The Ideal Solution for AI Applications — Speedcore eFPGAs

achronix
SEMICONDUCTOR CORPORATION

December 22, 2017 **WP011**

Introduction and Background

Artificial intelligence (AI) is reshaping the way the world works, opening up countless opportunities in commercial and industrial systems. Applications span diverse markets such as autonomous driving, medical diagnostics, home appliances, industrial automation, adaptive websites and financial analytics. Even the communications infrastructure linking these systems together is moving towards automated self-repair and optimization. These new architectures will perform functions such as load balancing and allocating resources such as wireless channels and network ports based on predictions learned from experience.

These applications demand high performance and, in many cases, low latency to respond successfully to real-time changes in conditions and demands. They also require power consumption to be as low as possible, rendering unusable, solutions that underpin machine-learning in cloud servers where power and cooling are plentiful. A further requirement is for these embedded systems to be always on and ready to respond even in the absence of a network connection to the cloud. This combination of factors calls for a change in the way that hardware is designed.

Requirements for Machine Learning

Many algorithms can be used for machine learning, but the most successful ones today are deep neural networks — technology that takes advantage of the high computing performance and memory bandwidth available in today's silicon. Inspired by biological processes and structures, a deep neural network can employ ten or more layers in a feed-forward arrangement. Each layer uses virtual neurons that perform a weighted sum on a set of inputs, and then passing the result to one or more neurons in the next layer.

In the classical neural networks of the 1980s and 1990s, the neurons were generally fully connected. Each neuron in a layer processed data from all of the neurons in the layer above. Although these structures were capable of classifying objects in images, they proved impractical for data-intensive applications such as image, video and audio processing.

The convolutional layer employs far fewer connections per neuron and one that echoes the organization of neurons in the visual cortex of the organic brain. Each neuron in a convolutional layer processes data for only a subset of the entire image, reducing the computational overhead and demands on memory bandwidth compared to fully connected layers.

Pooling layers combine the outputs from multiple neurons, producing a single output. A max-pooling layer, for example, takes the maximum value from the inputs and passes the result to the next stage of the deep neural network. Such layers are useful for reducing the dimensionality of the data. They are often used in combination with convolutional layers to refine the data in stages and move from pixel- or sample-level representations to object classifications. Typical of this deep-learning structure is the AlexNet entry to the ImageNet LSCVRC-2010 contest, which employs five convolution layers, three fully connected layers, and three max-pooling stages.

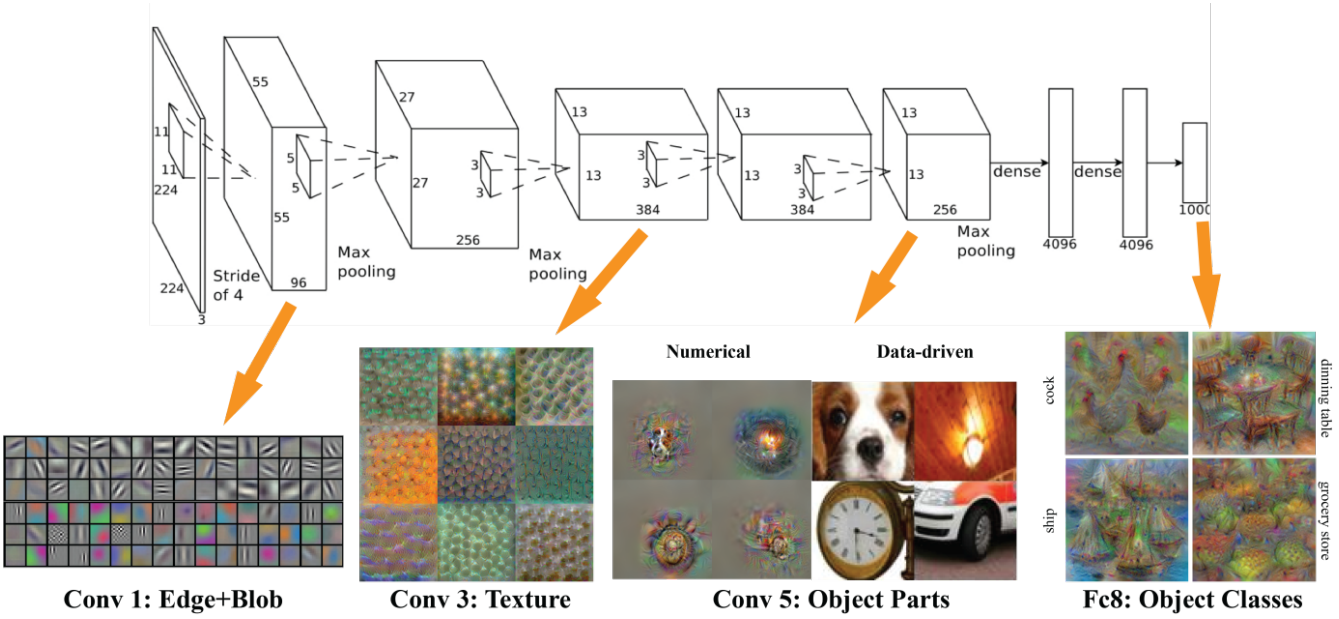

Figure 1: ***Deep-Learning Network using Convolutional, Max-Pooling and Fully Connected Layers to Interpret Images from the ImageNet Database***
(Source: MIT – *http://vision03.csail.mit.edu/cnn_art/index.html****)***

Although there is a common-core approach to constructing most deep neural networks, research into deep learning has pushed the technology in many directions. There is no one-size-fits-all architecture for deep learning. For example, the internal structure of a deep-learning system for recognizing and reacting to human dialog is quite different to one employed to find road signs in camera images. Increasingly, deep-learning applications are incorporating elements that are not based on simulated neurons. Instead, they are making use of structures that move beyond the relatively simple feed-forward network exemplified by AlexNet.

Recurrent neural networks, which employ feedback loops to steer outputs back into the input stream to provide contextual information, are now commonly used in voice recognition and processing. The memory network provides another variant of deep learning that is beginning to yield high-quality results. As its name suggests, the memory network adds local, temporary storage to the core neural network to hold short-term information about recent dialogs. The neural network relies on this context storage to help determine the best response to the user.

Other types of neural network include adversarial architectures that use two linked networks. Competition between the two is used to produce better answers in situations where there is a risk of a single network becoming stuck. Cellular networks, self-organizing maps, and support vector machines provide other avenues for developing systems that can learn. As the technology continues to develop, other novel architectures will emerge. Much like the organic brain itself, plasticity is a major requirement for any organization that aims to build machine learning into its product designs.

One important contrast between the organic brain and AI is the ability to separate activities such as training and the inferencing stage when the trained network is called upon to make decisions. The key breakthrough that led to the explosion in deep learning over the past decade came in the mid-2000s, when efficient techniques were discovered to allow the training of multiple layers at once. Although the techniques are relatively efficient, they rely on enormous compute power generally supplied by servers that use many GPUs or processors for the task. The training process is performed in the background (often on the cloud) and does not require a result to be produced in real time. A further advantage of performing training on cloud servers is that it allows data to be combined from numerous systems in the field, providing a much richer source of information for the training process.

For inferencing, the compute demand is lower than with training but is generally called upon to provide a real-time response in most real-world applications. Energy-efficient parallel processing is a key requirement for inferencing systems because many will not have a permanent external power source.

Some embedded systems for non mission-critical services pass much of the deep-learning workload to remote cloud servers. In applications such as advanced driver assistance and robotic control, the system cannot rely on the network connection always being available. Therefore, local high-performance support for inferencing is vital. Only the training workload is routinely offloaded onto remote cloud servers.

Designers can exploit some features of neural networks to improve processing efficiency on systems used for inferencing. Typically, training demands high precision in the floating-point arithmetic used to compute neural weights to minimize the errors that could accumulate from multiple rounding errors during passes backwards and forwards through the deep layered structure. In most cases, 32-bit floating point has been shown to be sufficient in terms of precision.

For inferencing, errors are less likely to accumulate and networks can tolerate much lower precision representations for many of the connections. A post-training analysis can show which connections are likely to be unaffected by a reduction in precision. Often 8-bit fixed-point arithmetic is sufficient, and, for some connections, 4-bit resolution does not increase errors significantly. Systems will benefit from the ability to reconfigure datapaths so they can process many streams in parallel at 4-bit or 8-bit precision. But designers will want to retain the ability to combine execution units for high-precision arithmetic where needed.

Clearly, machine-learning systems call for a hardware substrate that provides a combination of high performance and plasticity.

Substrates for Machine Learning (and the Importance of the eFPGA)

A number of processing fabrics are available to support high-performance machine learning. But for use in real-time embedded systems, some will be ruled out at an early stage due to power consumption and performance reasons. For example, general-purpose processors offer high flexibility but low overall performance and energy efficiency.

When deep learning took hold in the research community in the first half of this decade, the general-purpose graphics processing unit (GPGPU) became a popular choice for both training and inferencing. The GPGPU provides hundreds of on-chip floating-point units, able to sum the inputs for many neurons in parallel much faster than clusters of general-purpose CPUs.

There is a drawback with applying GPGPUs to deep-learning architectures — these devices are designed primarily for accelerating 2D and 3D graphics applications, which employ homogeneous and predictable memory access patterns. For example, the shader cores in GPGPUs are tightly connected to small local memories designed to be filled using block transfers. This structure favors algorithms with arithmetically intensive operations on data that can easily be grouped closely together in memory. Such a structure can be used to process convolutional neural-network layers reasonably efficiently. However, other types of layers can prove troublesome because they place a greater emphasis on data transfers between neurons. These access patterns make the local-memory architecture less efficient, reducing both performance and energy efficiency. Furthermore, the emphasis on peak floating-point performance in the GPGPU tends to make the architecture unsuitable for embedded use. GPGPUs often require sustained power levels that exceed 150W, demanding active cooling and a permanent mains supply.

An ASIC implementation armed with custom logic and memory managers can overcome the bottlenecks that challenge GPGPUs in the implementation of deep-learning systems. Memory management units that can be tuned for the different access patterns encountered in neural-network code can do a much better job of enhancing overall speed.

In structures such as convolutional neural network layers, power savings can be achieved by not transferring data in and out of local or intermediate memories. Instead, the fabric can adopt the structure of a systolic array and pump results directly to the execution units that need them.

This custom approach need not bake the neural-network structure into hardware — a potentially wasteful approach. For example, the AlexNet implementation employs three different sizes of filters within its convolution layers: 11 × 11, 5 × 5, and 3 × 3. There is no need even to create separate execution units to implement these filters directly. The smallest filter size can be used as the basis for the convolution with larger filter sizes handled by using multiple passes through the smaller 3 × 3 filters with a buffer memory used to cache intermediate values. Despite allowing for logic reuse between the different layers, this approach slows down the implementation, which has to be compensated for through the deployment of a larger number of filter processors.

Even with elements shared across layers, the problem that faces any ASIC implementation is its relative inflexibility compared with software-based processors. It is possible to prototype a wide range of deep-learning structures and then choose to optimize one for deployment in silicon. A particular application may need to deploy more convolution layers or increase the complexity of the filter kernels to handle a particular kind of input. Supporting this complexity may require an increased number of filter-kernel processors relative to other hardware accelerators. This structure can be accommodated by an ASIC, but it may prove to be a poor fit for a changing algorithm or an adjacent application.

FPGAs provide a way of achieving the benefits of custom processors and memory-management techniques without locking the implementation to a specific, immutable hardware structure. Many FPGA architectures today provide a mix of fully customizable logic and digital signal processing (DSP) engines that provide support for both fixed- and floating-point arithmetic. In many cases, the DSP engines employ a building-block approach, composed of 8-bit or 16-bit units, that allows them to be combined to support higher-precision data types. Low precision can be accommodated via logic implemented in the fabric's LUTs.

The ability to rework the logic array within an FPGA makes it easy to tune the structure of the parallel processors and the routing between them for the specific needs of the application. The freedom remains to make changes later when the results from training indicate ways in which layers can be expanded or rearranged to improve performance. The disadvantage of the standalone FPGA is the need of suppliers to support many markets with the same solution. Suppliers need to provide a standard set of functions that apply as well to network switches as they do to deep learning. As a result, vendors find it hard to react to the specific needs of a market sector even when opportunities arise for converting common functions into more silicon- and power-efficient hardwired cores.

The relative inefficiency of the programmable logic array may mean a user has to compromise on performance – sharing functions among different layers within the neural network when the application really demands dedicated functionality for some high-throughput parts of the network. One approach is to augment the FPGA with a smaller ASIC that provides acceleration for commonly used functions, such as convolution kernels or max-pooling calculations.

The ability to harness specialized off-chip devices creates possibilities for novel approaches in dealing with the high throughput requirements of neural network. For example, ternary content-addressable memories (TCAMs) can not only provide fast, efficient temporary storage they can be employed to perform neuronal operations at high speed in Hopfield networks and layers with irregular or changing connectivity. TCAMs are difficult and expensive to build using programmable logic. Discrete devices are more silicon efficient but present drawbacks.

However, moving data between discrete devices increases energy consumption and the complexity of design. Programmable I/O circuitry used to transfer data off chip accounts for half of the total power consumption of the typical high-density standalone FPGA. The need for SerDes I/O to restrict the number of PCB traces further increases latency (problematic for many real-time inferencing applications) and increases design complexity because of signal-integrity issues.

Embedding an FPGA fabric in an SoC provides a solution to the drawbacks of both the standalone FPGA and ASIC, and the issues of passing data between them. One or more FPGA slices embedded into an ASIC provides the ability to tune the performance of the neural network on the fly, delivering the high data-transfer bandwidth required to make full use of customized engines. Embedded FPGAs make it possible to achieve the best balance between throughput and reprogrammability and deliver the performance that real-world machine-learning systems require.

The ability to bring FPGA blocks on-chip also saves significant silicon area by:

- Eliminating the large, power hungry I/O associated with a standalone FPGA
- Moving fixed functions to more efficient ASIC blocks
- Conversion of repetitive functions to Speedcore custom blocks (see the next section).

This die area savings translates into reduced manufacturing cost.

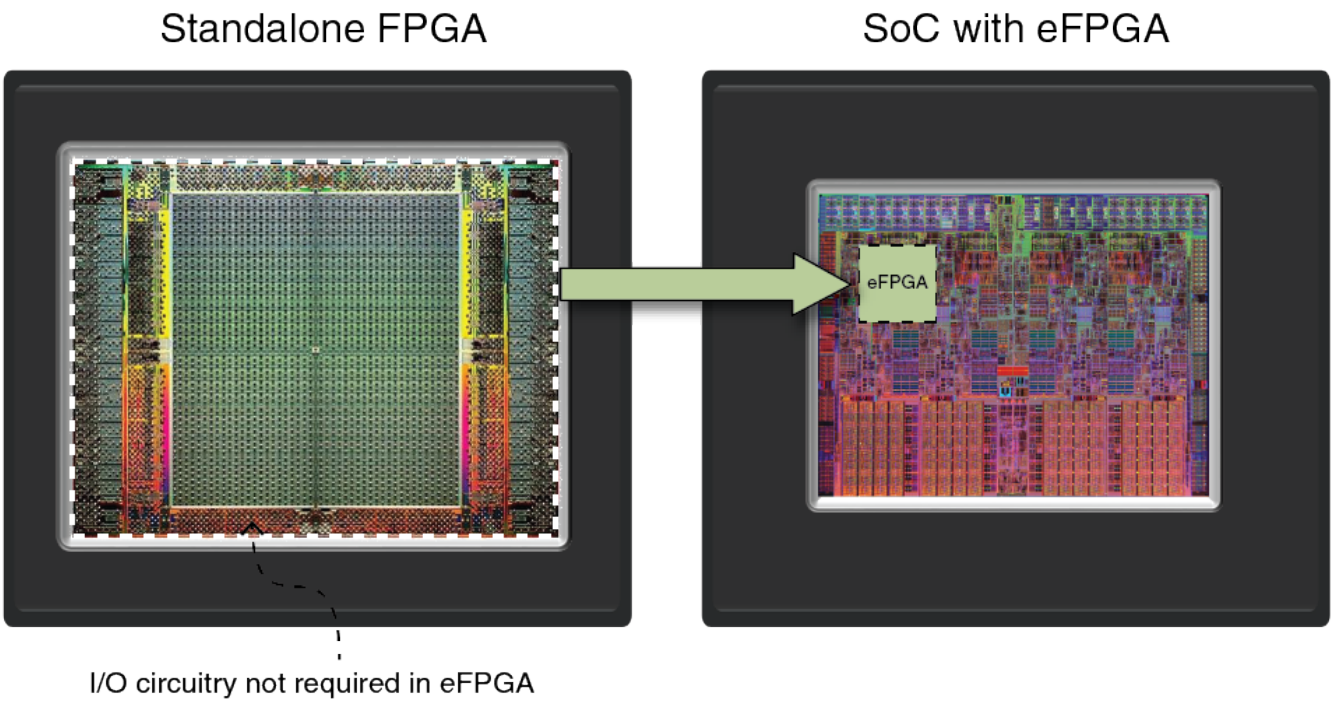

Figure 2: ***Embedding Programmable Logic Saves Space Overall by Removing the Need for Large, Power-Hungry I/O Circuitry***

eFPGA for Machine Learning

Designed specifically to be an embedded in SoCs and ASICs, Achronix Speedcore™ eFPGA IP is a highly flexible solution that supports the data throughput required in high-performance machine-learning applications. Using its slice-based architecture, Speedcore IP provides designers with the ability to mix and match eFPGA functions as required by the application. The core functions include logic based on four-input LUTs, small logic-oriented memories (LRAMs) for register files and similar uses, larger block RAMs (BRAMs), and configurable DSP blocks. The column-based architecture of the Speedcore fabric provides the ability to mix resources exactly as required by the application.

The core functions can be augmented with custom blocks that provide more specialized features that are silicon-intensive in programmable logic, such as TCAMs, ultrawide multiplexers and memory blocks optimized for pipelined accesses. One Achronix customer requested a custom processing and RAM blocks with widths optimized for CNN-oriented multiply-accumulate functions. Compared to an architecture that employed standard arithmetic and RAM blocks, the customer achieved a die size reduction of 38%, achieving a throughput of 1 Gop/s

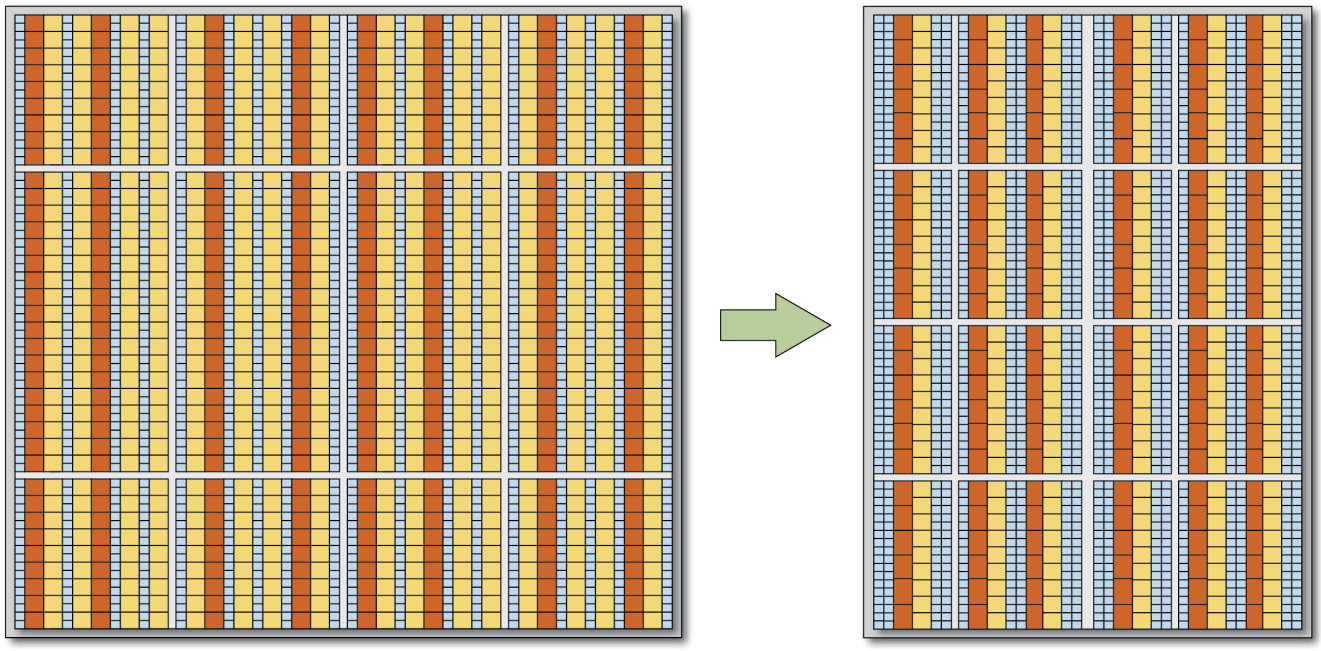

Figure 3: ***A 41% Die Space Saving was Achieved by Using Speedcore Custom Blocks***

Table 1: ***Custom Block Conversion for YOLO Details***

	Before				After				Savings
	Function	Logic Height	Count	Total Area	Function	Logic Height	Count	Total Area	
DSP Block	18 × 27	2	576	1,152	3 × (16 × 8)	2	216	432	63%
RAM	10 × 512	2	288	576	16 × 512	3	144	432	25%

Through the embeddable architecture, access to the programmable fabric is available to custom cores in the SoC without the energy and performance penalties of off-chip accesses. With no need for programmable I/O buffers around the FPGA fabric, overall die area within the solution is reduced. Moreover, the modular nature of the architecture makes it easy to port the technology to a wide variety of process technologies, down to the emerging 7 nm nodes.

To enable easy development of systems based around an eFPGA fabric, Speedcore IP fully supports a JTAG-based test and debug architecture. The Snapshot debug macro provides the ability to capture large quantities of on-chip data and deliver it to the software debugger to aid in the development and performance optimization of algorithms mapped to the eFPGA fabric.

The result is an architecture that provides the best possible starting point for real-time AI acceleration for embedded systems that range from consumer appliances through to advanced robotics and autonomous vehicles.

Conclusion

Machine-learning techniques represent a new frontier for embedded systems. Real-time AI will augment a wide variety of applications, but it can only deliver on its promise if it can be performed in a cost-effective, power-efficient way. Existing solutions such as multi-core CPUs, GPGPU and standalone FPGAs can be used to support advanced AI algorithms such as deep learning, but they are poorly positioned to handle the increased demands developers are placing on hardware as their machine-learning architectures evolve.

AI requires a careful balance of datapath performance, memory latency, and throughput that requires an approach based on pulling as much of the functionality as possible into an ASIC or SoC. But that single-chip device needs plasticity to be able to handle the changes in structure that are inevitable in machine-learning projects. Adding eFPGA technology provides the mixture of flexibility and support for custom logic that the market requires.

Achronix provides not only the building blocks required for an AI-ready eFPGA solution, but also delivers a framework that supports design through to debug and test of the final application. Only Achronix Speedcore IP has the right mix of features for advanced AI that will support a new generation of real-time, self-learning systems.

Achronix Semiconductor Corporation

2953 Bunker Hill Lane, Suite 101
Santa Clara, CA 95054
USA

Phone : 855.GHZ.FPGA (855.449.3742)
Fax : 408.286.3645
E-mail : info@achronix.com