# The AI Evolution Calls for Adaptable Inferencing Platforms (WP023)

Achronix
Data Acceleration

**December 02, 2020** **White Paper**

## Growth in Model Sizes Challenges Existing Architectures

Deep learning's demand for computing power is growing at an incredible rate, accelerating recently from doubling every year to doubling every three months. Increasing the capacity of deep neural network (DNN) models has shown improvements across a wide range of areas ranging from natural language processing to image processing — critical technology for real-time applications such as autonomous driving and robotics. Research at Facebook, for example, has shown a linear increase in accuracy versus model size, boosted even further by training on larger datasets.

At the leading edge, model size is currently growing far faster than Moore's Law with trillion-parameter models for some applications now being considered. Though few production systems will go to the same extremes, the effect of parameter count on performance in these examples will have a knock-on effect on real-world applications. This increase in model size presents a challenge for implementers. Without the ability to rely solely on the silicon-scaling roadmap, other solutions are needed to keep pace with the demand for increased model capacity at a cost compatible with volume deployment. This growth calls for the adoption of customized architectures that squeeze the greatest amount of performance out of each transistor available.

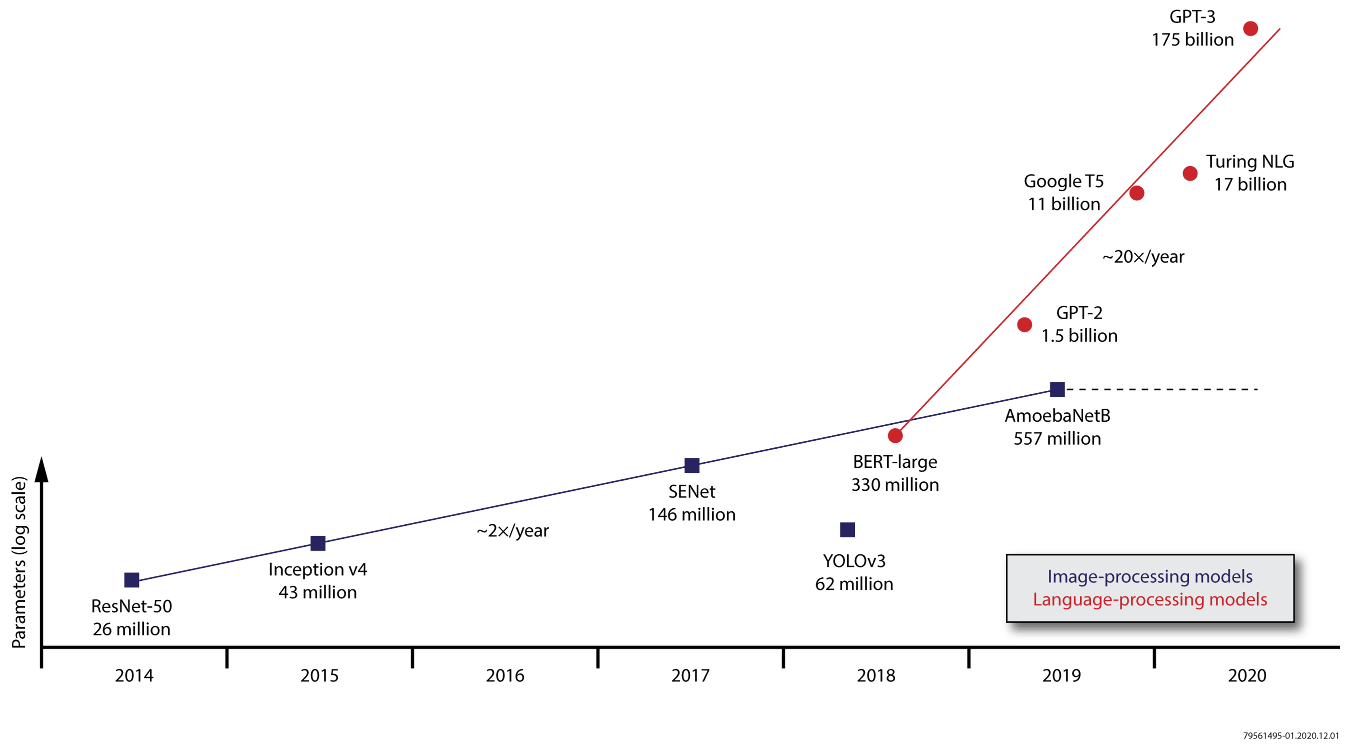

**Figure 1:** ***Growth in Model Size (source: Linley Group)***

As well as growing rapidly in terms of parameter count, deep-learning architectures are evolving quickly. While DNNs continue to make extensive use of a mixture of traditional convolution, fully connected, and pooling layers, other structures have emerged, such as the self-attention networks in NLP. They still demand high-speed matrix- and tensor-oriented arithmetic but the changes in memory-access patterns can be troublesome for GPUs and the current generation of off-the-shelf accelerators.

The changes in structure mean commonly cited metrics such as Tera-operations per second (TOps) have decreasing relevance. Very often, the processing engines cannot achieve their peak TOps score because the memory and data-movement infrastructure cannot provide enough throughput without changes to the way models are processed. For example, batching input samples is a common approach as it generally improves the usable parallelism on many architectures. However, batching increases the latency of response, which will often be unacceptable in real-time inferencing applications.

## Numeric Flexibility is One Path to Throughput

One avenue for improving inferencing performance that remains compatible with rapid structural evolution is to adapt the numeric resolution of the calculations to the needs of individual layers. Generally, during inferencing many deep-learning models can tolerate a significant loss in precision and increased quantization error compared to that needed for training, which is normally performed with standard or double-precision floating-point arithmetic. Such formats offer the ability to support high precision over an extremely wide dynamic range. This feature is important in training because the back-propagation algorithm common to training requires the ability to make subtle changes across many weights on each pass in order to guarantee convergence.

Conventionally, floating-point arithmetic requires a large amount of hardware support to enable the low-latency processing of high-resolution data types. Originally developed to support scientific applications on high-performance computers, the overhead needed to support it fully was not a major concern.

Many inferencing deployments convert the model to use fixed-point arithmetic operating with significantly reduced precision. In these cases, the impact on accuracy is generally minimal. Indeed, some layers can be converted to use extremely restricted numeric ranges, with even binary or ternary values becoming viable choices.

However, integer arithmetic is not always an effective solution. There are filters and layers that call for high dynamic ranges. To accommodate this requirement, integer hardware may need to process the data as 24- or 32-bit words, which will consume more resources than the 8- or 16-bit integer data types that are readily supported in typical single-instruction multiple-data (SIMD) accelerators.

One compromise is to use a narrow floating-point format, such as one that fits into a 16-bit word. This choice allows for greater parallelism but it does not overcome a performance obstacle inherent in most floating-point data types. The issue is that after each calculation, the two parts of the floating-point format need to be adjusted as the most significant bit of the mantissa is not stored explicitly. The magnitude of the exponent, therefore, needs to be adjusted through a series of logical shift operations to ensure that the implied leading '1' is always present. This normalization operations has the benefit that there is only one representation for any single numeric value, which is important for software compatibility in user applications. However, for many signal processing and AI inference routines, it is unnecessary.

Much of the hardware overhead of these operations can be avoided by moving away from the need to normalize the mantissa and adjust the exponent after each and every calculation. This is the approach taken with block floating-point arithmetic, a data format that has been used on standard fixed-point DSPs to improve their performance on algorithms in audio processing for mobile devices, DSL modems, and radar systems.

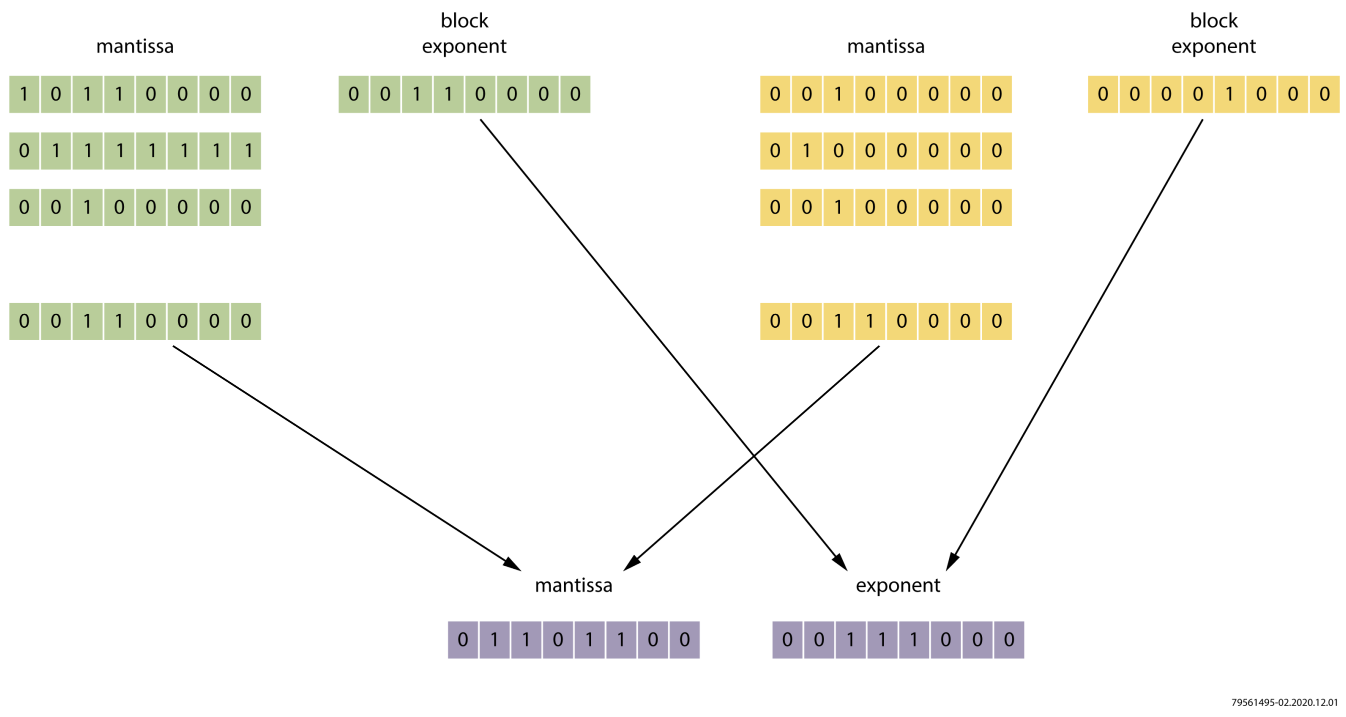

**Figure 2:** ***Example of Block Floating-Point Computation***

With block floating-point arithmetic, there is no requirement to left-align the mantissa. The data elements for a series of calculations can share the same exponent, a change that simplifies the design of the execution pipeline. There is minimal loss of accuracy caused by rounding for values that occupy a similar dynamic range. The choice of suitable range for each block of calculations is made at design time. After the block of calculations is complete, an exit function can round and normalize the values so that they can be used as conventional regular floating-point values if required.

Support for block floating-point formats is one of the functions of the machine learning processor (MLP). This highly flexible arithmetic logic unit is available in the Achronix Speedster®7t and Speedcore™ eFPGA architectures. The MLP is optimized for the dot-product and similar matrix operations needed in AI. These MLPs support for block floating-point provides substantial improvements over conventional floating point. Throughput for 16-bit block floating point is eight times higher than that of conventional half-precision floating point, making it as fast as 8-bit integer arithmetic with only a 15% increase in active power consumption compared to operations performed purely in integer form.

Another potentially important data type is the TensorFloat 32 (TF32) format, which has reduced precision compared to standard-precision formats but keeps the high dynamic range. TF32 also lacks the optimized throughput of blocked exponent handling but is useful for applications where the ease of porting models created using TensorFlow and similar environments is important. The highly flexible nature of the MLP in Speedster7t FPGAs makes it possible to handle TF32 arithmetic using the 24-bit floating-point mode. Moreover, the highly configurable nature of the MLP means that a new, block-floating point version of TF32 could be supported where four samples share the same exponent. Block-float TF32 is supported by the MLP at double the density of conventional TF32.

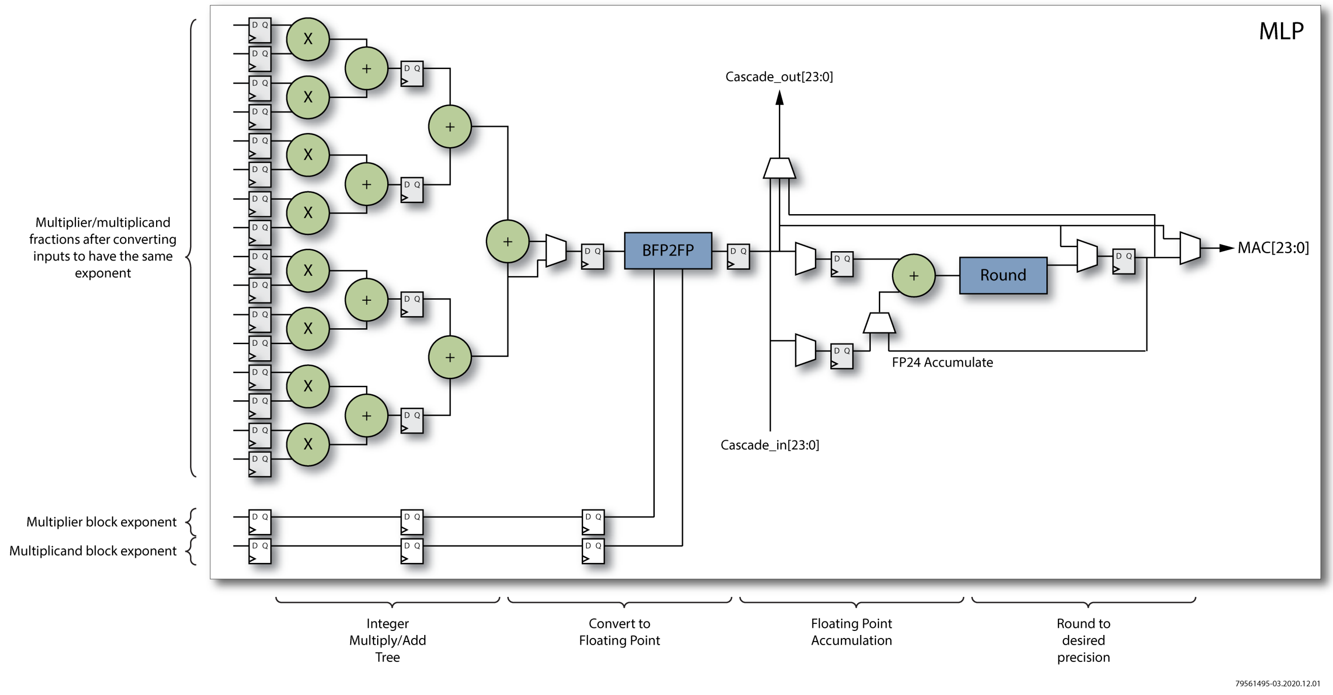
MLP
Multiplier/multiplicand fractions after converting inputs to have the same exponent
Multiplier block exponent
Multiplicand block exponent
D Q
X
+
BFP2FP
Cascade_out[23:0]
Cascade_in[23:0]
Round
FP24 Accumulate
MAC[23:0]
Integer Multiply/Add Tree
Convert to Floating Point
Floating Point Accumulation
Round to desired precision
79561495-03.2020.12.01

**Figure 3: *MLP Structure***

## Flexibility of Processing Optimizes Arithmetic Support

Though the MLPs support for multiple data types is vital to inferencing applications, its power is only unleashed by being part of an FPGA architecture. The ability to easily define different interconnect structures sets the FPGA apart from most architectures. The ability to define both interconnect and arithmetic logic together within the FPGA eases the path to building a balanced architecture. Not only does the designer have the ability to build direct support for custom data types, they can also define the most appropriate interconnect structure to pipeline data in and out of the processing engines. Reprogrammability provides the further ability to respond to the rapid evolution of AI. Changes in the dataflow of custom layers can be readily supported by modifying the FPGA's logic.

A major advantage of the FPGA is the ease with which functions can be switched between optimized embedded compute engines and the programmable logic implemented by lookup-table elements. Some functions map well onto the embedded compute engines such as the Speedster7t MLP. For example, higher-precision arithmetic is best allocated to the MLP because the increased bit width leads to exponential growth in the size of functional units that implement functions such as high-speed multiplication.

Lower-precision integer arithmetic can often be allocated efficiently to the lookup tables (LUTs) that are common to FPGA architectures. Designers have the option to implement high-latency, high-parallelism arrays using simple bit-serial multiplier circuits. Or they can dedicate more logic per function by building structures such as carry-save and carry-lookahead adders, circuits that are commonly used to implement low-latency multipliers. Support for high-speed arithmetic is enhanced through unique LUT configuration in Speedster7t devices where the LUT provides a highly efficient mechanism for implementing Booth encoding, an area-saving approach to multiplication.

The result is a halving of the number of LUTs needed to implement integer multipliers for a given bit width. As issues such as privacy and security in machine learning become more important, the response may be to deploy forms of homomorphic encryption in the models. These protocols often involve modulo and bit-field operations that are highly suited to LUT implementations, helping to cement the FPGA's positions as a futureproof technology for AI.

## Data Movement Provides the Key to Throughput

To make the most of the numeric customization in a machine-learning context, the surrounding architecture is just as important. The ability to transfer data where and when it is needed in increasingly irregular graph representations is a key advantage of programmable hardware. But not all FPGA architectures are created equal.

One issue with conventional FPGA architectures is that they have evolved from earlier applications where their primary function was to implement interface and control-circuit logic. Over time, because the devices provided a way for cellular basestation manufacturers to migrate from increasingly expensive ASICs, the FPGA architectures incorporated DSP blocks to handle filtering and channel estimation functions. In principle, these DSP blocks can handle AI functions. But the blocks were originally designed primarily to handle 1D finite impulse-response (FIR) filters that stream data through the processing elements using a relatively simple pipeline in which a series of fixed coefficients are applied to a continuous stream of samples.

Convolutional layers are relatively simple to support with traditional processor architectures. Others are far more problematic. Fully connected layers, for example, require the output of each neuron in one layer to be applied to all of the neurons in the next layer. The result is that the dataflow between arithmetic logic units is far more complex than for traditional DSP applications and puts far more pressure on the interconnect when throughput is at a premium.

Though a processing engine such as a DSP core may be able generate a result every cycle, the routing constraints within the FPGA may be such that it is not possible to deliver the data to it quickly enough. Typically the 1D FIR filters common to communications systems for which many traditional FPGAs were designed, congestion is not an issue. The result of each filter stage can easily be pipelined to the next. But the far higher interconnect required in tensor operations and lower data locality of machine-learning applications makes the interconnect far more important for any implementation.

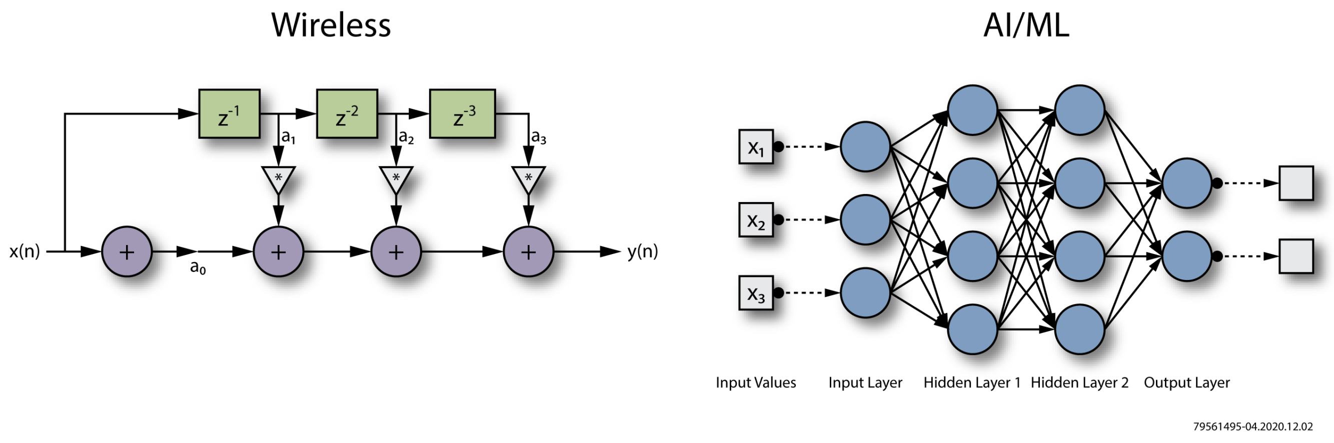

**Figure 4:** ***Filter and AI Dataflows***

The issue of data locality in machine learning requires attention to interconnect design at multiple levels. Due to the sheer number of parameters in the most effective models, off-chip data storage is often a necessity. The key requirement is for mechanisms that can deliver data when needed with low latency and for the use of efficient scratchpad memories close to the processing engines to make the most effective use of prefetch and other strategies that use predictable access patterns to ensure data is available at the right times.

In the Speedster7t architecture, there are three innovations used for data movement:

- An optimized memory hierarchy
- Efficient local routing technologies
- A high-speed 2D network-on-chip (NoC) for cross-chip and off-chip data movement

Conventional FPGAs typically have on-chip RAM blocks distributed across the fabric that are placed at some distance from the processing engines. This choice was an effective architecture for typical FPGA designs but it imposes additional and unnecessary routing overhead in an AI context. In the Speedster7t architecture, each MLP is associated with a 72-kb dual-port block RAM (BRAM72k) and a smaller 2-kb dual-port logic RAM (LRAM2k) that can act as a tightly coupled register file.

The MLP and its associated memories can be accessed separately via FPGA routing resources; however, if a memory is driving the associated MLP it can use a direct connection, offloading the FPGA routing resources and providing a high-bandwidth connection.

Employed in AI applications, the BRAM can act as a memory for values that are not expected to change on each cycle, such as neuron weights and activation values. The LRAM is more suited to storing temporary values with only short-term data locality, such as a short pipeline of input samples and for accumulated values for tensor contractions and pooling activities.

The architecture takes into account the need to be able to partition large complex layers into segments that can operate in parallel and feed temporary data values to each. Both the BRAM and LRAM have cascade connections that readily support the construction of systolic arrays commonly used in machine-learning accelerators.

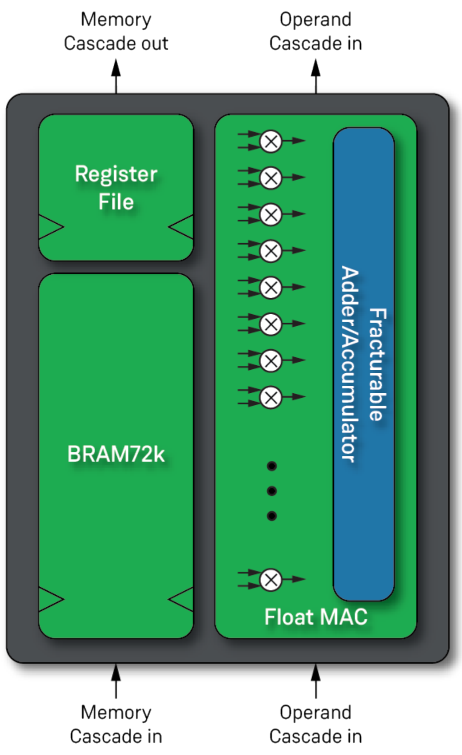

**Figure 5:** ***MLP with Memory and Cascade Connections***

The MLP can be driven from the logic fabric, the cascade path for shared data, as well as from the associated BRAM72k and LRAM2k on a cycle-by-cycle basis. This arrangement enables the construction of complex scheduling mechanisms and data processing pipelines that keep the MLP fed with data while supporting the widest possible range of connectivity patterns between neurons. Keeping the MLP fed with data is key in increasing the effective TOps rate.

The output of the MLP has the same flexibility, providing the ability to create systolic arrays and more complex routing topologies that provide an optimized fabric for each type of layer that might be needed in a deep-learning model.

The 2D NoC in the Speedster7t architecture provides high-bandwidth connections from the programmable logic of the fabric to the high-speed intreface subsystems located in the I/O ring for connecting to off-chip resources. They include GDDR6 for high-speed memory accesses and inter-chip interconnect protocols such as PCIe Gen5 and 400G Ethernet. This structure supports the construction of highly parallelized architectures as well as highly data-optimized accelerators based around a central FPGA.

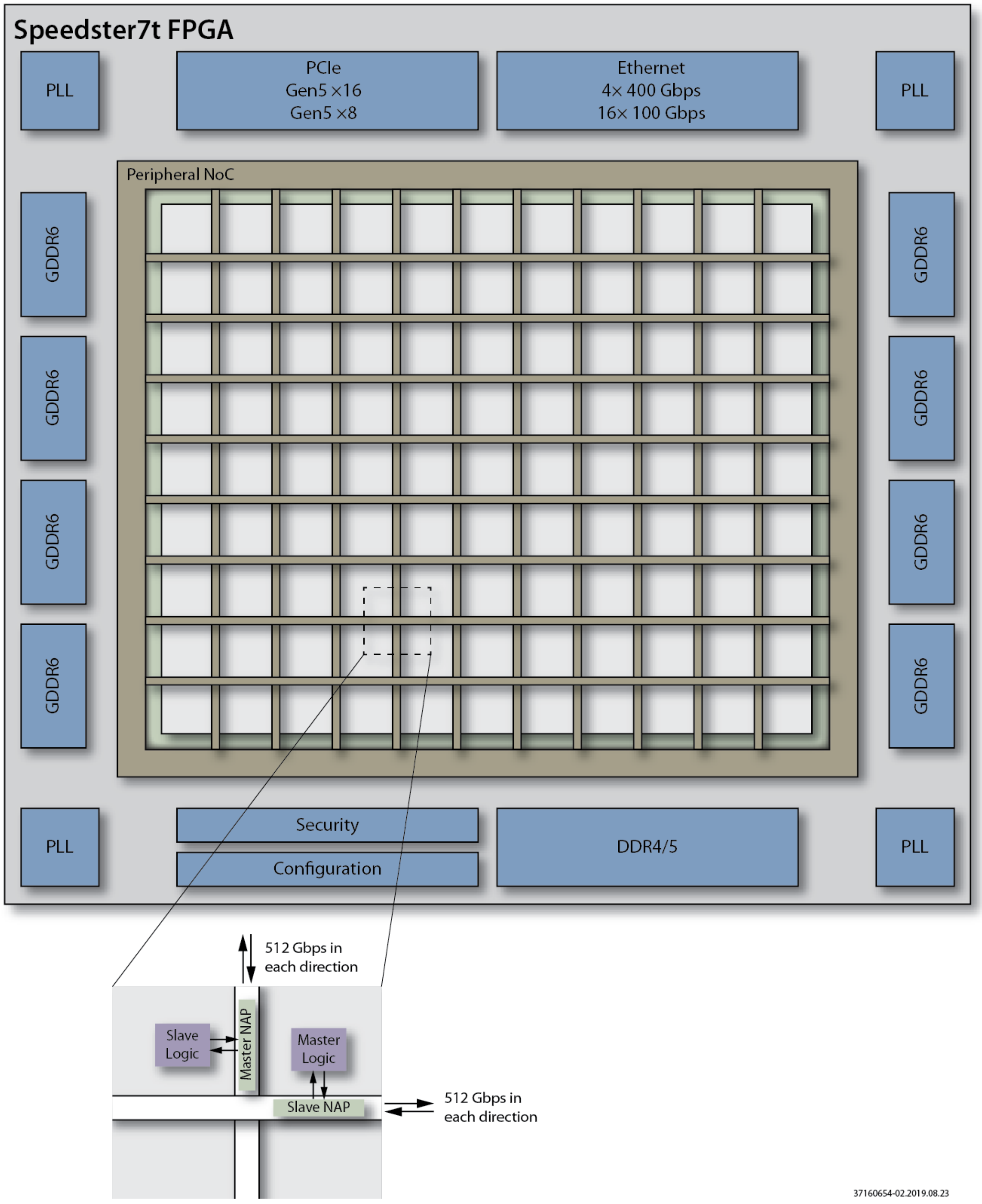

**Figure 6:** ***NoC with Endpoints and I/O Blocks***

The 2D NoC makes it possible to massively increase the available bandwidth on an FPGA by bringing high-density packet routing to hundreds of access points distributed across the fabric. Conventional FPGAs would have to dedicate thousands of individually programmed routing paths to achieve the same throughput and in doing so, would starve local resources of interconnect. By bringing gigabit data transfers to local regions through the network access points, the 2D NoC alleviates the routing problem while supporting the ability to stream data in and out of MLPs and LUT-based custom processors easily and quickly.

The associated resource saving can be considerable — a 2D NoC implemented in traditional FPGA soft logic with 64 NoC access points (NAP), each providing a 128-bit interface running at 400 MHz would consume 390 kLUTs. By comparison, the hard 2D NoC in the Speedster 7t1500 device, which has 80 NAPs, consumes no FPGA soft logic and provides far higher bandwidth.

There are several additional advantages in the use of a 2D NoC. The logic design is easier to floorplan because of the lower interconnect congestion between adjacent regions. Designs are also more regular as there is no need to allocate resources from neighboring regions in order to implement the control logic for the high-bandwidth paths. A further benefit is that partial reconfiguration is greatly simplified — the NAPs allow individual regions to be effectively self-contained elements that can be swapped in and out as the application demands. This approach to reconfigurability in turn supports architectures where there is a need to use different models at certain times or to support on-chip fine-tuning or retraining of the model at periodic intervals.

## Conclusion

As models grow and become more structurally complex, the FPGA is becoming an increasingly attractive foundation for building effective, low-latency AI inferencing solutions thanks to its support for a huge variety of numeric data types and data-steering capability. However, it is not enough to take a conventional FPGA and apply it to machine learning. The data-centric nature of machine learning calls for a balanced architecture that ensures performance is not constrained artificially. By taking into account the characteristics of machine learning, not just now, but its future evolution, the Achronix Speedster7t FPGA provides the ideal foundation for AI inferencing.

Achronix®

Data Acceleration

Achronix Semiconductor Corporation

2903 Bunker Hill Lane
Santa Clara, CA 95054
USA

Website: www.achronix.com
E-mail : info@achronix.com

## Notice of Disclaimer