramatically increases the utilization and effective performance for arithmetic rich designs. The emergence of AI inferencing as a significant new FPGA application has brought some of the shortcomings of the FPGA and current design flows into focus. We describe new results where near 100% logic utilization of the FPGA is not only possible, but deterministic, with consistent high clock rates. Alternately, smaller datapaths can be synthesized, and combined to make chip filling designs. In one benchmark consisting of purely arithmetic datapath for a large Stratix®10 FPGA (E-2 speedgrade), we will show 92% logic utilization at 460 MHz for an automatically placed arithmetic datapath, and 410MHz with 97% logic utilization. Furthermore, we describe new results, where these performance and density level can be applied to non-arithmetic designs, by extending these techniques to placement.

CCS CONCEPTS

• **Hardware → Reconfigurable logic and FPGAs**; **Logic synthesis**; **Technology-mapping**; **Physical synthesis**; **Placement**; • **Software and its engineering → Compilers**.

KEYWORDS

FPGA, Fractal Synthesis, Fractal Packing

ACM Reference Format:

Martin Langhammer, Gregg Baeckler, and Sergey Gribok. 2019. Fractal Synthesis: - Invited Tutorial - . In *The 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '19), February 24–26, 2019, Seaside, CA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3289602.3293927

1 INTRODUCTION

Traditionally, FPGAs have been a restricted platform, with two significant shortcomings: full utilization of the logic and other resources was essentially impossible, and performance could not be defined in advance. A skilled designer could reasonably expect to achieve around 85% utilization for a typical design. Closing timing was design dependent, and would often have required a combination of design optimization, tools effort, and manual floorplanning. A project that largely consisted of soft arithmetic, and therefore used a large number of carry chains, would suffer a lower utilization, usually around 70%. As utiliz… generally decreased - often d… the maximum achievable for t… time often increased significa… became fuller.

Recently AI inferencing ha… file application area for FPGA… quires many low precision a… arithmetic capability is critical… other existing and emerging t… [15, 22]. Benchmarking is no… becoming increasingly impor… tional GPU competitors. One a… flexibility allows data to appe… potentially increasing sustain… same time reducing latency. T… sign [14], offers up to 75% sust… type inferencing on single bat… performance implementation… devices.

In this paper, we introduce… ing and packing, which incorp… and re-synthesis, with empha… clustering and packing are oft… ture, we will make a distinctio… combination of likely related … full utilization of the logic.

Fractal Synthesis (Fractal)… mance density compared to n… bility consists of three elemen… [16] will decrease logic cost fo… 30% over previously reporte… device (especially for arithm… over current algorithms. Last… packing regularization maint… utilization increases, with 90%… ence circuit with 90% of the de… entirely full FPGA, even for t… The effect of these three impr… the compile times (including … from low utilization to full ch… LUT) designs.

We will use the term regular… referring to the mapping leve…

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
FPGA '19, February 24–26, 2019, Seaside, CA, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6137-8/19/02…$15.00
https://doi.org/10.1145/3289602.3293927



2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)

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:

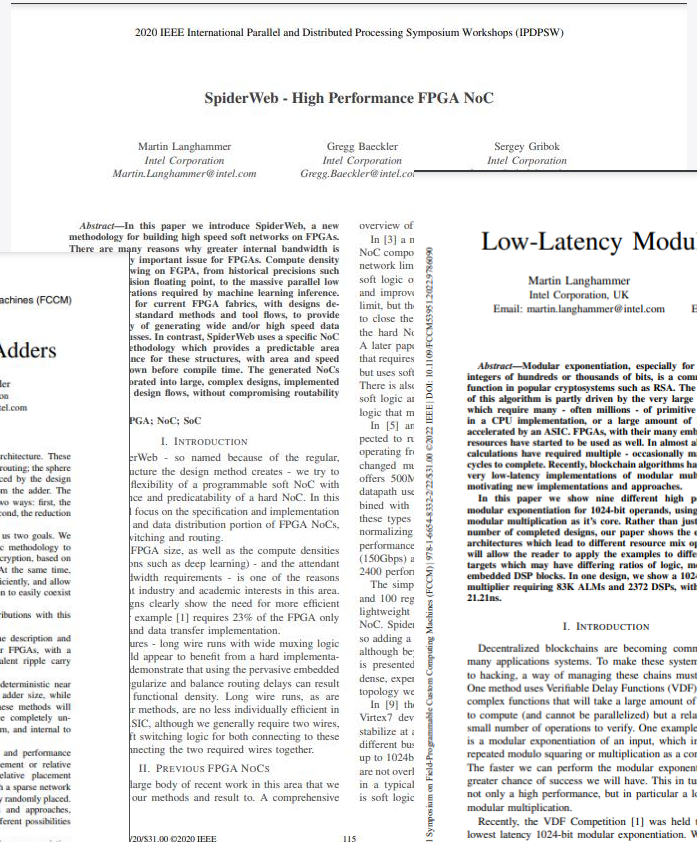
- We develop a way of generalizing the description and construction of very large adders for FPGAs, with a modest area increase over an equivalent ripple carry adder.
- We show a method for achieving a deterministic near 100% logic packing density, for any adder size, while maintaining a very short latency. 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

2576-2621/19/$31.00 ©2019 IEEE
DOI 10.1109/FCCM.2019.00047

Authorized licensed use limited to: Intel Corporation via the Virtual Library. Downloaded on August 09,2024 at 11:24:53 UTC from IEEE Xplore. Restrictions apply.



2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)

SpiderWeb - High Performance FPGA NoC

Martin Langhammer, *Intel Corporation*, *Martin.Langhammer@intel.com*
Gregg Baeckler, *Intel Corporation*, *Gregg.Baeckler@intel.com*
Sergey Gribok, *Intel Corporation*

***Abstract*—In this paper we introduce SpiderWeb, a new methodology for building high speed soft networks on FPGAs. There are many reasons why greater internal bandwidth is … important issue for FPGAs. Compute density …wing on FGPA, from historical precisions such …ision floating point, to the massive parallel low …ations required by machine learning inference. … for current FPGA fabrics, with designs de… standard methods and tool flows, to provide …y of generating wide and/or high speed data …sses. In contrast, SpiderWeb uses a specific NoC …ethodology which provides a predictable area …nce for these structures, with area and speed …own before compile time. The generated NoCs …orated into large, complex designs, implemented … design flows, without compromising routability**

…PGA; NoC; SoC

I. Introduction

…erWeb - so named because of the regular, …acture the design method creates - we try to …flexibility of a programmable soft NoC with …ce and predicatability of a hard NoC. In this … focus on the specification and implementation … and data distribution portion of FPGA NoCs, …vitching and routing.

…FPGA size, as well as the compute densities …ns such as deep learning) - and the attendant …dwidth requirements - is one of the reasons …t industry and academic interests in this area. …gns clearly show the need for more efficient … example [1] requires 23% of the FPGA only …nd data transfer implementation.

…ures - long wire runs with wide muxing logic …ld appear to benefit from a hard implementa… demonstrate that using the pervasive embedded …gularize and balance routing delays can result …functional density. Long wire runs, as are …r methods, are no less individually efficient in …SIC, although we generally require two wires, …ft switching logic for both connecting to these …nnecting the two required wires together.

II. Previous FPGA NoCs

…large body of recent work in this area that we … our methods and result to. A comprehensive

…overview of … In [3] a n… NoC compo… network lim… soft logic o… and improv… limit, but th… to close the… the hard N… A later pape… that requires… but uses soft… There is also… soft logic an… logic that m… In [5] an… pected to r… operating fr… changed m… offers 500M… datapath use… bined with … these types … normalizing … performance … (150Gbps) a… 2400 perfor… The simp… and 100 reg… lightweight … NoC. Spider… so adding a … although be… is presented… dense, exper… topology we… In [9] th… Virtex7 dev… stabilize at … different bu… up to 1024b… are not overl… in a typical … is soft logic…

…20/$31.00 ©2020 IEEE
…SW50202.2020.00025

…d use limited to: Intel Corporation via the Virtual Library. Downloaded on August 0…



2022 IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM) 978-1-6654-8332-2/22/$31.00 ©2022 IEEE DOI: 10.1109/FCCM53951.2022.9786090

Low-Latency Modular Exponentiation for FPGAs

Martin Langhammer, Intel Corporation, UK, Email: martin.langhammer@intel.com
Sergey Gribok, Intel Corporation, USA, Email: sergey.gribok@intel.com
Bogdan Pasca, Intel Corporation, France, Email: bogdan.pasca@intel.com

***Abstract*—Modular exponentiation, especially for very large integers of hundreds or thousands of bits, is a commonly used function in popular cryptosystems such as RSA. The complexity of this algorithm is partly driven by the very large word sizes, which require many - often millions - of primitive operations in a CPU implementation, or a large amount of logic when accelerated by an ASIC. FPGAs, with their many embedded DSP resources have started to be used as well. In almost all cases, the calculations have required multiple - occasionally many - clock cycles to complete. Recently, blockchain algorithms have required very low-latency implementations of modular multiplications, motivating new implementations and approaches.**

In this paper we show nine different high performance modular exponentiation for 1024-bit operands, using a 1024-bit modular multiplication as it's core. Rather than just showing a number of completed designs, our paper shows the evolution of architectures which lead to different resource mix options. This will allow the reader to apply the examples to different FPGA targets which may have differing ratios of logic, memory, and embedded DSP blocks. In one design, we show a 1024b modular multiplier requiring 83K ALMs and 2372 DSPs, with a delay of 21.21ns.

I. Introduction

Decentralized blockchains are becoming common across many applications systems. To make these systems resistant to hacking, a way of managing these chains must be found. One method uses Verifiable Delay Functions (VDF). These are complex functions that will take a large amount of operations to compute (and cannot be parallelized) but a relatively very small number of operations to verify. One example of a VDF is a modular exponentiation of an input, which in turn uses repeated modulo squaring or multiplication as a core function. The faster we can perform the modular exponentiation, the greater chance of success we will have. This in turn requires not only a high performance, but in particular a low-latency, modular multiplication.

Recently, the VDF Competition [1] was held to find the lowest latency 1024-bit modular exponentiation. We will use the required kernel of this function, a 1024-bit modular multiplication (which we then develop into the more efficient squaring), as a reference point for our work.

We will start with the definition of a generic modular multiplication (`modmul`):

$$Q = AB \bmod N.$$

Followed by the modular exponentiation algorithm (`modexp`), which is comprised of multiple modular multiplications:

$$R = C^E \bmod N.$$

The `modexp` Right-to-Left algorithm iterates over the bits of the exponent E; when the scanned bit is '1' two `modmul`s are performed, as opposed to a single one when the scanned bit is '0'. The base C is therefore squared on the first iteration, and the output of each `modmul` is then squared on each successive `modmul`.

A number of FPGA `modmul` implementations have been reported, with the majority of these using either the methods of Montgomery [2] or Barrett [3]. Most of the previously reported implementations used multi-cycle multipliers, with each component multiplier requiring several execution cycles to calculate each product.

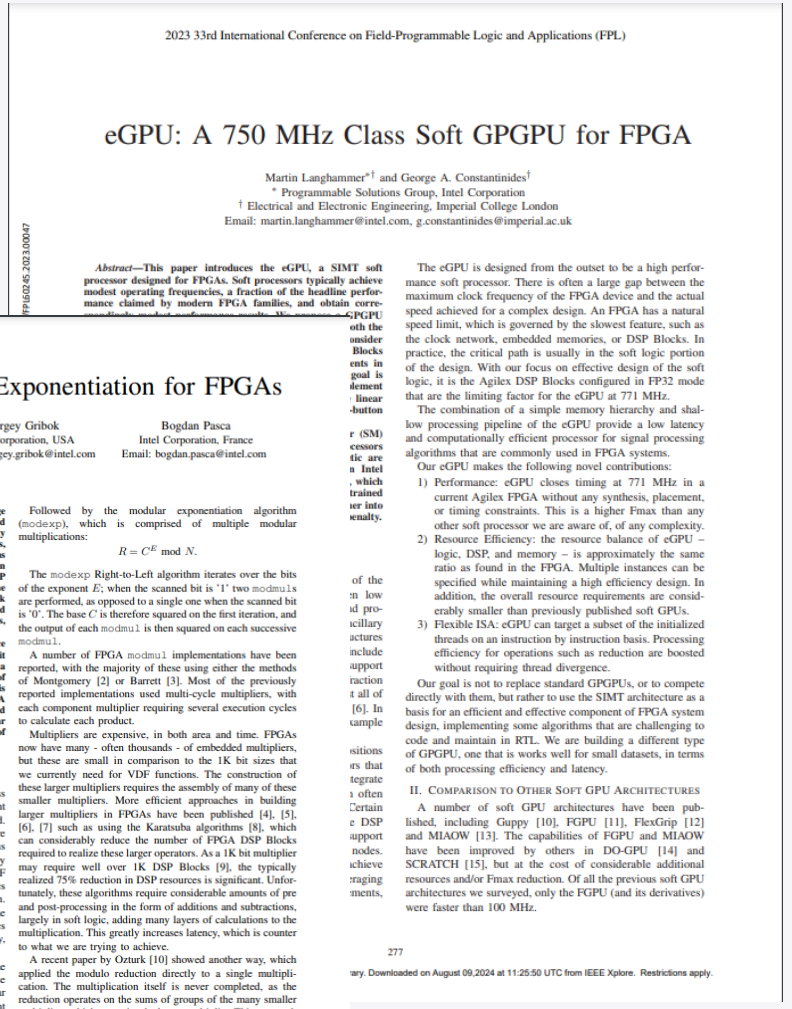
Multipliers are expensive, in both area and time. FPGAs now have many - often thousands - of embedded multipliers, but these are small in comparison to the 1K bit sizes that we currently need for VDF functions. The construction of these larger multipliers requires the assembly of many of these smaller multipliers. More efficient approaches in building larger multipliers in FPGAs have been published [4], [5], [6], [7] such as using the Karatsuba algorithms [8], which can considerably reduce the number of FPGA DSP Blocks required to realize these larger operators. As a 1K bit multiplier may require well over 1K DSP Blocks [9], the typically realized 75% reduction in DSP resources is significant. Unfortunately, these algorithms require considerable amounts of pre and post-processing in the form of additions and subtractions, largely in soft logic, adding many layers of calculations to the multiplication. This greatly increases latency, which is counter to what we are trying to achieve.

A recent paper by Ozturk [10] showed another way, which applied the modulo reduction directly to a single multiplication. The multiplication itself is never completed, as the reduction operates on the sums of groups of the many smaller multipliers which comprise the larger multiplier. This approach was the basis of the winning entry to the VDF competition.

We will start with a background of multiplier construction, along with a review of Ozturk's method. We will show a combination of nine different algorithms and optimizations, with a progressively lower latency. These will have differing resource mixes. Our results should guide the reader into choosing the correct approach for their application requirement

978-1-6654-8332-2/22/$31.00 ©2022 IEEE

Authorized licensed use limited to: Intel Corporation via the Virtual Library. Downloaded on August 09,2024 at 11:26:40 UTC from IEEE Xplore. Restrictions apply.



2023 33rd International Conference on Field-Programmable Logic and Applications (FPL)

eGPU: A 750 MHz Class Soft GPGPU for FPGA

Martin Langhammer*† and George A. Constantinides†
* Programmable Solutions Group, Intel Corporation
† Electrical and Electronic Engineering, Imperial College London
Email: martin.langhammer@intel.com, g.constantinides@imperial.ac.uk

***Abstract*—This paper introduces the eGPU, a SIMT soft processor designed for FPGAs. Soft processors typically achieve modest operating frequencies, a fraction of the headline performance claimed by modern FPGA families, and obtain corre-**

FPL60245.2023.00047

The eGPU is designed from the outset to be a high performance soft processor. There is often a large gap between the maximum clock frequency of the FPGA device and the actual speed achieved for a complex design. An FPGA has a natural speed limit, which is governed by the slowest feature, such as the clock network, embedded memories, or DSP Blocks. In practice, the critical path is usually in the soft logic portion of the design. With our focus on effective design of the soft logic, it is the Agilex DSP Blocks configured in FP32 mode that are the limiting factor for the eGPU at 771 MHz.

The combination of a simple memory hierarchy and shallow processing pipeline of the eGPU provide a low latency and computationally efficient processor for signal processing algorithms that are commonly used in FPGA systems.

Our eGPU makes the following novel contributions:

1) Performance: eGPU closes timing at 771 MHz in a current Agilex FPGA without any synthesis, placement, or timing constraints. This is a higher Fmax than any other soft processor we are aware of, of any complexity.
2) Resource Efficiency: the resource balance of eGPU – logic, DSP, and memory – is approximately the same ratio as found in the FPGA. Multiple instances can be specified while maintaining a high efficiency design. In addition, the overall resource requirements are considerably smaller than previously published soft GPUs.
3) Flexible ISA: eGPU can target a subset of the initialized threads on an instruction by instruction basis. Processing efficiency for operations such as reduction are boosted without requiring thread divergence.

Our goal is not to replace standard GPGPUs, or to compete directly with them, but rather to use the SIMT architecture as a basis for an efficient and effective component of FPGA system design, implementing some algorithms that are challenging to code and maintain in RTL. We are building a different type of GPGPU, one that is works well for small datasets, in terms of both processing efficiency and latency.

II. Comparison to Other Soft GPU Architectures

A number of soft GPU architectures have been published, including Guppy [10], FGPU [11], FlexGrip [12] and MIAOW [13]. The capabilities of FGPU and MIAOW have been improved by others in DO-GPU [14] and SCRATCH [15], but at the cost of considerable additional resources and/or Fmax reduction. Of all the previous soft GPU architectures we surveyed, only the FGPU (and its derivatives) were faster than 100 MHz.

…ary. Downloaded on August 09,2024 at 11:25:50 UTC from IEEE Xplore. Restrictions apply.

Conclusions

Mindset

- (almost) 100% is possible
- But more complex than a simple benchmarking
- Not only reaching ~100% of the LUTs, but using ~100 % of the LUTs you reach
- And fully utilizing the routing

Method

- Planning a FPGA design – properly – is not trivial
  - Micro – full use at LUT level
  - Midi – how does this map to a local region?
  - Macro - how do your blocks map to full chip?



Accelerating Innovators.