Harnessing the Power of AI: An Easy Start with Lattice's sensAI

A Lattice Semiconductor White Paper.

January 2019

Artificial intelligence, or AI, is everywhere. It's a revolutionary technology that is slowly pervading more industries than you can imagine. It seems that every company, no matter what their business, needs to have some kind of AI story. In particular, you see AI seriously pursued for applications like self-driving automobiles, the Internet of Things (IoT), network security, and medicine. Company visionaries are expected to have a good understanding of how AI can be applied to their businesses, and success by early adopters will force holdouts into the fray.

Not all AI is the same, however, and different application categories require different AI approaches. The application class that appears to have gotten the most traction so far is embedded vision. AI for this category makes use of so-called convolutional neural networks, or CNNs, which attempt to mimic the way that the biological eye is believed to operate. We will focus on vision in this AI whitepaper, even though many of the concepts will apply to other applications as well.

Learn more:

www.latticesemi.com

Contact us online:

www.latticesemi.com/contact
www.latticesemi.com/buy

TABLE OF CONTENTS

AI Edge Requirements

AI involves the creation of a trained model of how something works. That model is then used to make inferences about the real world when deployed in an application. This gives an AI application two major life phases: training and inference.

Training is done during development, typically in the cloud. Inference, on the other hand, is required by deployed devices as an ongoing activity. Because inference can also be a computationally difficult problem, much of it is currently done in the cloud. But there is often little time to make decisions. Sending data to the cloud and then waiting until a decision arrives back can take time – and by then, it may be too late. Making that decision locally can save precious seconds.

This need for real-time control applies to many application areas where decisions are needed quickly. Many such examples detect human presence:

Smart-home appliances

Consumer smart audio/video electronics

Smart doorbells

Vending machines

Security cameras

Smart doors

Other always-on applications include:

Smart speakers

Retail store cameras

Selfie drones

Toll-gate cameras

Machine vision

After-market
automotive cameras

Because of this need for quick decisions, there is a strong move underway to take inference out of the cloud and implement it at the “edge” – that is, in the devices that gather data and then take action based on the AI decisions. This takes the delays inherent in the cloud out of the picture.

There are two other benefits to local inference. The first is privacy. Data enroute to and from the cloud, and data stored up in the cloud, is subject to hacking and theft. If the data never leaves the equipment, then there is far less opportunity for mischief.

The other benefit relates to the bandwidth available in the internet. Sending video up to the cloud for real-time interpretation chews up an enormous amount of bandwidth. Making the decisions locally frees that bandwidth up for all of the other demanding uses.

In addition:

- Many such devices are powered by a battery – or, if they are mains-supplied, have heat constraints that limit how much power is sustainable. In the cloud, it’s the facility’s responsibility to manage power and cooling.
- AI models are evolving rapidly. Between the beginning and end of training, the size of the model may change dramatically, and the size of the required computing platform may not be well understood until well into the development process. In addition, small changes to the training can have a significant impact on the model, adding yet more variability. All of this makes it a challenge to size the hardware in the edge device appropriately.
- There will always be tradeoffs during the process of optimizing the models for your specific device. That means that a model might operate differently in different pieces of equipment.
- Finally, edge devices are often very small. This limits the size of any devices used for AI inference.

All of this leads to the following important requirements for interference at the edge:

Engines for making AI inference at the edge must:

- Consume very little power
- Be very flexible
- Be very scalable
- Have a small physical footprint

Lattice's sensAI offering lets you develop engines with precisely these four characteristics. It does so by including a hardware platform, soft IP, a neural-net compiler, development modules, and resources that will help get the design right quickly.

Inference Engine Options

There are two aspects to building an inference engine into an edge device: developing the hardware platform that will host the execution of the model, and developing the model itself.

Execution of a model can, in theory, take place on many different architectures. But execution at the edge, taking into account the power, flexibility, and scalability requirements above, limits the choices – particularly for always-on applications.

- **MCUs -** The most common way of handling AI models is by using a processor. That may be a GPU or a DSP, or it may be a microcontroller. But the processors in edge devices may not be up to the challenge of executing even simple models; such a device may have only a low-end microcontroller (MCU) available. Using a larger processor may violate the power and cost requirements of the device, so it might seem like AI would be out of reach for such devices.

  This is where low-power FPGAs can play an important role. Rather than beefing up a processor to handle the algorithms, a Lattice ECP5 or UltraPlus FPGA can act as a co-processor to the MCU, providing the heavy lifting that the MCU can't handle while keeping power within the required range. Because these Lattice FPGAs can implement DSPs, they provide computing power not available in a low-end MCU.

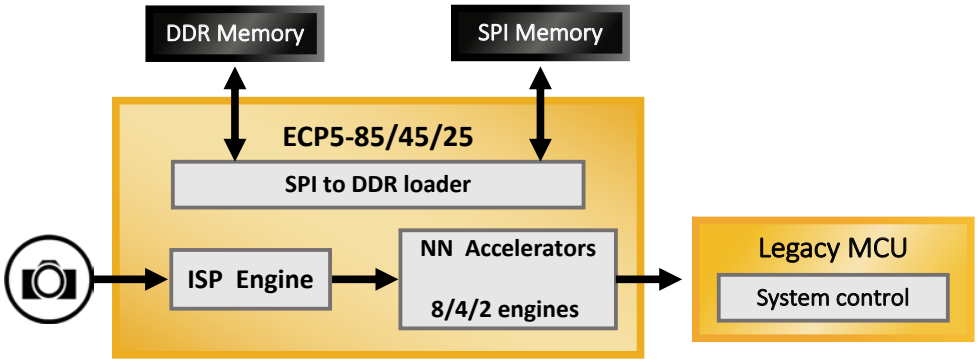


Figure 1: FPGA as a Co-Processor to MCU

- **ASICS and ASSPs -** For AI models that are more mature and will sell in high volumes, ASICs or application-specific standard products (ASSPs) may be appropriate. But, because of their activity load, they will consume too much power for an always-on application.

  Here Lattice FPGAs can act as activity gates, handling wake-up activities involving wake words or recognition of some broad class of video image (like identifying something that looks like it might be a person) before waking up the ASIC or ASSP to complete the task of identifying more speech or confirming with high confidence that an artifact in a video is indeed a person (or even a specific person).
  The FPGA handles the always-on part, where power is most critical. While not all FPGAs can handle this role, since many of them still consume too much power, Lattice's ECP5 and UltraPlus FPGAs have the power characteristics necessary for this role.

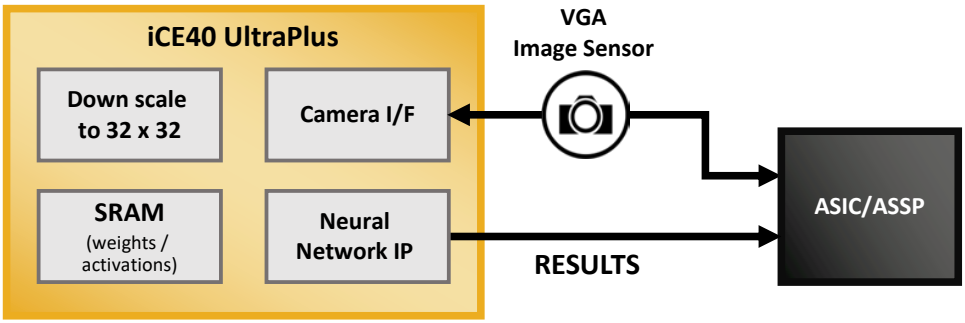


Figure 2: FPGA as activity gate to ASIC/ASSP

- **Stand-Alone FPGA AI Engines -** Finally, low-power FPGAs can act as stand-alone, integrated AI engines. The DSPs available in the FPGAs take the starring role here. Even if an edge device has no other computing resources, AI capabilities can be added without breaking the power, cost, or board-area budgets. And they have the flexibility and scalability necessary for rapidly evolving algorithms.

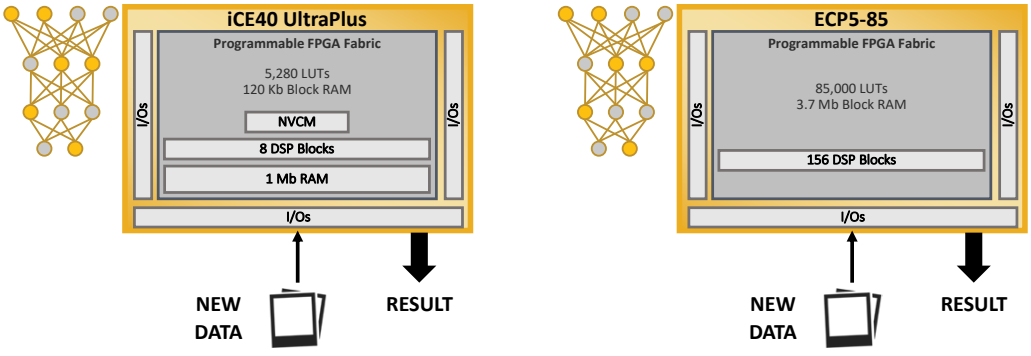


Figure 3: Stand-alone, Integrated FPGA Solution

Building an Inference Engine in a Lattice FPGA

Designing hardware that will execute an AI inference model is an exercise in balancing the number of resources needed against performance and power requirements. Lattice's ECP5 and UltraPlus familes provide this balance.

The ECP5 family has three members of differing sizes that can host from one to eight inference engines. They contain anywhere from 1 Mb to 3.7 Mb of local memory. They run up to 1 W of power, and they have a 100 mm^2 footprint.

The UltraPlus family, by contrast, has power levels as low as one thousandth the power of the ECP5 family, at 1 mW. Consuming a mere 5.5 mm^2 of board area, it contains up to eight multipliers and up to 1 Mb of local memory.

Lattice also provides CNN IP designed to operate efficiently on these devices. For the ECP5 family, Lattice has a CNN Accelerator.

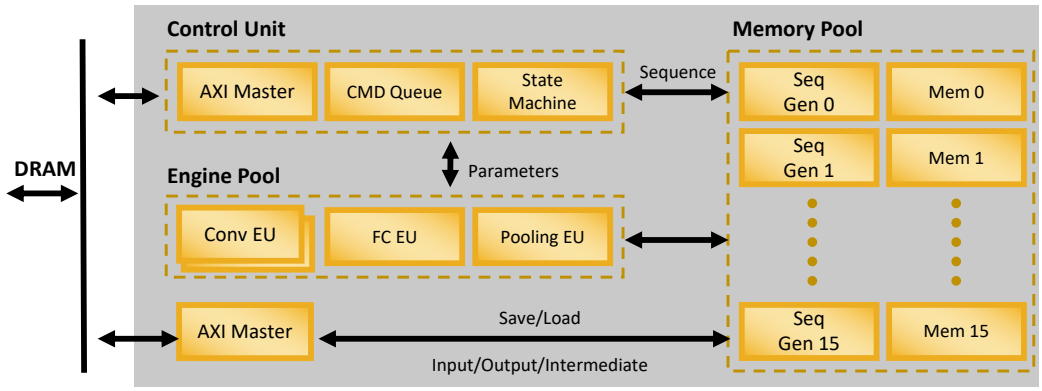


Figure 4: CNN Accelerator for the ECP5 family

For the UltraPlus family, Lattice provides a CNN Compact Accelerator.

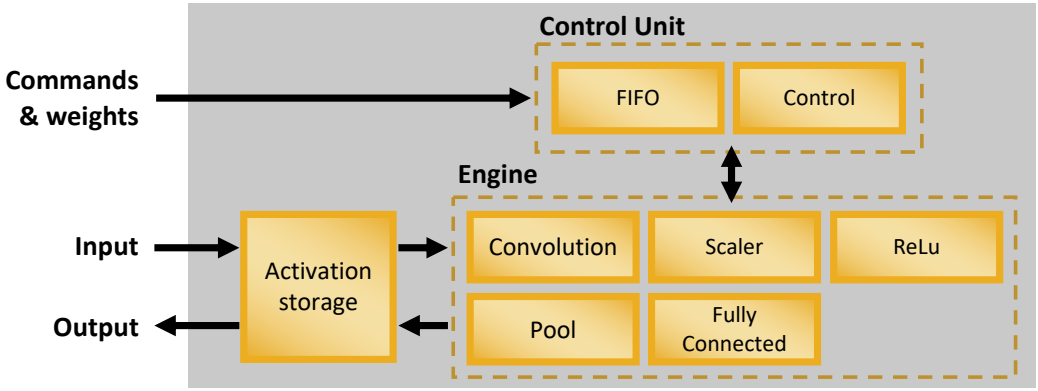
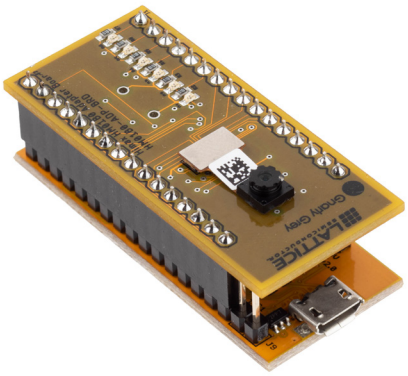


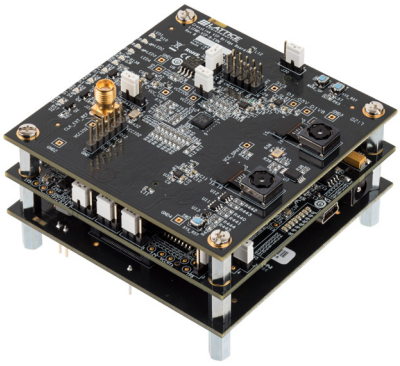
Figure 5. Compact CNN Accelerator for the UltraPlus family

We won't dive into the details here; the main point is that you don't have to design your own engine from scratch. Much more information is available from Lattice regarding these pieces of IP.

Finally, you can run examples like this and test them out on development modules, with one for each device family. The Himax HM01B0 UPduino shield uses an UltraPlus device, requiring 22 x 50 mm^2 of space. The Embedded Vision Development Kit uses an ECP5 device, claiming 80 x 80 mm^2 of space.



Himax HM01B0 UPduino Shield



Embedded Vision Development Kit

Figure 6. Development modules for evaluation of AI application

Given an FPGA, soft IP, and all of the other hardware details needed to move data around, the platform can be compiled using Lattice's Diamond design tools in order to generate the bitstream that will configure the FPGAs at each power-up in the targeted equipment.

Building the Inference Model in a Lattice FPGA

Creating an inference model is very different from creating the underlying execution platform. It's more abstract and mathematical, involving no RTL design. There are two main steps: creating the abstract model and then optimizing the model implementation for your chosen platform.

Model training takes place on any of several frameworks designed specifically for this process. The two best-known frameworks are Caffe and TensorFlow, but there are others as well.

A CNN consists of a series of layers – convolution layers, along with possible pooling and fully connected layers – each of which has nodes that are fed by the result of the prior layer. Each of those results is weighted at each node, and it is the training process that decides what the weights should be.

The weights output by the training frameworks are typically floating-point numbers. This is the most precise embodiment of the weights – and yet most edge devices aren't equipped with floating-point capabilities. This is where we need to take this abstract model and optimize it for a specific platform – a job handled by Lattice's Neural Network Compiler.

The Compiler allows you to load and review the original model as downloaded from one of the CNN frameworks. You can run performance analysis, which is important for what is likely the most critical aspect of model optimization: quantization.

Because we can't deal with floating-point numbers, we have to convert them to integers. That means that we will lose some accuracy simply by virtue of rounding off floating-point numbers. The question is, what integer precision is needed to achieve the accuracy you want? 16 bits is usually the highest precision used, but weights – and inputs – may be expressed as smaller integers. Lattice currently supports 16-, 8-, and 1-bit implementations. 1-bit designs are actually trained in the single-bit integer domain to maintain accuracy.Clearly, smaller data units mean higher performance, smaller hardware, and, critically, lower power. But, make the precision too low, and you won't have the accuracy required to faithfully infer the objects in a field of view.

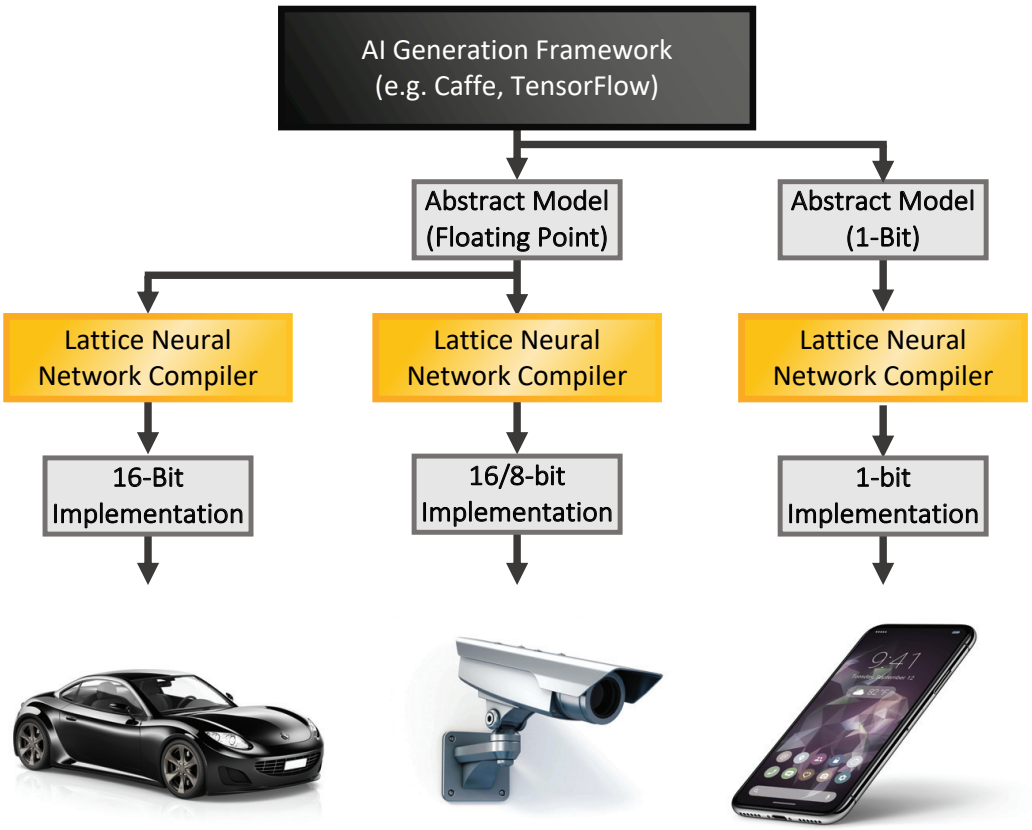


Figure 7. A single model can be optimized differently for different equipment

So the neural-network compiler lets you create an instruction stream that represents the model, and those instructions can then be simulated or outright tested to judge whether the right balance has been struck between performance, power, and accuracy. This is usually measured by the percentage of images that were correctly processed out of a set of test images (different from the training images).

Improved operation can often be obtained by optimizing a model, including pruning of some nodes to reduce resource consumption, and then retraining the model in the abstract again. This is a design loop that allows you to fine-tune the accuracy while operating within constrained resources.

Two Detection Examples

We can see how the tradeoffs play out with two different vision examples. The first is a face-detection application; the second is a human-presence-detection application. We can view how the differences in the resources available in the different FPGAs affects the performance and power of the resulting implementations.

Both of these examples take their inputs from a camera, and they both execute on the same underlying engine architecture. For the UltraPlus implementation, the camera image is downsized and then processed through eight multipliers, leveraging internal storage and using LEDs as indicators.

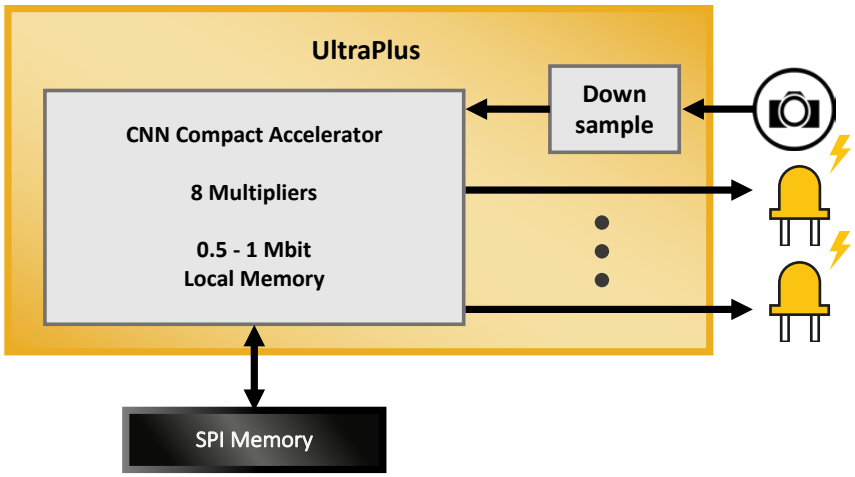


Figure 8. UltraPlus platform for face-detection and human-presence applications

The ECP5 family has more resources, and so it provides a platform with more computing power. Here the camera image is pre-processed in an image signal processor (ISP) before being sent into the CNN. The results are combined with the original image in an overlay engine that allows text or annotations to be overlaid on the original image.

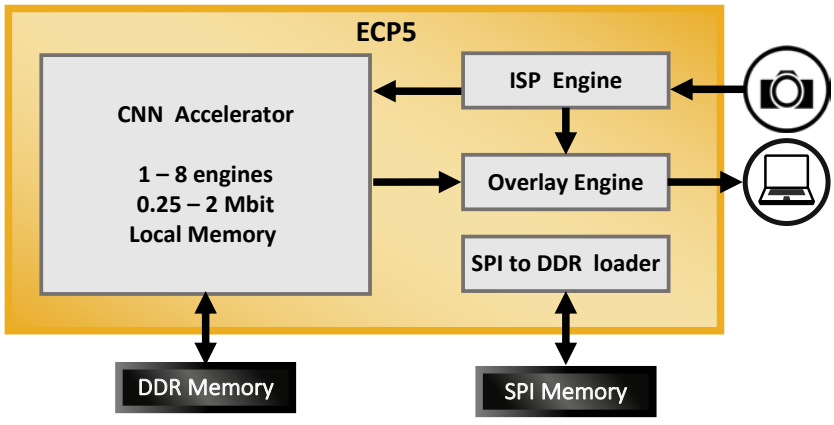
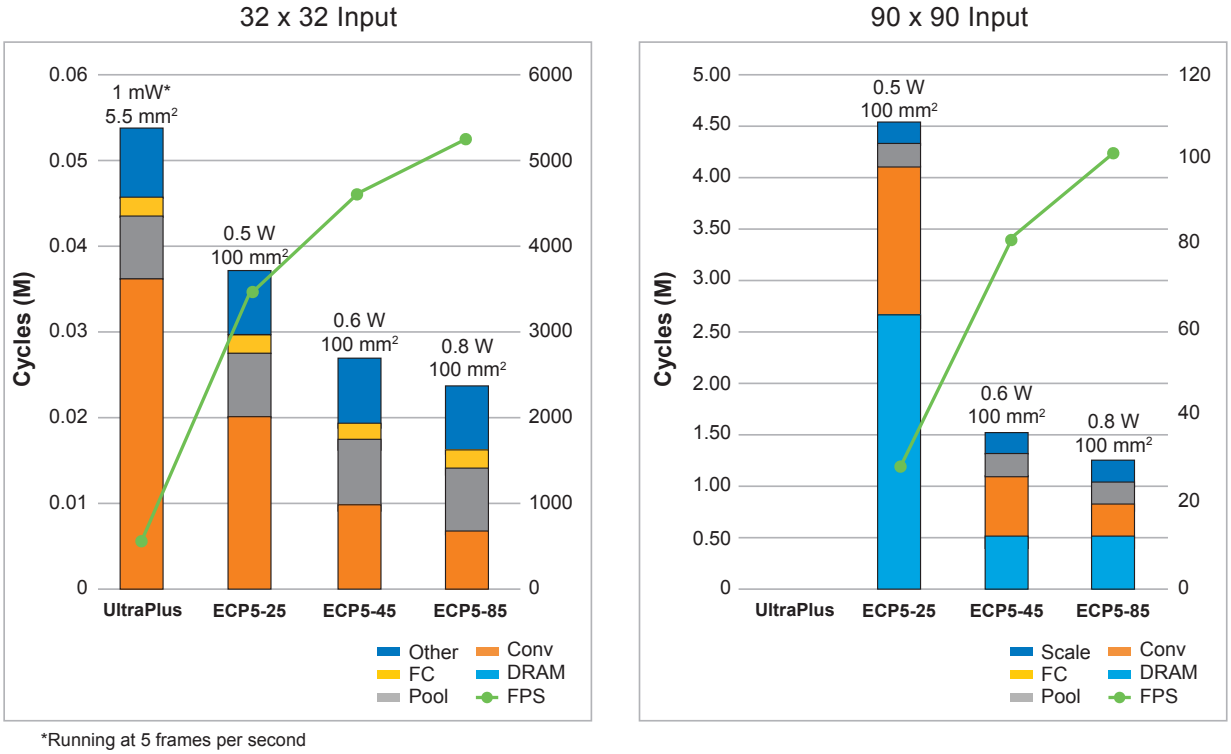


Figure 9. ECP5 platform for face-detection and human-presence applications

We can use a series of charts to measure the performance, power, and area of each implementation of the applications. We also do two implementations of each application: one with fewer inputs and one with more inputs.

For the face-detection application, we can see the results in Figure 7. Here the two implementations use 32 x 32 inputs for the simple version and 90x90 inputs for the more complex one.



*Running at 5 frames per second

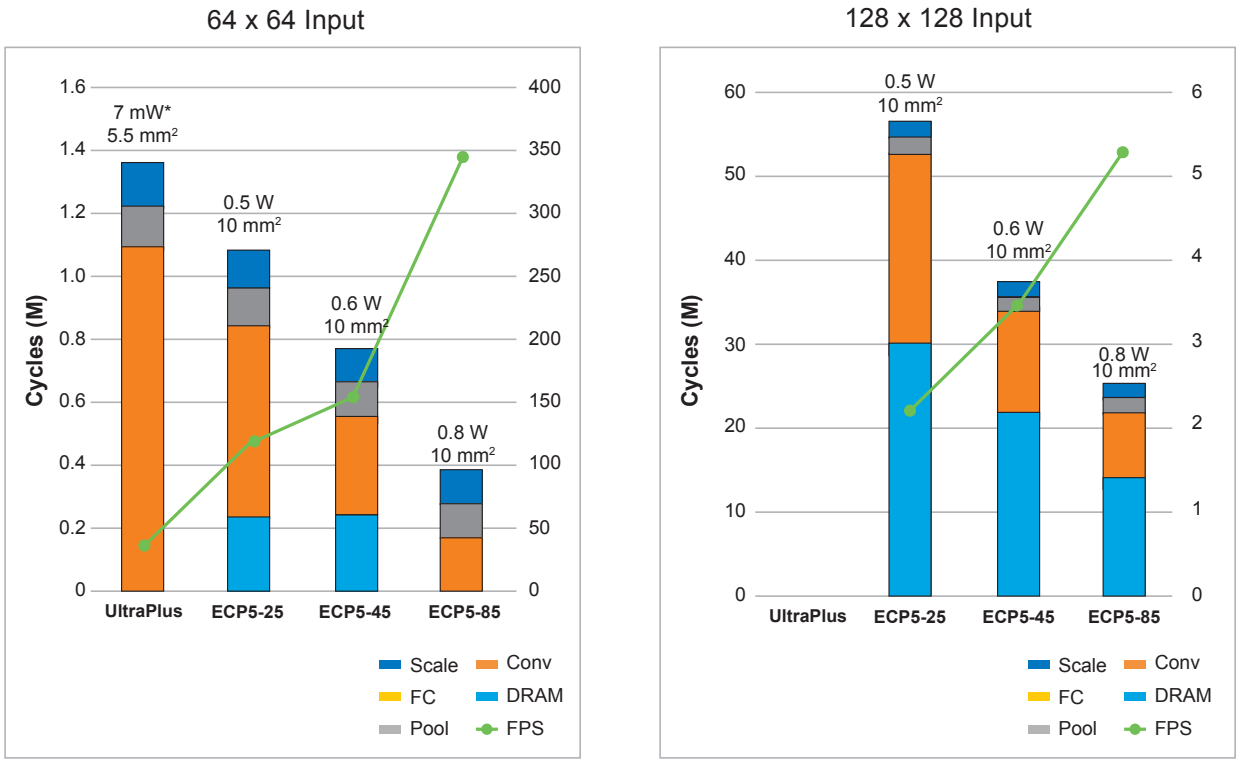
Figure 10. Performance, power, and area results for simple and complex implementations of the face-recognition application in UltraPlus and ECP5 FPGAs

The left-hand axis shows the number of cycles required to process an image and how those cycles are spent. The right-hand axis shows the resulting frames-per-second (fps) performance for each implementation (the green line). Finally, each implementation shows the power and area.

The orange bars in the 32 x 32 example on the left represent the cycles spent on convolution. The UltraPlus has the fewest multipliers of the four examples; the other three are ECP5 devices with successively more multipliers. As the number of multipliers increases, the number of cycles required for convolution decreases.

The 90 x 90 example is on the right, and the results are quite different. There is a significant new blue contribution to the cycles on the bottom of each stack. This is the result of the more complex design using more memory than is available internally in the devices. As a result, they have to go out to DRAM, which hurts performance. Note also that this version cannot be implemented in the smaller UltraPlus device.

A similar situation holds for the human-presence application. Here the simple version uses 64 x 64 inputs, while the complex version works with 128 x 128 inputs.



*Running at 5 frames per second

Figure 11. Performance, power, and area results for simple and complex implementations of the human-presence application in UltraPlus and ECP5 FPGAs

Again, more multipliers reduce the convolution burden, and relying on DRAM hurts performance.

The performance for all versions is summarized in Figure 9. This includes a measure of the smallest identifiable object or feature in an image, expressed as a percent of the full field of view. Using more inputs helps here, providing additional resolution for smaller objects.

		Device Size / Power / Performance			
Network	**Smallest Object**	**UltraPlus 1-7 mW² 5.5 mm²**	**ECP5-25 0.5 W 100 mm²**	**ECP5-45 0.8 W 100 mm²**	**ECP5-85 0.8 W 100 mm²**
Face Detection 32 x 32 Input	50%	465	3360	4511	5251
Face Detection 90 x 90 Input	20%	--	28	82	101
Human Presence Detect 64 x 64 Input	20%	18	115	161	338
Human Presence Detect 128 x 128 Input	10%	--	2.3	3.5	5.4

Figure 12. Performance summary of the two example applications in four different FPGAs

Summary

In summary, then, edge-inference AI designs that demand low power, flexibility, and scalability can be readily implemented in Lattice FPGAs using the resources provided by the Lattice sensAI offering. It makes available the critical elements needed for successful deployment of AI algorithms:

- Neural network compiler
- Neural engine soft IP
- Diamond design tools
- Development boards
- Reference designs

Much more information is available from Lattice; go to www.latticesemi.com to start using the power of AI in your designs.

Learn more:

www.latticesemi.com

Contact us online:

www.latticesemi.com/contact
www.latticesemi.com/buy