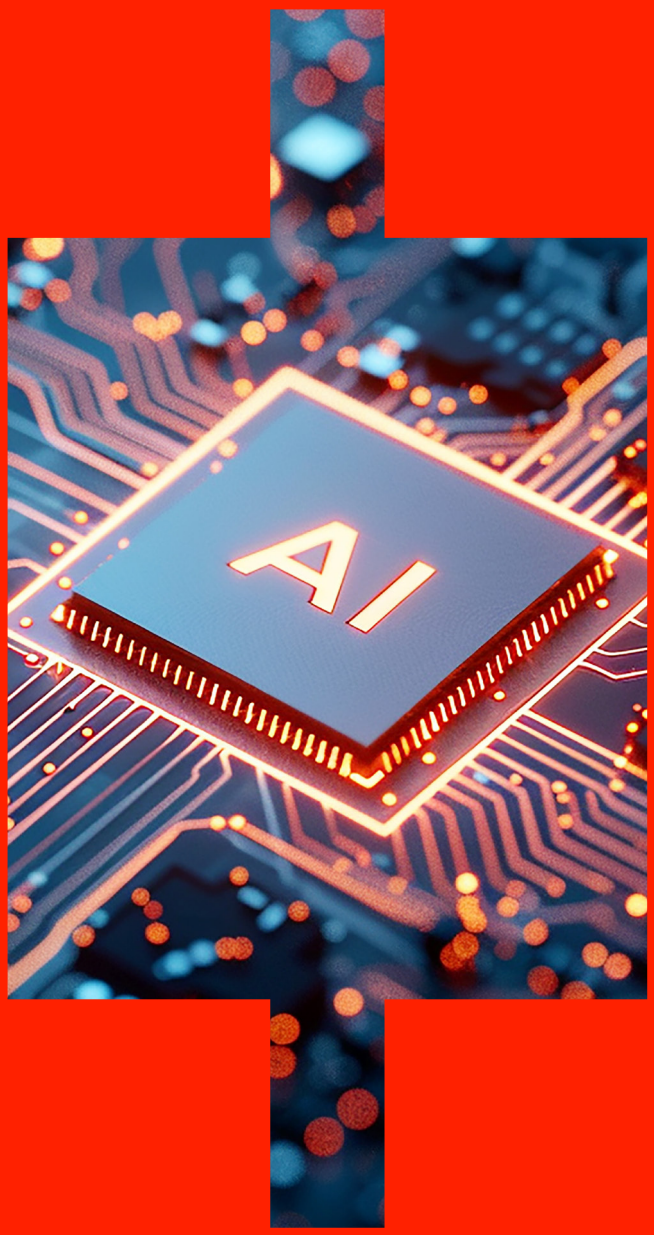
We get technical

How to use FPGA SoCs for secure and connected hard real-time systems

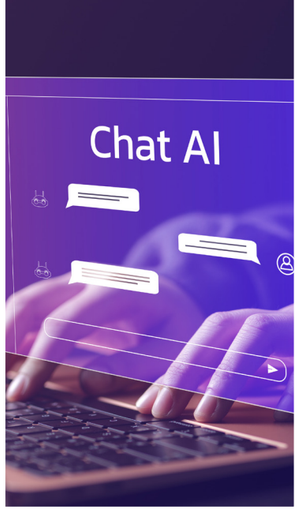
How automation, machine learning, and Blockchain are driving the future of electronics manufacturing

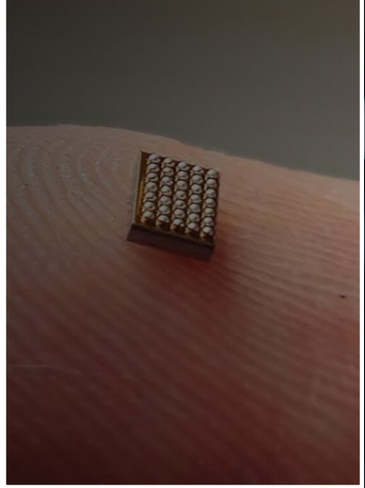
Quickly implement spoofing-resistant face recognition without a Cloud connection

Why and how to use Efinix FPGAs for AI/ML imaging



DigiKey





contents

4 AI development potential with the Agilex 5 system on module

8 How to run a 'Hello World' machine learning model on STM32 microcontrollers

14 How to use FPGA SoCs for secure and connected hard real-time systems

20 **Special feature: retroelectro** Programming a calculator to form concepts: the birth of artificial intelligence

28 How automation, machine learning, and Blockchain are driving the future of electronics manufacturing

34 Quickly implement spoofing-resistant face recognition without a Cloud connection

44 Why and how to use Efinix FPGAs for AI/ML imaging – Part 1: getting started

50 Why and how to use Efinix FPGAs for AI/ML imaging – Part 2: Image capture and processing

56 Powering the Edge: The Evolution of AI from Digital to Neuromorphic Systems for Ultra-Low Power Performance

60 All about AI/machine learning

Editor's note

Edge AI and machine learning have transformed the way systems and devices process data, enabling real-time decision-making at the source rather than relying on remote servers. As engineers navigate the rapid evolution of technology, understanding and applying these concepts have become crucial for developing intelligent, autonomous systems across a range of industries, from manufacturing to healthcare.

Edge AI refers to the deployment of artificial intelligence algorithms on Edge devices, which are located at the periphery of a network, close to the data source. These devices are equipped with enough processing power to handle AI computations locally. This eliminates the need to send large volumes of data to a centralized Cloud for analysis, drastically reducing latency, bandwidth usage, and energy consumption. In many industrial applications, where immediate responses are critical, these advantages make Edge AI an attractive option.

Machine learning is the driving force behind the AI revolution, empowering systems to learn from data and improve their performance over time without explicit programming. When combined with Edge computing, machine learning enables the creation of systems that can operate autonomously, make decisions in real time, and adapt to changing conditions on the fly. This combination is already being applied in predictive maintenance, robotics, autonomous vehicles, and industrial automation.

As the demand for intelligent, connected devices continues to rise, engineers must stay ahead of the curve by embracing Edge AI and machine learning. These technologies are no longer confined to large-scale enterprises with vast resources but are increasingly accessible to a wider range of industries and applications. By understanding how to implement and optimise Edge AI and machine learning, engineers can create more responsive, efficient, and secure systems that will drive the next wave of innovation.

For more information,
please check out our website at
www.digikey.com/automation.

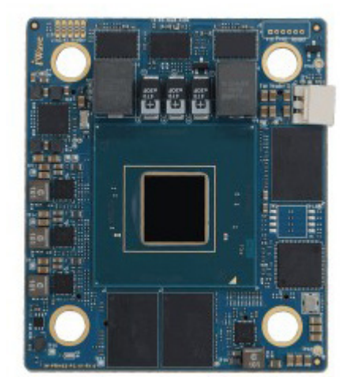
AI development potential with the Agilex 5 system on module

Written by:
Tawfeeq Ahmad

Artificial Intelligence (AI) is revolutionizing various industries by providing transformative solutions that significantly enhance efficiency, accuracy, and the ability to make informed decisions. In this landscape, the concept of Edge AI – processing AI algorithms on devices located at the Edge of a network – has emerged as a game-changing approach. It allows for real-time data processing, reduced latency, improved data privacy, and autonomy in decision-making, especially critical in sectors like healthcare, robotics, and industrial automation.

iWave, a pioneering force in embedded systems engineering, is at the forefront of this revolution, offering embedded platforms designed to push the boundaries of AI at the Edge. These platforms are specifically tailored for applications requiring high-performance computing and sophisticated AI/ML capabilities, such as media processing, robotics, and visual computing.

Figure 1. The iWave iW-RainboW-G58M SoM, powered by the Intel Agilex 5 FPGA which is the first FPGA to feature directly integrated AI capabilities.
Image source: iWave



Introducing iW-RainboW-G58M: the next generation of AI-infused FPGAs

In a significant advancement for the embedded systems market, iWave is thrilled to introduce the iW-RainboW-G58M System on Module (SoM) (Figure 1), powered by the Intel Agilex 5 FPGA. This is the first FPGA to feature AI capabilities integrated directly into its fabric, marking a new era in FPGA technology. The iW-RainboW-G58M is meticulously engineered for applications demanding high-performance, low-latency processing, and custom logic implementation with embedded AI/ML support, making it an ideal choice for industries such as medical imaging, robotics, and industrial automation.

The iW-RainboW-G58M SoM is compact, measuring just 60mm x 70mm, yet it is packed with powerful features. It supports the Intel Agilex 5 FPGA and SoC E-Series family in the B32A package, available in two distinct device variants to cater to a range of application needs:

- Group A: A5E 065A/052A/043A/028A/013A SoC FPGA – These variants offer higher performance and are suitable for applications requiring more complex processing capabilities
- Group B: A5E 065B/052B/043B/028B/013B/008B SoC FPGA – These variants provide cost-effective solutions for less demanding tasks, ensuring flexibility in design and implementation

The combination of these options allows developers to select the most appropriate FPGA variant for their specific application, balancing performance, power consumption, and cost.

Harnessing the full potential of Intel Agilex 5 FPGAs for Edge AI

Intel's Agilex 5 FPGAs and SoCs represent a significant leap forward in FPGA technology, especially in the context of AI and machine learning applications at the Edge. The Agilex 5 series builds on Intel's legacy of AI-optimized FPGAs, introducing the industry's first AI tensor block in a mid-range FPGA. This block is specifically designed to accelerate AI workloads, making these FPGAs a perfect fit for edge AI applications where real-time processing and decision-making are critical.

A key feature of the Agilex 5 FPGA is its asymmetric applications processor system, which includes dual Arm Cortex-A76 cores and dual Cortex-A55 cores. This configuration allows the FPGA to deliver exceptional processing power while optimizing power efficiency, a crucial factor in Edge

computing environments where power consumption must be minimized without compromising performance.

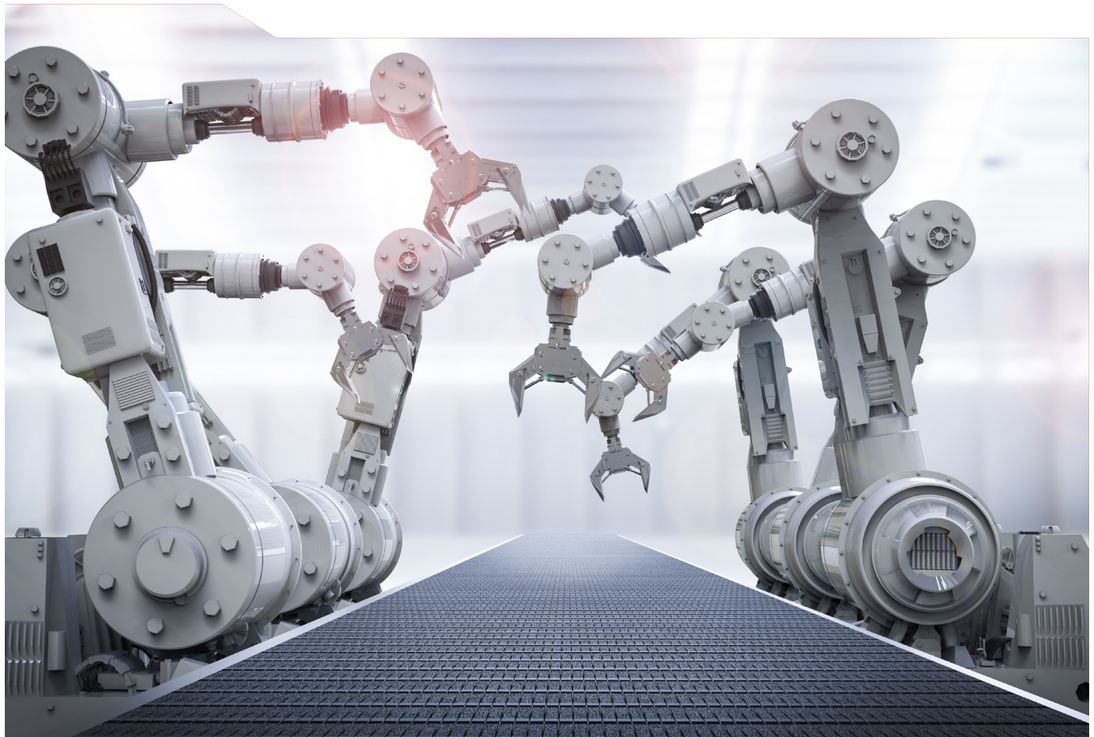
The Agilex 5 FPGA also includes enhanced Digital Signal Processing (DSP) capabilities, integrated with an AI tensor block. This combination allows the FPGA to handle complex AI tasks such as deep learning inference, image processing, and predictive analytics with greater efficiency and accuracy. Moreover, the FPGA's advanced connectivity features, including high-speed GTS transceivers that support data rates up to 28.1 Gbps, PCI Express* (PCIe*) 4.0 × 8, and outputs for DisplayPort and HDMI, make it a versatile solution for a wide range of applications.

Comprehensive AI/ML software ecosystem: accelerating development

The iW-RainboW-G58M SoM is complemented by a comprehensive software ecosystem that significantly accelerates AI and machine learning development. Central to this ecosystem is the support for popular AI frameworks such as TensorFlow and PyTorch, ensuring that developers can leverage these familiar platforms to create sophisticated AI models without steep learning curves.

A critical component of this ecosystem is the OpenVINO toolkit. This open-source toolkit is designed to optimize deep learning models for inference on a variety of hardware architectures, including CPUs, GPUs, and FPGAs. By using the OpenVINO toolkit, developers can ensure that their AI models are not only optimized for performance but are also highly portable across different hardware platforms, allowing for greater flexibility in deployment.

Additionally, the Intel FPGA AI Suite plays a pivotal role in simplifying the development process. This suite is designed with ease of use in mind, enabling FPGA designers, machine learning engineers, and software developers to create AI platforms that are optimized for FPGA architectures. By integrating with industry-standard tools such as TensorFlow, PyTorch, and the OpenVINO toolkit, the Intel FPGA AI Suite allows developers to speed up the development process while maintaining a high degree of reliability and performance in their AI solutions.



The suite also integrates seamlessly with the Intel Quartus Prime FPGA design software, a powerful tool that supports the design, analysis, and optimization of FPGA-based systems. This integration ensures that developers have access to a robust and proven workflow, reducing time to market and enhancing the overall reliability of their AI applications.

Cloud AI vs. Edge AI: a comparative analysis

As AI continues to evolve, the distinction between Cloud AI and Edge AI becomes increasingly important. Cloud AI, which relies on the vast computational resources of remote data centers, offers high scalability and the ability to process large volumes of data. However, this approach often comes with higher latency and potential security concerns due to the need for data transmission over the internet.

On the other hand, Edge AI offers significant advantages in scenarios where real-time processing, low latency, and enhanced data privacy are critical. By processing data locally on the device, Edge AI eliminates the need for constant communication with the cloud, reducing latency and improving the responsiveness of AI systems. This is particularly important in applications such as autonomous vehicles, industrial automation, and healthcare, where delays in decision-making can have serious consequences.

Moreover, Edge AI contributes to data privacy by keeping sensitive information on the local device, reducing the risk of data breaches associated with cloud-based processing. The hybrid approach, where edge devices perform initial data processing before transmitting it to the cloud for more complex analysis, is becoming increasingly popular. This method combines the strengths of both Edge AI and Cloud AI, allowing for efficient resource utilization, enhanced security, and improved system performance.

Ensuring longevity and comprehensive support: iWave's commitment to customers

One of iWave's key commitments is to ensure the long-term availability of its products. The company's product longevity program guarantees that its System on Modules (SoMs) are available for extended periods, often exceeding 10 years. This is especially important for industries like medical devices, aerospace, and industrial automation, where product lifecycles are typically long, and consistent component availability is critical.

In addition to longevity, iWave provides extensive technical support throughout the product development process. This support includes ODM (Original Design Manufacturer) services, such as carrier card design, thermal simulation, and system-level design, allowing customers to focus on their core competencies while iWave handles the complex aspects of hardware design and integration.

iWave's commitment to customer success is further demonstrated by the provision of comprehensive evaluation kits for its SoMs. These kits come with complete user documentation, software drivers, and a board support package, enabling customers to rapidly evaluate and prototype their designs. By offering these resources, iWave helps customers reduce development time and bring their products to market faster.

Summary

iWave's iW-RainboW-G58M SoM, with the Intel Agilex 5 FPGA that features integrated AI capabilities, is carefully engineered for high-performance, low-latency processing, and custom logic implementation with embedded AI/ML support applications. This makes it a good choice for industries such as medical imaging, robotics, and industrial automation.

How to run a 'Hello World' machine learning model on STM32 microcontrollers

Written by: Jacob Beningo

Machine learning (ML) has been all the rage in server and mobile applications for years, but it has now migrated and become critical on Edge devices. Given that Edge devices need to be energy efficient, developers need to learn and understand how to deploy ML models to microcontroller-based systems. ML models running on a microcontroller are often referred to as tinyML. Unfortunately, deploying a model to a microcontroller is not a trivial endeavor. Still, it is getting easier, and developers without any specialized training will find that they can do so in a timely manner.

This article explores how embedded developers can get started with ML using STMicroelectronics' STM32 microcontrollers. To do so, it shows how to create a 'Hello World' application by converting a TensorFlow Lite for Microcontrollers model for use in STM32CubeIDE using X-CUBE-AI.

Introduction to tinyML use cases

TinyML is a growing field that brings the power of ML to resource and power-constrained devices like microcontrollers, usually using deep neural networks. These microcontroller devices can then run the ML model and perform valuable work at the edge. There are several use cases where tinyML is now quite interesting.

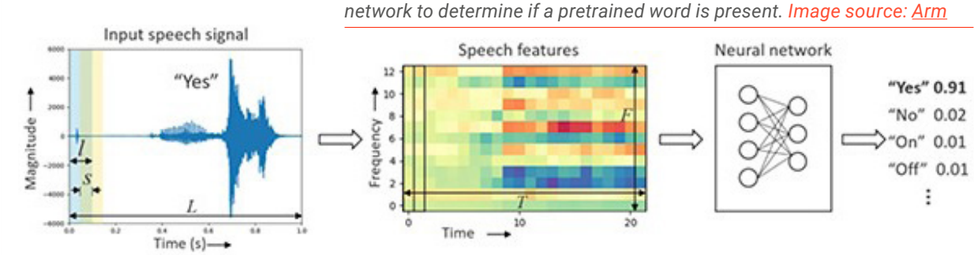
The first use case, which is seen in many mobile devices and home automation equipment, is keyword spotting. Keyword spotting allows the embedded device to use a microphone to capture speech and detect pretrained keywords. The tinyML model uses a time-series input that represents the speech and converts it to speech features, usually a spectrogram. The spectrogram contains frequency information over time. The spectrogram is then fed into a neural network trained to detect specific words, and the result is a probability that a particular word is detected. Figure 1 shows an example of what this process looks like.

The next use case for tinyML that many embedded developers are interested in is image recognition. The microcontroller captures images from a camera, which are then fed into a pre-trained model. The model can discern what is in the image. For example, one might be able to determine if there is a cat, a dog, a fish, and so forth. A great example of how image recognition is used at the edge is in video doorbells. The video doorbell can often detect if a human is present at the door or whether a package has been left.

One last use case with high popularity is using tinyML for predictive maintenance. Predictive maintenance uses ML to predict equipment states based on abnormality detection, classification algorithms, and predictive models. Again, plenty of applications are available, ranging from HVAC systems to factory floor equipment.

While the above three use cases are currently popular for tinyML, there are undoubtedly many potential use cases that developers can find.

Figure 1. Keyword spotting is an interesting use case for tinyML. The input speech is converted to a spectrogram and then fed into a trained neural network to determine if a pretrained word is present. Image source: Arm



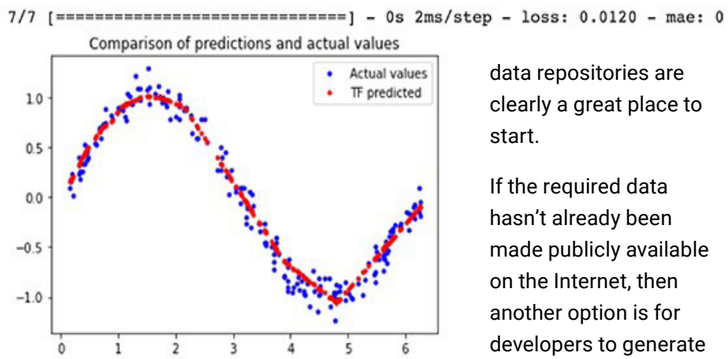


Figure 2. A comparison between TensorFlow model predictions for a sine wave versus the actual values. Image source: Beningo Embedded Group

Here's a quick list:

- Gesture classification
- Anomaly detection
- Analog meter reader
- Guidance and control (GNC)
- Package detection

No matter the use case, the best way to start getting familiar with tinyML is with a 'Hello World' application, which helps developers learn and understand the basic process they will follow to get a minimal system up and running. There are five necessary steps to run a tinyML model on an STM32 microcontroller:

1. Capture data
2. Label data
3. Train the neural network
4. Convert the model
5. Run the model on the microcontroller

Capturing, labelling, and training a 'Hello World' model

Developers generally have many options available for how they will capture and label the data needed to train their model. First, there are a lot of online training databases. Developers can search for data that someone has collected and labelled. For example, for basic image detection, there's CIFAR-10 or ImageNet. To train a model to detect smiles in photos, there's an image collection for that too. Online data repositories are clearly a great place to start.

If the required data hasn't already been made publicly available on the Internet, then another option is for developers to generate their own data. Matlab or some other tool can be used to generate the datasets. If automatic data generation is not an option, it can be done manually. Finally, if this all seems too time-consuming, there are some datasets available for purchase, also on the Internet. Collecting the data is often the most exciting and interesting option, but it is also the most work.

The 'Hello World' example being explored here shows how to train a model to generate a sine wave and deploy it to an STM32. The example was put together by Pete Warden and Daniel Situnayake as part of their work at Google on TensorFlow Lite for Microcontrollers. This makes the job easier because they have put together a simple, public tutorial on capturing, labelling, and training the model. It can be found on Github here; once there, developers should click the 'Run in Google Colab' button. Google Colab, short for Google Colaboratory, allows developers to write and execute Python in their browser with zero configuration and provides free access to Google GPUs.

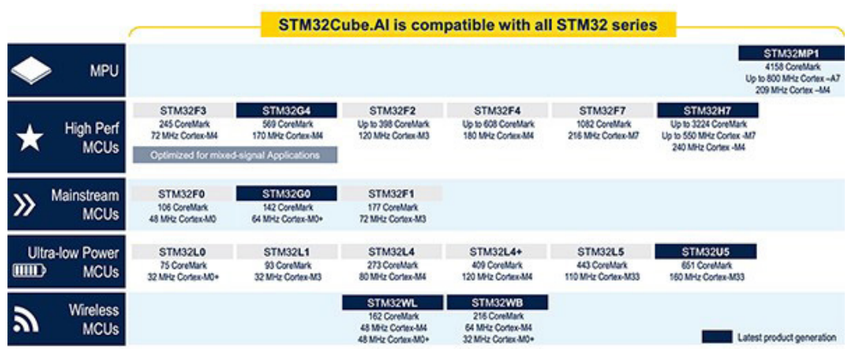


Figure 3. Shown are the microcontrollers and the microprocessor unit (MPU) currently supported by the STMicroelectronics AI ecosystem. Image source: STMicroelectronics

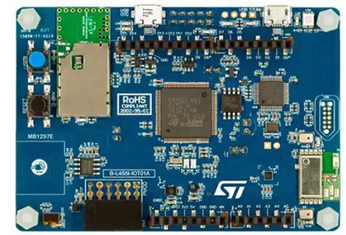


Figure 4. The STM32 B-L4S5I-IOT01A Discovery IoT Node is an adaptable experimentation platform for tinyML due to its onboard Arm Cortex-M4 processor, MEMS microphone, and three-axis accelerometer. Image source: STMicroelectronics

The output from walking through the training example will include two different model files; a model.tflite TensorFlow model that is quantized for microcontrollers and a model_no_quant.tflite model that is not quantized. The quantization indicates how the model activations and bias are stored numerically. The quantized version produces a smaller model that is more suited to a microcontroller. For those curious readers, the trained model results versus actual sine wave results can be seen in Figure 2. The output of the model is in red. The sine wave output isn't perfect, but it works well enough for a 'Hello World' program.

Selecting a development board

Before looking at how to convert the TensorFlow model to run on a microcontroller, a microcontroller needs to be selected for deployment in the model. This article will focus on STM32 microcontrollers because STMicroelectronics has many tinyML/ML tools that work well for converting and running models. In addition, STMicroelectronics has a wide variety of parts compatible with their ML tools (Figure 3).

If one of these boards are lying around the office, it's perfect for getting the 'Hello World' application up and running. However, for those interested in going beyond this example and getting into gesture control or keyword spotting, opt for the STM32 B-L4S5I-IOT01A Discovery IoT Node (Figure 4).

This board has an Arm Cortex-M4 processor based on the STM32L4+ series. The processor has 2 megabytes (Mbytes) of flash memory and 640 kilobytes (Kbytes) of RAM, providing plenty of space for tinyML models. The module is adaptable for tinyML use case experiments because it also has STMicroelectronics' MP34DT01 microelectromechanical systems (MEMS) microphone that can be used for keyword spotting application development. In addition, the onboard LIS3MDLTR three-axis accelerometer, also from STMicroelectronics, can be used for tinyML-based gesture detection.

Converting and running the TensorFlow Lite model using STM32Cube.AI

Armed with a development board that can be used to run the tinyML model, developers can now start to convert the TensorFlow Lite model into something that can run on the microcontroller. The TensorFlow Lite model can run directly on the microcontroller, but it needs a runtime environment to process it.

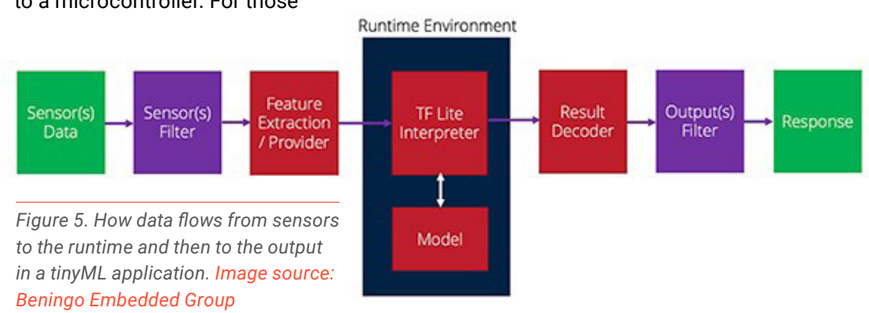
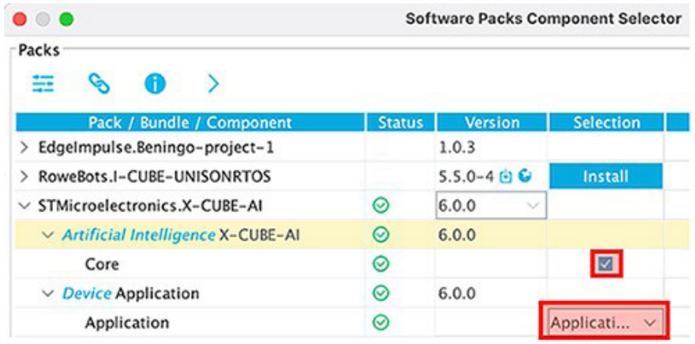


Figure 5. How data flows from sensors to the runtime and then to the output in a tinyML application. Image source: Beningo Embedded Group

When the model is run, a series of functions need to be performed. These functions start with collecting the sensor data, then filtering it, extracting

Figure 6. The X-CUBE-AI plug-in needs to be enabled using the application template for this example. Image source: Beningo Embedded Group



the necessary features, and feeding it to the model. The model will spit out a result which can then be further filtered, and then – usually – some action is taken. Figure 5 provides an overview of what this process looks like.

The X-CUBE-AI plug-in to STM32CubeMx provides the runtime environment to interpret the TensorFlow Lite model and offers alternative runtimes and conversion tools that developers can leverage. The X-CUBE-AI plug-in is not enabled by default in a project. However, after creating a new project and initializing the board, under Software Packs-> Select Components, there is an option to enable the AI runtime. There are several options here; make sure that the Application template is used for this example, as shown in Figure 6.

Once X-CUBE-AI is enabled, an STMicroelectronics X-CUBE-AI category will appear in the toolchain. Clicking on the category will give the developer the ability to select the model file they created and set the model parameters, as shown in Figure 7. An analyze button will also analyze the model and provide developers with RAM, ROM, and execution cycle information. It's highly recommended that developers compare the Keras and TFLite model options. On the sine wave model example, which is small, there won't be a huge difference, but it is noticeable. The project can then be generated by clicking 'Generate code'.

The code generator will initialize the project and build in the runtime environment for the tinyML model. However, by default, nothing is feeding the model. Developers need to add code to provide the model input values – x values – which the model will then interpret and use to generate the sine y values. A few pieces of code need to be added to the acquire_and_process_data and post_process functions, as shown in Figure 8.

At this point, the example is now ready to run. Note: add some printf statements to get the model output for quick verification. A fast compile and deployment results in the 'Hello World' tinyML model running. Pulling the model output

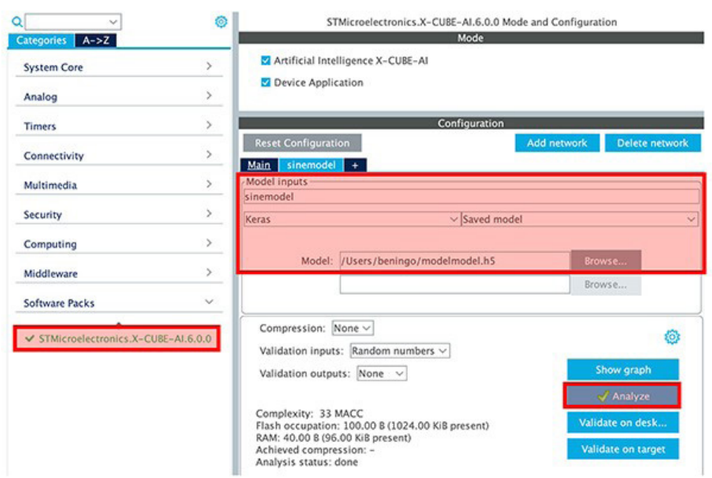


Figure 7. The analyze button will provide developers with RAM, ROM, and execution cycle information. Image source: Beningo Embedded Group


```
174  /* USER CODE BEGIN 2 */
175  int acquire_and_process_data(void * data)
176  {
177    static uint8_t position = 0;
178    uint8_t * Value = data;
179
180    * Value = position;
181
182    position++;
183
184    return 0;
185  }
```

```
187  int post_process(void * data)
188  {
189    uint8_t * Value = data;
190
191    if(*Value >= 128)
192    {
193        *Value -= 128;
194    }
195    else
196    {
197        *Value += 128;
198    }
199
200    return 0;
201  }
```

Figure 7. The analyze button will provide developers with RAM, ROM, and execution cycle information. Image source: Beningo Embedded Group

for a full cycle results in the sine wave shown in Figure 9. It's not perfect, but it is excellent for a first tinyML application. From here, developers could tie the output to a pulse width modulator (PWM) and generate the sine wave.

Tips and tricks for ML on embedded systems

Developers looking to get started with ML on microcontroller-based systems will have quite a bit on their plate to get their first tinyML application up and running. However, there are several 'tips and tricks' to keep in mind that can simplify and speed up their development:

- Walk through the TensorFlow Lite for microcontrollers 'Hello World' example, including the Google Colab file. Take some time to adjust parameters and understand how they affect the trained model
- Use quantized models for microcontroller applications. The quantized model is compressed to work with uint8_t rather than 32-bit floating-point numbers. As a result, the model will be smaller and execute faster
- Explore the additional examples in the TensorFlow Lite for Microcontrollers repository. Other examples include gesture detection and keyword detection
- Take the 'Hello World' example by connecting the model output to a PWM and a low-pass filter to see the resultant sine wave. Experiment with the runtime to increase and decrease the sine wave frequency
- Select a development board that includes 'extra' sensors that will allow for a wide range of ML applications to be tried
- As much fun as collecting data can be, it's generally easier to purchase or use an open-source database to train the model

Developers who follow these 'tips and tricks' will save quite a bit of time and grief when securing their application.

Conclusion

ML has come to the network Edge, and resource-constrained microcontroller-based systems are a prime target. The latest tools allow ML models to be converted and optimized to run on real-time systems. As shown, getting a model up and running on an STM32 development board is relatively easy, despite the complexities involved. While the discussion examined a simple model that generates a sine wave, far more complex models like gesture detection and keyword spotting are possible.

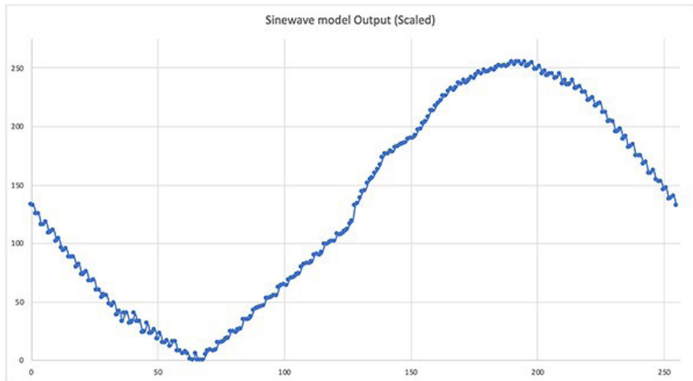


Figure 9. The 'Hello World' sine wave model output when running on the STM32. Image source: Beningo Embedded Group

How to use FPGA SoCs for secure and connected hard real-time systems

Written by: Jeff Shepard

Field programmable gate arrays (FPGAs), Linux-capable RISC-V microcontroller unit (MCU) subsystems, advanced memory architectures, and high-performance communications interfaces are important tools for designers. This is particularly true for designers of secure connected systems, safety-critical systems, and a wide range of hard real-time deterministic systems like artificial intelligence (AI) and machine learning (ML).

However, the integration of those diverse elements into a secure, connected, and deterministic system can be a challenging and time-consuming activity, as is laying out the high-speed interconnects for the various

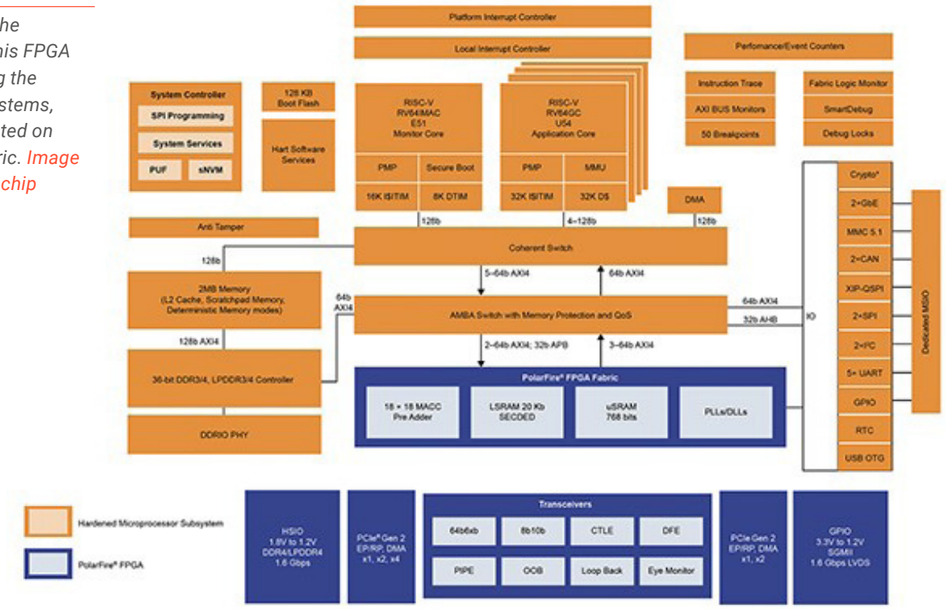


Figure 1. All the elements in this FPGA SoC, including the RISC-V subsystems, are implemented on the FPGA fabric. Image source: Microchip Technology

system elements. Designers need to include a memory management unit, memory protection unit, secure boot capability, and gigabit-class transceivers for high-speed connectivity. The design will need active and static power management and control of inrush currents. Some designs will require operation over the extended commercial temperature range of 0°C to +100°C junction temperature (TJ), while systems in industrial environments will need to operate with TJ from -40°C to +100°C.

To address these and other challenges, designers can turn to FPGA system-on-chip (SoC) devices that combine low power consumption, thermal efficiency, and defense-grade security for smart, connected, and deterministic systems.

This article reviews the architecture of such an FPGA SoC and how it supports the efficient design of connected and deterministic systems. It then briefly presents the EEMBC CoreMark-Pro processing power versus power consumption benchmark, along with a view of the benchmark performance of a representative FPGA SoC. It looks at how security is baked into these FPGA SoCs and details exemplary FPGA SoCs from Microchip Technology, along with a development platform to accelerate the design process. It closes with a brief listing of expansion boards from MikroElektronika that can be used to implement a range of communications interfaces, as well as global navigation satellite system (GNSS) location capability.

SoCs built with an FPGA fabric

The 'chip' for this SoC is an FPGA fabric that contains the system elements, from the FPGA to the RISC-V MCU subsystem that's built with hardened FPGA logic. The MCU subsystem includes a quad-core RISC-V MCU cluster, a RISC-V monitor core, a system controller, and a deterministic Level 2 (L2) memory subsystem. The FPGA in these SoCs includes up to 460 K logic elements, up to 12.7 gigabit per second (Gbps) transceivers, and other input/output (I/O) blocks, including general purpose I/O

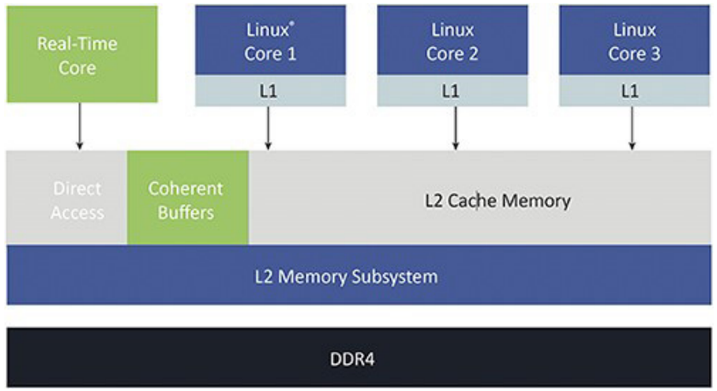


Figure 2. The RISC-V subsystem includes several processor and memory elements. Image source: Microchip Technology

(GPIO) and Peripheral Component Interconnect Express (PCIe) 2. The overall architecture is designed for reliability. It includes single-error correction and double-error detection (SECDED) on all memories, differential power analysis (DPA), physical memory protection, and 128 kilobits (Kbits) of flash boot memory (Figure 1).

Microchip offers its Mi-V (pronounced 'my five') ecosystem of third-party tools and design resources to support the implementation of RISC-V systems. It's built to speed the adoption of the RISC-V instruction set architecture (ISA) for hardened RISC-V cores and for RISC-V soft cores. Elements of the Mi-V ecosystem include access to:

- Intellectual property (IP) licenses
- Hardware
- Operating systems and middleware
- Debuggers, compilers, and design services

The hardened RISC-V MCUs in the FPGA SoC include several debugging capabilities like passive run-time configurable advanced extensible interface (AXI) and instruction trace. AXI enables designers to monitor data that's being written to or read from various memories and to know when it's being written or read.

The RISC-V MCU subsystem uses a five-stage single-issue, in-order pipeline. It's not vulnerable to Spectre or Meltdown exploits that can afflict out-of-order architectures. All five MCUs are coherent with the memory subsystem, supporting a mix of deterministic asymmetric multi-processing (AMP) mode real-time systems and Linux. Capabilities of the RISC-V subsystem include (Figure 2):

- Run Linux and hard real-time operations

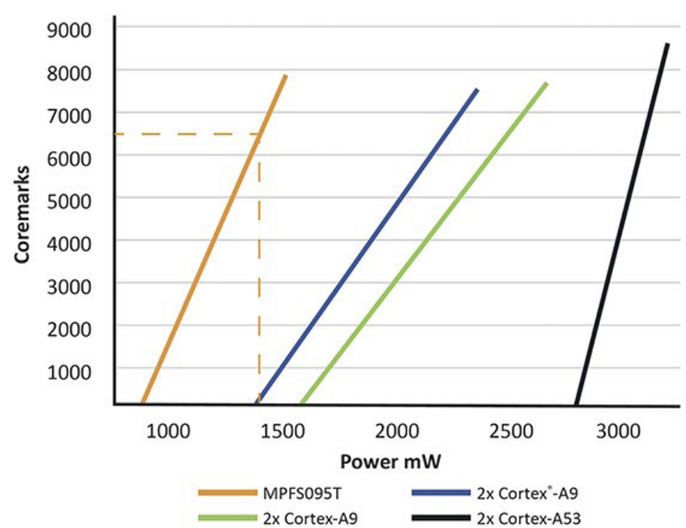


Figure 3. The MPFS095T FPGA SoC (orange line) delivers 6500 Coremarks at 1.3 watts. Image source: Microchip Technology

- Configure L1 and L2 as deterministic memory
- DDR4 memory subsystem
- Disable/enable branch predictors
- In-order pipeline operation

More processing with less energy

In addition to their system operation benefits, including support for hard, real-time processing, these FPGA SoCs are highly energy efficient. The EEMBC CoreMark-PRO benchmark is an industry standard for comparing the efficiency and performance of MCUs in embedded systems. It was designed specifically to benchmark hardware performance and to replace the Dhrystone benchmark.

The CoreMark-PRO workloads include a diversity of performance characteristics, instruction-level parallelism, and memory utilization based on four floating-point workloads and five common integer workloads. The floating-point workloads include a linear algebra routine derived from LINPACK, a fast Fourier transform, a neural net algorithm for pattern evaluation, and an improved version of the Livermore loops benchmark. JPEG compression, an XML parser, ZIP compression, and a 256-bit secure hash algorithm (SHA-256) form the basis of the integer workloads.

The MPFSO95T models of these SoC FPGAs, like the MPFS095TL-FCSG536E, can deliver up to 6,500 Coremarks at 1.3 watts (Figure 3).

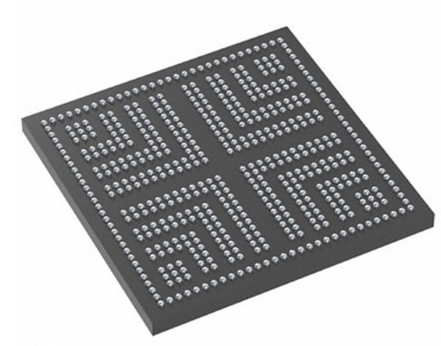


Figure 4. The automotive temperature MPFS250T-1FCSG536T2 comes in a 16 x 16mm package with a ball count of 536 and a 0.5mm pitch. Image source: Microchip Technology

Security considerations

The safety-critical and hard real-time applications for these FPGA SoCs require strong security in addition to high energy efficiency and powerful processing capabilities. The basic security functions of these FPGA SoCs include differential power analysis (DPA) resistant bitstream programming, a true random number generator (TRNG), and a physically unclonable function (PUF). They also include standard and user-defined secure boot, physical memory protection that provides memory access restrictions related to the machine's privilege state, including machine, supervisor, or user modes, and immunity from Meltdown and Spectre attacks.

Security begins with secure supply chain management, including the use of hardware security modules (HSMs) during wafer testing and packaging. The use of a 768-byte digitally signed x.509 FPGA certificate embedded in every FPGA SoC adds to supply chain assurance.

Numerous on-chip tamper detectors are included in these FPGA SoCs to ensure secure and reliable operation. If tampering is detected, a tamper flag is issued that enables the system to respond as needed. Some of the available tamper detectors include:

- Voltage monitors
- Temperature sensors
- Clock glitch and clock frequency detectors
- JTAG active detector
- Mesh active detector

Security is further ensured with 256-bit advanced encryption standard (AES-256) symmetric block cipher correlation power attack (CPA) countermeasures, integrated cryptographic digest capabilities to ensure data integrity, integrated PUF for key storage, and zeroization capabilities for the FPGA fabric and all on-chip memories.

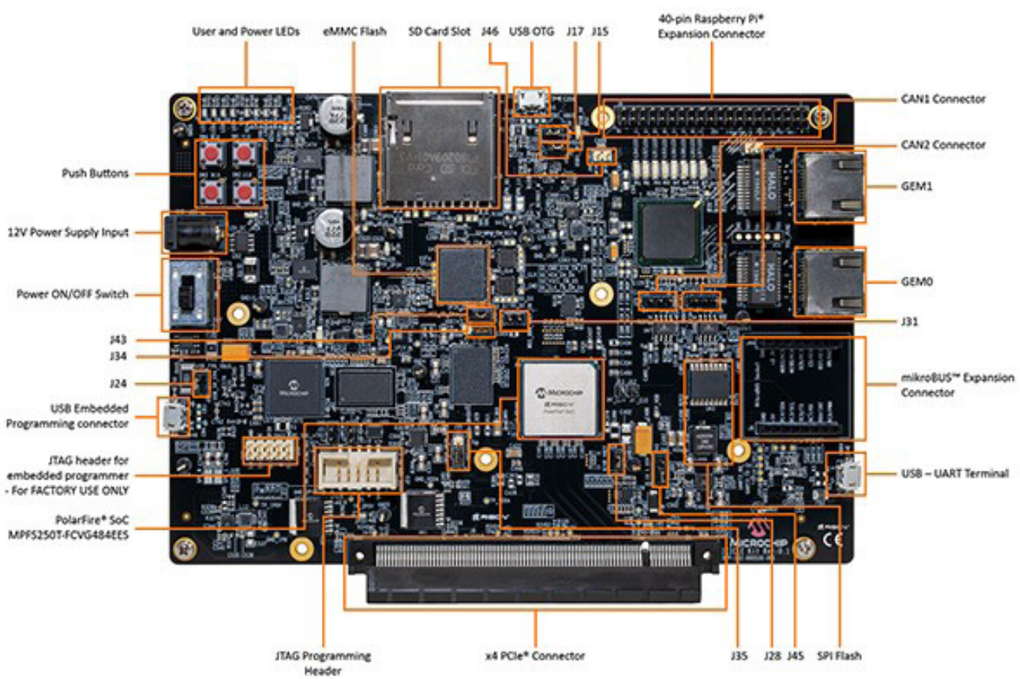
FPGA SoC examples

Microchip Technology combines these capabilities and technologies into its PolarFire FPGA SoCs with multiple speed grades, temperature ratings, and various package sizes to support designers' needs for a wide range of solutions with between 25 K and 460 K logic elements. Four temperature grades are available (all rated for TJ), 0°C to +100°C extended commercial range, -40°C to +100°C industrial range, -40°C to +125°C automotive range, and -55°C to +125°C military range.

Designers can choose from standard speed grade devices, or -1 speed grade devices that are 15% faster. These FPGA SoCs can be operated at 1.0 volt for lowest power operation, or at 1.05 volts for higher performance. They are available in a range of package sizes, including 11 x 11 millimeters (mm), 16 x 16 mm, and 19 x 19 mm.

For applications that need extended commercial temperature operation, standard speed operation, and 254 K logic elements in a 19 x 19mm package, designers can use the MPFS250T-FCVG484EES. For simpler solutions that need 23 K logic elements, designers can turn to the MPFS025T-FCVG484E, also with extended commercial temperature operation and standard speed grade in a 19 x 19 mm package. The MPFS250T-1FCSG536T2 with 254 K logic elements is designed for high-performance automotive systems and has an operating temperature range of -40 to 125°C and a -1 speed grade for a 15% faster clock, in a compact 16 x 16mm package with 536 balls on a 0.5mm pitch (Figure 4).

Figure 5. This comprehensive FPGA SoC development environment includes connectors for Raspberry Pi (top right) and mikroBUS (lower right side) expansion boards. Image source: Microchip Technology



FPGA SoC dev platform

To speed the design of systems with the PolarFire FPGA SoC, Microchip offers the MPFS-ICICLE-KIT-ES PolarFire SoC Icicle kit that enables exploration of the five-core Linux-capable RISC-V microprocessor subsystem with low-power, real-time execution. The kit includes a free Libero Silver license that's needed to evaluate designs. It supports programming and debugging features in a single language.

These FPGA SoCs are supported with the VectorBlox accelerator software development kit (SDK) that enables low-power, small-form-factor AI/ML applications. The emphasis is on simplifying the design process to the point that designers don't need to have prior FPGA design experience. The VectorBlox accelerator SDK enables developers to program power-efficient neural networks using C/C++. The Icicle kit has numerous features to provide a comprehensive development environment, including a multi-rail power sensor system to monitor the various power domains, PCIe root port, and on-board memories – including LPDDR4, QSPI, and eMMC Flash – to run Linux and Raspberry Pi, and mikroBUS expansion ports for a host of wired and wireless connectivity options, plus functional extensions like GNSS location capability (Figure 5).

Expansion boards

A few examples of mikroBUS expansion boards include:

MIKROE-986, for adding CAN bus connectivity using a serial peripheral interface (SPI).

MIKROE-1582, for interfacing between the MCU and an RS-232 bus.

MIKROE-989, for connecting with an RS422/485 communication bus.

MIKROE-3144, supports the LTE Cat M1 and NB1 technologies enabling reliable and simple connectivity with 3GPP IoT devices.

MIKROE-2670, enables GNSS functionality with concurrent reception of GPS and Galileo constellations plus either BeiDou or GLONASS, resulting in high position accuracy in situations with weak signals or interference in urban canyons.

Conclusion

Designers can turn to FPGA SoCs when developing connected, safety-critical and hard real-time deterministic systems. FPGA SoCs provide a wide range of system elements, including an FPGA fabric, RISC-V MCU subsystem with high-performance memories, high-speed communications interfaces, and numerous security functions. To help designers get started, development boards and environments are available that include all the necessary elements, including expansion boards that can be used to implement a wide range of communications and location functions.

Recommended reading

1. How to Implement Time Sensitive Networking to Ensure Deterministic Communication
2. Real-Time Operating Systems (RTOS) and Their Applications

Figure 1. Some attendees of the Summer Research Project. Back row, from left to right, Oliver Selfridge, Nathaniel Rochester, Marvin Minsky, and John McCarthy. In front, Ray Solomonoff, Peter Milner, and Claude Shannon.

Retro Electro: Programming a calculator to form concepts: the birth of artificial intelligence

The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can, in principle, be so precisely described that a machine can be made to simulate it.

Written by: David Ray, Cyber City Circuits

A proposal for The Dartmouth Summer Research Project on artificial intelligence

Since the earliest days of 'computers', it has taken thousands of people to bring 'artificial intelligence' and machine learning to where it is today.

In the Summer of 1955, Dr. John McCarthy started a new position as an assistant professor of Mathematics at Dartmouth College. Historians say McCarthy was the first to use the term 'Artificial Intelligence' in this proposal to the Rockefeller Foundation. Proposed and Organized by John McCarthy of Dartmouth College, Marvin Minsky of Harvard University, Nathaniel Rochester of IBM, and Claude E. Shannon of Bell Labs. The proposal was for a two-month, ten-man study on artificial intelligence. The aim was to gather many of the nation's top scientists, engineers, and mathematicians in the same room together to focus on what artificial intelligence could mean and how they could get there. They requested $13,500 to complete this study, but the Rockefeller Foundation only provided $7500 for a five-week study instead of two months.

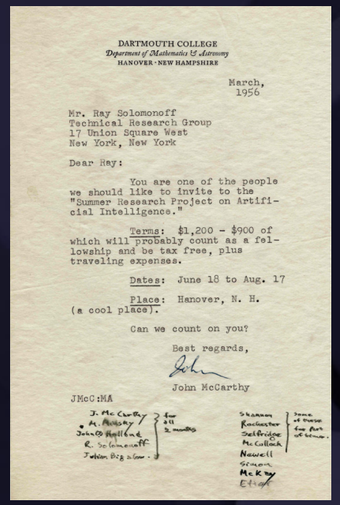
The group of four organizers were all highly distinguished researches and inventors. They were developing the fundamentals for today's generative AI, nearly seventy years ago.

The proposal outlines seven distinct parts of the problem.

Automatic computers

"If a machine can do a job, then an automatic calculator can be programmed to simulate the machine."

The idea was simple: if a machine could do a job, a computer could be



DARTMOUTH COLLEGE
Department of Mathematics & Astronomy
HANOVER · NEW HAMPSHIRE

March,
1956

Mr. Ray Solomonoff
Technical Research Group
17 Union Square West
New York, New York

Dear Ray:

You are one of the people we should like to invite to the "Summer Research Project on Artificial Intelligence."

Terms: $1,200 - $900 of which will probably count as a fellowship and be tax free, plus traveling expenses.

Dates: June 18 to Aug. 17

Place: Hanover, N. H. (a cool place).

Can we count on you?

Best regards,

John

John McCarthy

JMcC:MA

J. McCarthy
M. Minsky
John Nash
R. Solomonoff
Jerome Bigelow

} for all 2 months

Shannon
Rochester
Selfridge
McCulloch
Newell
Simon
McKay
Etzioni

} some of these for part of time.

Figure 2. Personal invitation to Dartmouth from McCarthy to Ray Solomonoff.

Figure 2. Claude Shannon with his self-solving 'mouse-in-a-maze' machine, Theseus.

programmed to replicate that task. Here, they admit that the speed and memory sizes of the machines they had at the time were 'insufficient' to simulate higher brain function. An issue they felt they could tackle is that there was no programming language available to do such a thing in the first place.

How can a computer be programmed to use a language

"It may be speculated that a large part of human thought consists of manipulating words according to rules of reasoning and rules of conjecture... This idea has never been very precisely formulated nor have examples been worked out."

Up to this point, the closest thing available for programming was assembly language. Here, the thought was that since much of thinking is really made up of words, grammar, and syntax, any thinking machine would likely need to operate in a similar way, governed by whitespace and syntax.

Neuron nets

"How can a set of (hypothetical) neurons be arranged so as to form concepts."

As scientists began to grapple with the challenge of mimicking human thought, they turned to the brain's fundamental building blocks: neurons. The question was how to arrange a set of hypothetical neurons to form concepts. Pioneers in the field had made strides in both theoretical and experimental work, but the problem remained far from solved.

Theory of the size of a calculation

"If we are given a well-defined problem, one way of solving it is to try all possible answers in order."

In their quest to solve complex problems, early computer scientists realized that brute-force methods were too time consuming. To address this, they sought to understand and measure how efficient a calculation could be.

Self-improvement

"Probably a truly intelligent machine will carry out activities which may best be described as self-improvement."

The vision of creating a truly intelligent machine led to a fascinating concept: self-improvement. Researchers speculated that for a machine to be intelligent, it would need the ability to enhance its own capabilities over time.

Abstraction

"A number of types of 'abstraction' can be distinctly defined and several others less distinctly. A direct attempt to classify these and to describe machine methods of forming abstractions from sensory and other data would seem worthwhile."

Abstraction, the ability to distill complex information into simpler concepts, was identified as a key process in human thought. To replicate this in machines, scientists needed to classify and define different types of abstraction. This task was seen as essential for enabling machines to interpret sensory data and other information in a human-like manner.

Randomness and creativity

"A fairly attractive and yet clearly incomplete conjecture is that the difference between creative thinking and unimaginative competent thinking lies in the injection of some randomness."

As researchers delved into the nature of creativity, they considered the role of randomness in the creative process. The intriguing idea emerged that the difference between routine and creative thinking might lie in the controlled injection of randomness. This theory suggested that when guided by intuition, randomness could be the secret ingredient that makes creative thinking possible.

> "We will concentrate on a problem of devising a way of programming a calculator to form concepts and to form generalizations. This, of course, is subject to change when the group gets together."

Proposal for research by C. E. Shannon

Claude Shannon's master's thesis, A Symbolic Analysis of Relay and Switching Circuits, is credited with introducing Boolean logic to electronic circuits and creating the digital age. After completing his doctorate at MIT, Shannon worked at Bell Labs, where he colaborated with and mentored McCarthy and Minsky in 1951 and 1952. Together, they developed 'Theseus', a self-solving 'mouse in a maze' using relay logic.

Shannon's research proposal for the Summer Research Project delved into two key areas related to information theory and brain models:

Application of information theory to computing machines and brain models

Shannon's first research focus addresses the challenge of reliably transmitting information across noisy channels using unreliable components. He explores how information flows in parallel data streams over closed-loop networks and examines the complications that may arise, such as propagation delays and redundancy. Shannon proposes investigating new approaches to minimize these delays, ensuring reliable transmission of information across complex systems.

The matched environment and brain model approach to Automata

In the second topic, Shannon theorizes that both animal and human brain development occurs in stages, beginning with simpler environments and eventually moving toward more complex ones. As someone gets older, the more their brain can comprehend the universe around them. He wanted to explore the specific stages of brain development and express them mathematically. By understanding how brains adapt to increasingly complex environments, Shannon hopes to build models replicating this adaptability in 'automata', ultimately advancing our understanding of mechanized intelligence.

Proposal for research by M. L. Minsky

As a graduate student, Marvin Minsky developed the first 'neural network' (The 'Stochastic Neural Analog Reinforcement Calculator' or 'SNARC') at Bell Labs in the early 1950s. A Navy Veteran, he had degrees from Harvard and Princeton. He founded MIT's Artificial Intelligence Lab and generally stayed there from its inception in 1963 until he died in 2016.

Minsky's proposal focused on designing a machine capable of learning through sensory and 'motor abstractions'. Minsky

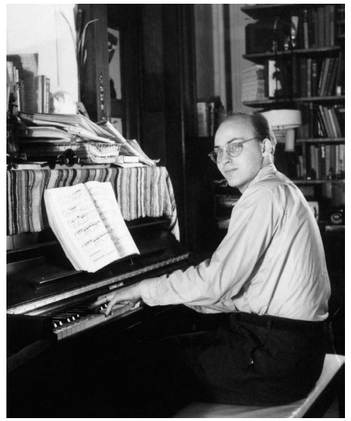


Figure 3. Marvin Minsky at Piano'

describes a machine that can be trained via a 'trial and error' process to perform specific tasks within an environment and exhibit 'goal-seeking' behavior. This hypothetical machine could process inputs, generate outputs, and adapt to success or failure by reading sensors and such, similar to Shannon's Theseus project, for which Minsky designed the SNARC. Minsky emphasizes the importance of pairing sensory and motor controls for the machine to affect and learn from its environment effectively.

Progress in the machine's learning would depend on its ability to relate environmental changes to corresponding changes in its sensor readings. Minsky further explains that the machine should develop an internal abstract model of its environment, stored in memory. This internal 'abstract' model would allow it to first experiment internally before conducting external tests, enabling it to perform tasks more intelligently. The machine's behavior would appear imaginative because it could predict and anticipate changes in the environment based on its motor actions.

Proposal for research by N. Rochester

Nathaniel Rochester worked at IBM at the time. He graduated from MIT in 1941 and then worked developing RADAR systems for the US Navy during the war. He started at IBM in 1948 after the wartime development dried up. A few years later, IBM released the first in the 700 series of electronic computers, the IBM 701, for which Rochester was the lead developer. At the time of the proposal, Rochester was the head of a research group studying information theory and automatic pattern recognition. McCarthy and Rochester first met when IBM gifted an IBM 704 to MIT's research lab, specifically for researching 'neural networks.'

Rochester's research proposal centers on the challenge of creating a machine capable of exhibiting originality in its problem-solving abilities. Typically, machines like automatic calculators are

Figure 4. Nathaniel Rochester designed the first electronic IBM computer.

programmed with a fixed set of rules to address specific contingencies and failures, leaving them without the flexibility to act intuitively or with common sense. For example, in your calculator, if you divide by '0', then you will likely get an error or some sort, but this is because the calculator was programmed to give an error when asked to divide by '0' instead of it learning on its own that dividing by '0' doesn't work and developing its own rules.

Rochester highlights the frustration from when machines fail due to rigid or contradictory rules and suggests that a more sophisticated approach is needed to enable machines to behave intelligently. Rochester draws on Kenneth Craik's model of human thought, which theorizes that the brain constructs 'engines' that simulate and predict outcomes in the environment.

He proposes that machines could similarly be designed to form abstractions of sensory data, define problems, and then simulate possible solutions, evaluating their success before acting. While this approach works for well-understood problems, Rochester notes that solving new or long-unsolved problems requires randomness and creativity. He argues that randomness could be key to overcoming the limitations of pre-programmed rules and enabling machines to behave in original ways, much like how scientists may rely on a 'hunch' to approach difficult problems.

Rochester discusses the Monte Carlo method, which involves conducting hundreds or thousands of random experiments to approximate solutions to complex problems. He sees potential in applying this method to machine learning, suggesting that machines could explore many possibilities simultaneously and uncover solutions that traditional methods might miss.

However, he acknowledges that simulating human-like randomness in machines is challenging, as the brain's control mechanisms differ significantly from those of calculators and computers.

Figure 5. John McCarthy while working with chess computers.

Proposal for research by John McCarthy

John McCarthy, an army veteran, is famously known for coining the term 'Artificial Intelligence.' Following his doctorate at Princeton, he took a few assistant professor positions in the area, landing at Dartmouth College in the summer of 1955. As a graduate student, he interned with Marvin Minsky at Bell Labs, where he was mentored by Claude Shannon. Following the Summer Research Project, however, he took a position at MIT with Marvin Minsky, continuing work in AI and developing the LISP programming language.

McCarthy's proposal focuses on studying the relationship between language and intelligence. It argues that direct applications of trial-and-error methods to the interaction between sensory data and motor activity are unlikely to result in complex behaviors. Instead, he advocates for applying trial and error at a higher level of abstraction.

He highlights language as a crucial tool people use to handle intricate phenomena, noting that human minds use language to formulate conjectures and test them. McCarthy points out that English has several advantageous properties for facilitating complex thought processes, properties that programming languages developed for computers often lack.

These properties include the ability to use concise arguments that can be supplemented by informal mathematics, a way of incorporating other languages within English, and the ability for users to reference their own problem-solving progress. He also

"Unless the machine is provided with, or is able to develop, a way of abstracting sensory material, it can progress through a complicated environment only through painfully slow steps, and in general will not reach a high level of behavior."

- Minsky

"So the mathematician has the machine making a few thousand random experiments … the results of these experiments provide a rough guess as to what the answer may be."

– Rochester

suggests that a fully formalized version of English would include rules not just for proofs but also for guesses and conjectures.

McCarthy contrasts this idea against the logical languages of the day, which were mainly used for creating pre-determined instruction lists or formalizing parts of mathematics. He proposes that an artificial language be developed to handle conjecture and self-reference effectively. This language would mirror English in that short statements in English would have equally concise counterparts in the artificial language.

McCarthy's eventual goal is to create a language that would allow a machine to engage in tasks such as learning to play games, with the potential to handle more advanced problem-solving tasks.

The impact of the Summer Research Project

The Dartmouth Summer Research Project on Artificial Intelligence took place the following summer and is a milestone moment in the history of AI. Throughout the conference eleven people attended.

While there were many 'projects' and 'conferences' concerning machine intelligence, this event marks the largest concerted effort to achieve Artificial Intelligence at the time.

While the goal was missed and the problem turned out harder than they thought, its results continue to inform machine learning and artificial intelligence development seventy years later.

The writer is thankful to Grace Solomonoff for her help archiving so much of the Dartmouth event, that otherwise would have been lost to time.

References

1. A Proposal For the Dartmouth Summer Research Project on Artificial Intelligence
2. Ray Solomonoff's Personal 'Dartmouth Archives'
3. 'The Meeting of the Minds That Launched AI' by Grace Solomonoff
4. The Turbulent Past and Uncertain Future of Artificial Intelligence by Eliza Strickland
5. 'Oral History of Nathaniel Rochester' Interview by A. Goldstein (June 1991)
6. 'Programs With Common Sense' by J. McCarthy
7. 'A Symbolic Analysis of Relay and Switching Circuits' by C.E. Shannon
8. 'Claude E. Shannon: A Retrospective on His Life, Work, and Impact' by R.G. Gallager
9. 'Claude Shannon – Father of the Information Age' from the University of California
10. 'Mouse With a Memory' by Bell Labs
11. 'The Pioneers of AI: Marvin Minsky and the SNARC' by Sahid Parvez
12. (Video) Claude Shannon demonstrates "Theseus" Machine Learning @ Bell Labs
13. (Video) Marvin Minsky Interview Series (Life Stories of Remarkable People)

"(The) main reason the 1956 Dartmouth workshop did not live up to my expectations is that AI is harder than we thought."

- Marvin Minsky

1938
Claude Shannon publishes his thesis 'A Symbolic Analysis of Relay and Switching Circuits,' introducing Boolean logic to electronic circuits.

1941
Nathaniel Rochester graduates from MIT.

1944
John McCarthy is drafted into the U.S. Army. Marvin Minsky joins the U.S. Navy to learn radio and electronics.

1946
ENIAC, the first electronic "general purpose" computer, begins operation, marking a significant advancement in digital computing.

1948
Rochester begins work at IBM.

1950
Alan Turing publishes Computing Machinery and Intelligence, proposing the Turing Test.

1951
McCarthy and Minsky intern at Bell Labs, mentored by Claude Shannon. Minsky develops the first neural network, SNARC.

1952
IBM releases the IBM 701, designed by Rochester and Haddad. Shannon demonstrates 'Theseus' at Bell Labs.

1955
McCarthy joins Dartmouth College as an assistant professor. McCarthy submits a proposal to the Rockefeller Foundation, coining the term "Artificial Intelligence."

1956
The Dartmouth Summer Research Project on Artificial Intelligence takes place.

1958
McCarthy develops the LISP programming language, which becomes the standard for AI development.

1963
MIT's AI Lab is founded by McCarthy and Minsky, funded by ARPA, becoming a hub for AI research.

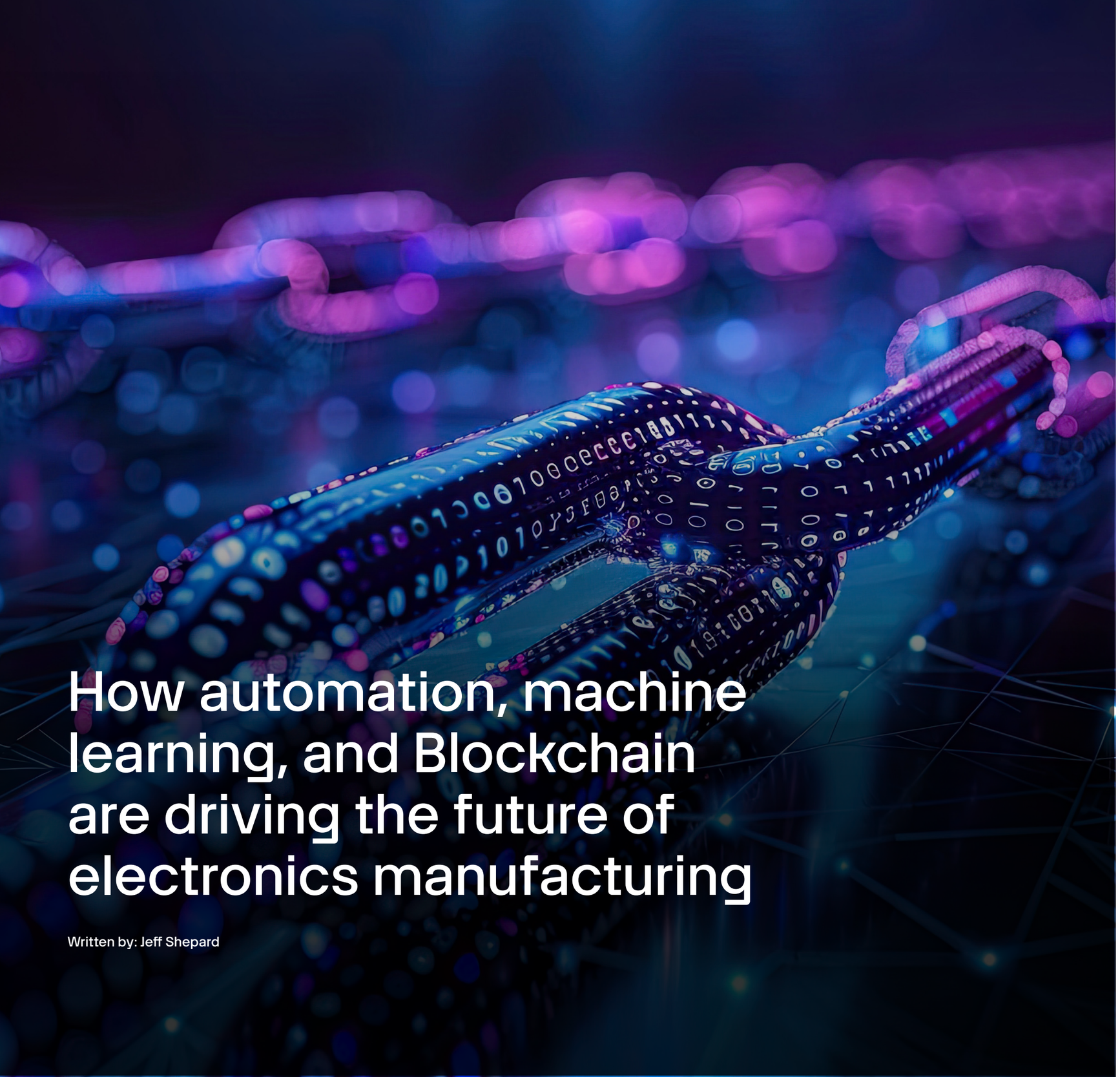
1965
Moore's Law is proposed, predicting long-term exponential growth in computing power.

1966
ELIZA, the first chatbot, is developed by Joseph Weizenbaum, pioneering early natural language processing.

1997
IBM's Deep Blue defeats world chess champion Garry Kasparov, a landmark achievement in AI history.

2015
OpenAI is founded.

2022
OpenAI releases ChatGPT to the public.



How automation, machine learning, and Blockchain are driving the future of electronics manufacturing

Written by: Jeff Shepard

Industry 4.0 relies on intelligent automation for manufacturing electronics. Increasingly capable automation is everywhere, from the edge to the cloud, in sensors, robots and cobots, programmable logic controllers (PLCs), and other equipment. Semiconductor wafers, integrated circuits, passive components, packaging, and electronic systems for consumer, green energy, automotive, medical, industrial, military/aerospace, and other applications depend on intelligent automation for their production. Unified manufacturing execution systems (MES) provide real-time monitoring, control, tracking, and documentation of the entire manufacturing chain, from raw materials to finished goods.

The cyber-physical automated systems in Industry 4.0 extend beyond traditional manufacturing activities and rely on various forms of machine learning (ML) ranging from deep reinforcement learning in the Cloud to tinyML on the Edge for flexible production, continuous improvement, and consistently high quality. The number of layers of connectivity is growing, and the combination of Edge computing, the Industrial Internet of Things (IIoT), and Cloud computing is increasing the challenges related to cyber security. Blockchain has recently entered the picture for comprehensive and secure supply chain management.

This article looks at key automation trends in electronics manufacturing, including the increasing layers of connectivity, the growing need for cybersecurity, the specialized implementations of ML being deployed, and how traceability and MES support real-time production metrics and analytics. Along the way, some of the technologies needed to fully realize the promise of Industry 4.0 for mass customization with high quality and low costs are reviewed, including how DigiKey supports the needs of automation system designers with a wide range of solutions. It closes with a look at how blockchain is used to deploy highly secure enterprise-wide supply chain management systems.

Increasing layers of connectivity

The IIoT in Industry 4.0 includes more wired and wireless network layers for sensor networks, autonomous mobile robots (AMRs), and other systems. For example, IO-Link was developed to provide a simplified wired network connection for the massive number of sensors, actuators, indicators, and other previously unconnected legacy edge devices to higher-level networks like Ethernet IP, Modbus TCP/IP, and PROFINET. With IO-Link, the inputs and outputs (IOs) of these devices are captured and

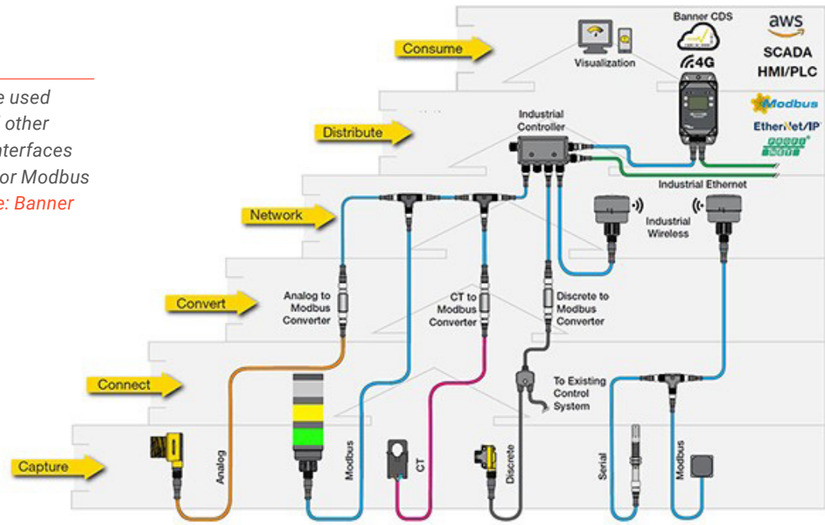


Figure 1. IO-Link can be used to connect sensors and other devices using diverse interfaces to Ethernet, PROFINET, or Modbus networks. Image source: Banner Engineering

converted to the IO-Link protocol for serial connectivity defined in IEC 61131-9 with a single 4- or 5-wire unshielded cable defined in IEC 60974-5-2 (Figure 1). In addition to providing a new networking layer to capture more granular information about factory processes, IO-Link supports rapid deployment and remote configuration, monitoring, and diagnostics of connected devices to support line and process changes needed for mass customization in Industry 4.0 factories.

Wireless IIoT devices, from sensors to robots, also contribute to the growing networking layers. Various wireless protocols, including Wi-Fi, 5G, LTE, and others, are used in modern factories. For example, AMRs use a combination of onboard sensors and Wi-Fi connectivity to understand their environment, identify possible obstacles and move safely and efficiently from place to place. Colaborative robots (cobots) are designed to work with people to improve operational efficiency and often require wireless connectivity. In some cases, AMRs move cobots from task to task as needed (Figure 2).

Increasing cyber dangers

The increasing layers in industrial networks, combined with the explosion in the number of connected devices, are resulting in a growing number of security threat vectors and increasing cyber dangers. Several industrial and IoT-specific security standards and methodologies have been developed, including International Electrotechnical Commission (IEC) 62443 and the Security Evaluation Standard for IoT Platform (SESIP).

IEC 62443 is a series of standards developed by the International Society of Automation (ISA) 99 committee and approved by the

Figure 2. An AMR (bottom) can navigate from place to place using a combination of onboard sensors and wireless connectivity and pick up and move a cubit (top) to a new workstation. Image source: Omron

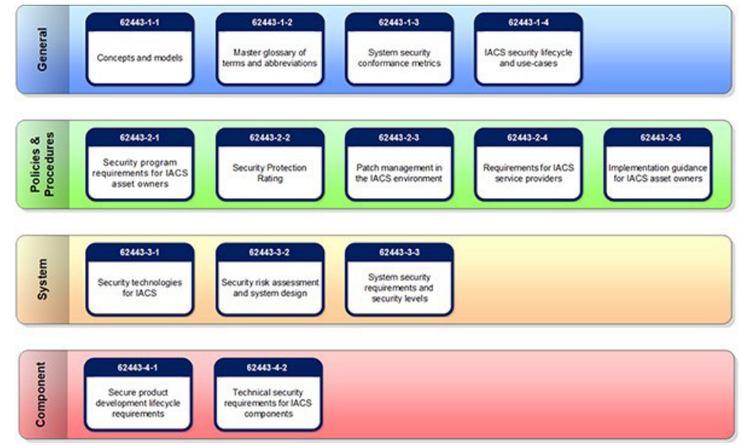


Figure 3. IEC 62443 is a comprehensive set of IACS security standards. Image source: IEC

IEC. IEC 62443 is an 800-plus-page series of standards for Industrial Automation and Control Systems (IACS) in 14 subsections and four tiers (Figure 3). Key sections that define the product development and security requirements for components are:

- IEC 62443-4-1: Product Security Development Lifecycle Requirements – defines a secure product development lifecycle including initial requirements definition, secure design and implementation, verification and validation, defect and patch management, and end-of-life.
- IEC 62443-4-2: Security for Industrial Automation and Control Systems: Technical Security Requirements for IACS Components – specifies security capabilities that enable a component to mitigate threats for a given security level.

SESIP is published by the GlobalPlatform and defines a common structure for evaluating the security of connected products and addresses IoT-specific compliance, security, privacy, and scalability challenges. SESIP provides clear definitions of security functionality on components and platforms in the form of Security Functional Requirements (SFRs). It also provides strength metrics that measure robustness against attacks in the form of SESIP 'levels' from 1 to 5, with 1 being self-certification and 5 corresponding to extensive testing and third-party certification.

ML from the Cloud to the Edge

ML is a key enabler of intelligent automation, supporting continuous process improvements and high-quality products. The use of neural networks is a well-established ML technique in Industry 4.0. It's beginning to be supplemented with deep reinforcement learning in the Cloud. Deep reinforcement learning adds a framework of goal-oriented algorithms to a neural network core. Initially, reinforcement learning was confined to repeatable environments like playing games; today, algorithms can operate in more ambiguous environments in the real world. In the future, advanced reinforcement learning implementations may achieve artificial general intelligence.

ML is not just in the Cloud; it's reaching onto the factory floor to the Edge. The expansion slots in industrial PCs and programmable controllers on the factory floor increasingly host ML and AI accelerator cards for intelligent process control.

Tiny machine learning (tinyML) is optimized for deployment in low-power applications. The use of tinyML in sensor applications

Figure 4. Arduino's Tiny Machine Learning Kit is designed for developing IIoT sensor applications. Image source: Digi-Key

is growing rapidly. One example of a tinyML application is IIoT sensor analytics in Edge devices powered by batteries or energy harvesting. Arduino offers a Tiny Machine Learning Kit that includes an Arduino Nano 33 BLE Sense board containing an MCU and a variety of sensors that can monitor movement, acceleration, rotation, sounds, gestures, proximity, color, light intensity, and movement (Figure 4). An OV7675 camera module and an Arduino shield are also included. The onboard MCU can implement deep neural networks based on the TensorFlow Lite open-source deep learning framework for on-device inference.

Real-time metrics and analytics

Real-time metrics and analytics are essential aspects of intelligent automation. Traceability 4.0 combines product visibility, supply chain visibility, and line-item visibility from previous generations of traceability and provides a complete history of all aspects of a product. In addition, it includes all machine and process parameters and supports overall equipment effectiveness (OEE) metrics that optimize manufacturing processes (Figure 5).

Traceability is vital in many

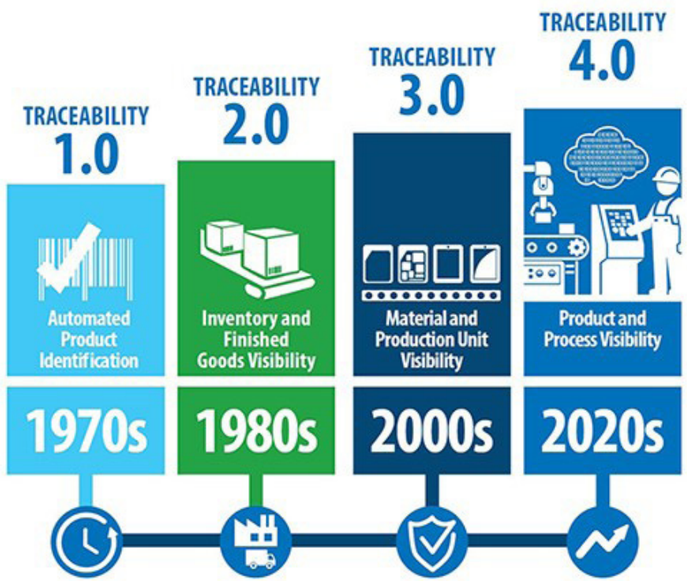


Figure 5. Traceability 4.0 is a comprehensive implementation that supports the diverse requirements of Industry 4.0 operations. Image source: Omron

industries, from medical device manufacturing to automotive and aerospace. In the case of medical devices, regulatory requirements demand extensive tracking and traceability. Automobiles and aerospace systems can have tens of thousands of parts to track. It's not just part history; traceability includes tracking individual part geometric dimensioning and tolerancing (GD&T). GD&T enables precision manufacturing and the installation of parts based on their exact GD&T values, supporting high-precision assemblies for industries like aerospace and automotive manufacturing.

Traceability can improve the accuracy and efficiency of implementing product recalls. It enables the manufacturer to identify all the affected products and the supplier or suppliers of any defective components.

Corrective and preventative actions can be accelerated through the use of traceability. Like product recalls, knowing the complete provenance of products enables manufacturers to efficiently target and schedule service and maintenance activities for products in the field.

Traceability and MES

Unified MES implementations

The intelligent automation that's the foundation of Industry 4.0 relies on numerous technologies for its implementation, including a growing number of network layers with wired and wireless connectivity that result in increasingly complex cyber security threats.

incorporating traceability can produce a searchable database of all the information related to individual products, including as-planned designs and as-built results. For example, traceability is used to track individual components and materials as they arrive, including inbound quality testing data, location of the supplying factory, and so on, before production starts. MES verifies that information based on the planned design and feeds into kitting operations and work in process databases.

Traceability data supplied by the IIoT combined with MES supports the mass customization of products in Industry 4.0. MES enables the right materials, processes, and other resources to be at the right place to ensure the lowest production cost and highest quality result. Also, MES and traceability can combine and demonstrate compliance with government regulations and make the data readily accessible to auditors or others as required.

Blockchain

A Blockchain is a decentralized, or distributed, digital ledger system for recording transactions between multiple parties in a tamperproof and verifiable manner. Any transactions where trust is important, like supply chain management, are potential uses for blockchain. In a supply chain with many participants, Blockchain can improve transaction efficiency and make transactions verifiable and tamperproof. Two examples of the benefits of using Blockchain in supply chain activities include:

Replacement of manual processes.

Manual paper-based processes that rely on signatures or other forms of physical verification can potentially be improved using Blockchain. The limitation is that the universe of participants in the ledger must be finite and easily identifiable. A delivery company with a constantly changing database of unfamiliar customers may not be a good candidate for Blockchain. A manufacturing operation with a finite and slowly changing group of trusted suppliers is a good candidate.

Strengthening traceability.

Blockchain can provide a good tool for improving supply chain transparency and meeting growing regulatory and consumer information requirements. For example, the Blockchain can support the Drug Supply Chain and Security Act and the unique device identifier mandate from the U.S. Food and Drug Administration. In the automotive and other industries, suppliers throughout the supply chain can be involved in implementation of recalls, and Blockchain can provide a good tool for implementing the Traceability Guideline published by the Automotive Industry Action Group.

Summary

The intelligent automation that's the foundation of Industry 4.0 relies on numerous technologies for its implementation, including a growing number of network layers with wired and wireless connectivity that result in increasingly complex cyber security threats. In addition, machine learning is being implemented from the edge to the Cloud to support real-time metrics and analytics, including traceability and unified MES. Finally, Blockchain technology is being introduced to support tamperproof and verifiable databases.

Quickly implement spoofing-resistant face recognition without a Cloud connection

Written by: Stephen Evanczuk

Face recognition has gained widespread acceptance for authenticating access to smartphones but attempts to apply this technology more broadly have fallen short in other areas despite its effectiveness and ease of use. Along with the technical challenges of implementing reliable, low-cost machine learning solutions, developers must address user concerns around the reliability and privacy of conventional face recognition methods that depend on cloud connections that are vulnerable to spoofing.

This article discusses the difficulty of secure authentication before introducing a hardware and software solution from NXP Semiconductors that addresses the issues. It then shows how developers without prior experience in machine learning methods can use the solution to rapidly implement offline anti-spoofing face recognition in a smart product.

The challenges of secure authentication for smart products

In addressing growing concerns about the security of smart products, developers have found themselves left with few tenable options for reliably authenticating users looking for quick yet secure access. Traditional methods rely on multifactor authentication methods that rest on some combination of the classical three factors of authentication: “Something you know”, such as a password; “Something you have”, such as a physical key or key card; and “Something you are”, which is typically a biometric factor such as a fingerprint or iris. Using this approach, a strongly authenticated door lock might require the user to enter a passcode, use a key card, and further provide a fingerprint to unlock the door. In practice, such stringent requirements are bothersome or simply impractical for consumers who need to frequently and easily re-authenticate themselves with a smartphone or other routinely used device.

The use of face recognition has significantly simplified authentication for smartphone users, but smartphones possess some advantages that might not be available in every device. Besides the significant processing power available in leading-edge smartphones, always-on connectivity is a fundamental requirement for delivering the sophisticated range of services routinely expected by their users.

For many products that require secure authentication, the underlying operating platform will typically provide more modest computing resources and more limited connectivity. Face recognition services from the leading cloud-service providers shift the processing load to the cloud, but the need for robust connectivity to ensure minimal response latency might impose requirements that remain beyond the capabilities of the platform. Of equal or more concern to users, transmitting their image across public networks for processing and potentially storing it in the cloud raises significant privacy issues.

Using NXP Semiconductors’ i.MX RT106F processors and associated software, developers can now implement offline face recognition that directly addresses these concerns.

Hardware and software for spoof-proof offline face recognition

A member of the NXP i.MX RT1060 Crossover microcontroller (MCU) family, the NXP i.MX RT106F series is specifically designed to support easy integration of offline face recognition into smart home devices, consumer appliances, security devices, and industrial equipment. Based on an Arm Cortex-M7 processor core, the processors run at 528 megahertz (MHz) for the industrial grade MIMXRT106FCVL5B, or 600MHz for commercial grade processors such as the MIMXRT106FDVL6A and MIMXRT106FDVL6B.

Besides supporting a wide range of external memory interfaces, i.MX RT106F processors include 1 megabyte (Mbyte) of on-chip random access memory (RAM) with 512 kilobytes (Kbyte)

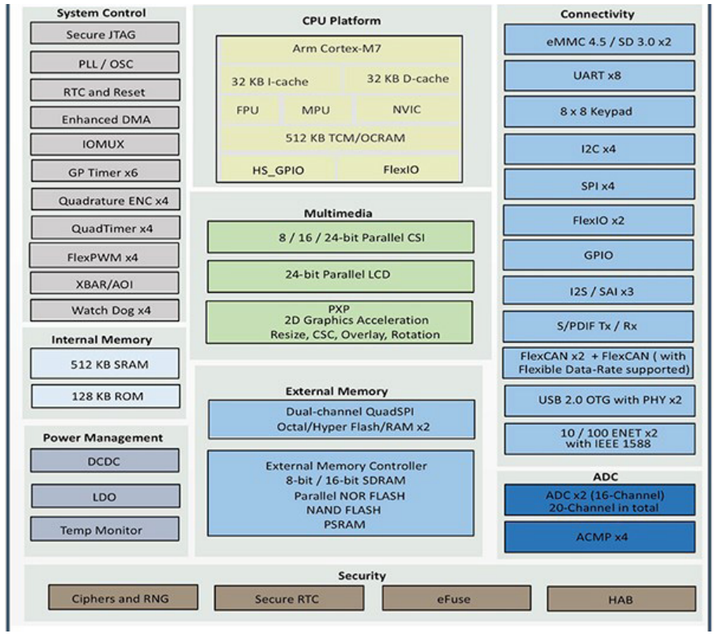


Figure 1. NXP Semiconductor's i.MX RT106F processors combine a full set of functional blocks needed to support face recognition for consumer, industrial and security products. Image source: NXP

configured as general purpose RAM, and 512 Kbytes that can be configured either as general purpose RAM or as tightly coupled memory (TCM) for instructions (I-TCM) or data (D-TCM). Along with on-chip power management, these processors offer an extensive set of integrated features for graphics, security, system control, and both analog and digital interfaces typically needed to support consumer devices, industrial human machine interfaces (HMIs), and motor control (Figure 1).

Although similar to other i.MX RT1060 family members, i.MX RT106F processors bundle in a runtime license for NXP's Oasis Lite face recognition software. Designed to speed inference on this class of processors, the Oasis Lite runtime environment performs face detection, recognition, and even limited emotion classification using neural network (NN) inference models running on an inference engine and MiniCV – a stripped-down version of the open source OpenCV computer vision library. The inference engine builds on an NXP NN library and the Arm Cortex Microcontroller System Interface Standard NN (CMSIS-NN) library (Figure 2).

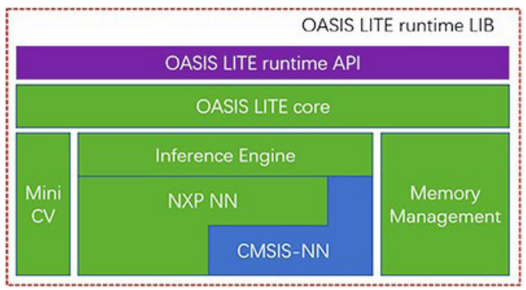


Figure 2. The NXP Oasis Lite runtime library includes an Oasis Lite core that uses MiniCV and an NXP inference engine built on neural network libraries from NXP and Arm. Image source: NXP

The inference models reside on the i.MX RT106F platform, so face detection and recognition execute locally, unlike other solutions that depend on Cloud-based resources to run the machine learning algorithms. Thanks to this offline face recognition capability, designers of smart products can ensure private, secure authentication despite low bandwidth or spotty Internet connectivity. Furthermore, authentication occurs quickly with this hardware and software combination, requiring less than 800 milliseconds (ms) for the processor to wake from low-power standby and complete face recognition.

Used with the i.MX RT106F processor, the Oasis Lite runtime simplifies implementation of offline face recognition for smart products, but the processor and runtime environment are of course only part of a required system solution. Along with a more complete set of system components, an effective authentication solution requires imaging capability that can mitigate a type of security threat called presentation attacks. These attacks attempt to spoof face recognition authentication by using photographs. For developers looking to rapidly deploy face-based authentication in their own products, the NXP SLN-VIZNAS-IOT development kit and associated software provide a ready-to-use platform for evaluation, prototyping and development of offline, anti-spoofing face recognition.

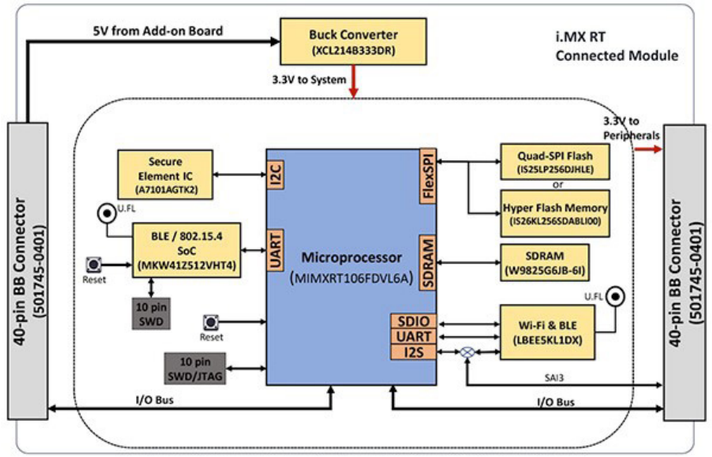


Figure 3. The NXP SLN-VIZNAS-IOT kit includes a connected module that provides a robust connected system platform needed to run authentication software. Image source: NXP

Complete secure systems solution for face recognition

As with most advanced processors, the i.MX RT106F processor requires only a few additional components to provide an effective computing platform. The NXP SLN-VIZNAS-IOT kit completes the design by integrating the i.MX RT106F with additional devices to provide a complete hardware platform (Figure 3).

The kit's connected module board combines an NXP MIMXRT106FDVL6A i.MX RT106F processor, an NXP A71CH secure element, and two connectivity options – NXP's MKW41Z512VHT4 Kinetis KW41Z Bluetooth low energy (BLE) system-on-chip (SoC) and Murata Electronics' LBEE5KL1DX-883 Wi-Fi/Bluetooth module.

To supplement the processor's on-chip memory, the connected module adds Winbond Electronics' W9825G6JB 256 megabit (Mbit) synchronous dynamic RAM (SDRAM), an Integrated Silicon Solution. Inc. (ISSI) IS26KL256S-DABLI00 256 Mbit NOR flash, and ISSI's IS25LP256D 256 Mbit Quad Serial Peripheral Interface (SPI) device.

Finally, the module adds a Torex Semiconductor XCL214B333DR

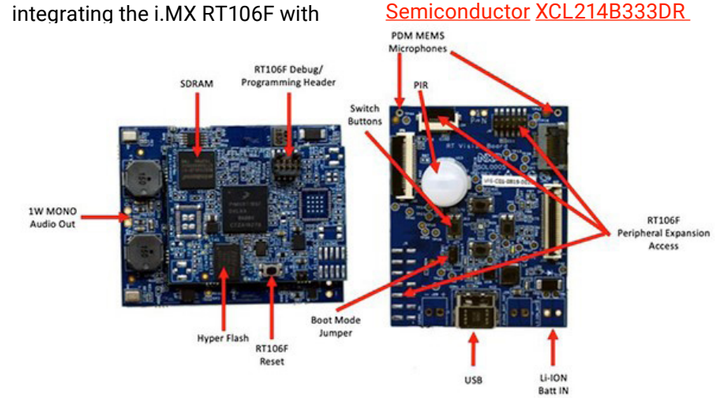


Figure 4. In the NXP SLN-VIZNAS-IOT kit, the connected module (left) is attached to the vision application board to provide the hardware foundation for face recognition. Image source: NXP

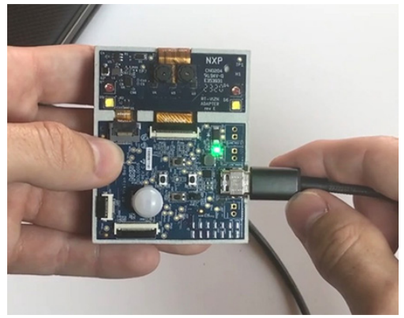


Figure 5. The NXP SLN-VIZNAS-IOT hardware kit integrates a dual camera system for liveness detection (top) and a vision application board (bottom) with a connected module to provide a drop-in solution for offline face recognition with anti-spoofing capability. Image source: NXP

buck converter to supplement the i.MX RT106F processor's internal power management capabilities for the additional devices on the connected module board.

The connected module in turn mounts on a vision application board that combines a Murata Electronics IRA-S210ST01 passive infrared (PIR) sensor, motion sensor, battery charger, audio support, light emitting diodes (LEDs), buttons, and interface connectors (Figure 4).

Figure 4. In the NXP SLN-VIZNAS-IOT kit, the connected module (left) is attached to the vision application board to provide the hardware foundation for face recognition. (Image source: NXP)

Along with this system platform, a face recognition system design clearly requires a suitable camera sensor to capture an image of the user's face. As mentioned earlier, however, concerns about presentation attacks require additional imaging capabilities.

Mitigating presentation attacks

Researchers have for years explored different presentation attack detection (PAD) methods designed to mitigate attempts such as using latent fingerprints or images of a face to spoof biometric-based authentication systems. Although the details are well beyond the scope of this article, PAD methods in general use deep analysis of the quality and characteristics of the biometric data captured as part of the process, as well as 'liveness' detection methods designed to determine if the biometric data was captured from a live person. Underlying many of these different methods, deep neural network (DNN) models play an important role not only in face recognition, but also in identifying attempts to spoof the system. Nevertheless, the imaging system used to capture the user's face can provide additional liveness detection support.

For the SLN-VIZNAS-IOT kit, NXP includes camera modules that contain a pair of ON Semiconductor's MT9M114 image sensors. Here, one camera is equipped with a red, green, blue (RGB) filter, and the other camera is fitted with an infrared (IR) filter. Attached through camera interfaces to the vision application board, the RGB camera generates a normal visible light image, while the IR camera captures an image that would be different for a live person compared to an image of the person. Using this liveness detection approach along with its internal face recognition capability, the SLN-VIZNAS-IOT kit provides offline, anti-spoofing face recognition capability in a package measuring about 30 x 40 millimeters (mm) (Figure 5).

Getting started with the SLN-VIZNAS-IOT kit

The NXP SLN-VIZNAS-IOT kit comes ready-to-use with built-in face recognition models. Developers plug in a USB cable and touch a button on the kit to perform a simple manual face registration using the preloaded 'elock' application and the accompanying mobile app (Figure 6, left). After registration, the mobile app will display a 'welcome home' message and 'unlocked' label when the kit authenticates the registered face (Figure 6, right).

The kit's Oasis Lite face recognition software processes models from its database of up to 3000 RGB faces with a recognition accuracy of 99.6%, and up to 100 IR faces with an anti-spoofing accuracy of 96.5%. As noted earlier, the NXP hardware/software solution needs less than one second (s) to perform face detection, image alignment, quality check, liveness detection, and recognition over a range from 0.2 to 1.0 meters (m). In fact, the system supports an alternate 'light' inference model capable of performing this same sequence in less than 0.5 s but supports a smaller maximum database size of 1000 RGB faces and 50 IR faces.

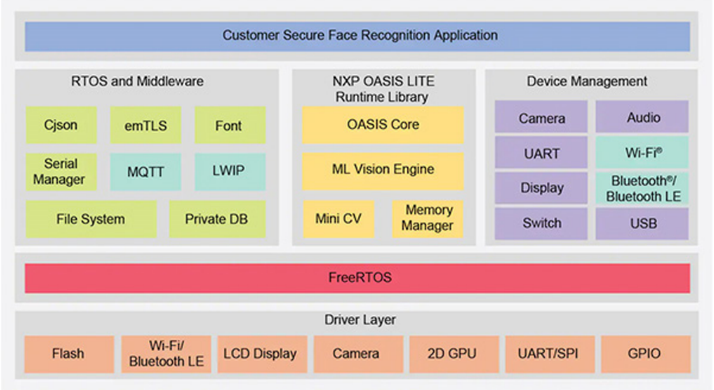


Figure 7. NXP provides a comprehensive software environment that executes the NXP Oasis Lite runtime library and utility middleware on the FreeRTOS operating system. Image source: NXP

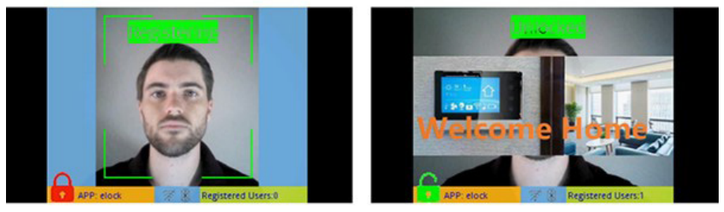
Building custom face recognition applications

Used as is, the NXP SLN-VIZNAS-IOT kit lets developers quickly evaluate, prototype and develop face recognition applications. When creating custom hardware solutions, the kit serves as a complete reference design with full schematics and a detailed bill of materials (BOM). For software development, programmers can use the NXP MCUXpresso integrated development environment (IDE) with FreeRTOS support and configuration tools. For this application, developers simply use NXP's online MCUXpresso SDK Builder to configure their software development environment with NXP's VIZNAS SDK, which includes the NXP Oasis Lite machine learning vision engine (Figure 7).

The software package includes complete source code for the operating environment as well as the elock sample application mentioned earlier. NXP does not provide source code for its proprietary Oasis Lite engine or for the models. Instead, developers work with the Oasis Lite runtime library using the provided application programming interface (API), which includes a set of intuitive function calls to perform supported operations. In addition, developers use a provided set of C defines and structures to specify various parameters including image size, memory allocation, callbacks and enabled functions used by the system when starting up the Oasis Lite runtime environment (Listing 1).

The elock sample application code demonstrates the key design patterns for launching Oasis as a task running under FreeRTOS, initializing the environment and entering its normal run stage. In the run stage, the runtime environment operates on each frame of an image, executing the provided callback functions associated with each event defined in the environment (Listing 2).

Figure 6. The NXP SLN-VIZNAS-IOT hardware kit works out of the box, utilizing a companion app to register a face (left) and recognize registered faces (right). Image source: NXP



The sample application can provide developers with step by step debug messages describing the results associated with each event processed by the event handler (EvtHandler). For example, after the quality check completes (OASISLT_EVT_QUALITY_CHK_COMPLETE), the system prints out debug messages describing the result, and after face recognition completes (OASISLT_EVT_REC_COMPLETE), the system pulls the user id and name from its database for recognized faces and prints out that information (Listing 3).

Besides supporting face recognition processing requirements, the NXP SLN-VIZNAS-IOT software is designed to protect the operating environment. To ensure runtime security, the system is designed to verify the integrity and authenticity of each signed image loaded into the system using a certificate stored in the SLN-VIZNAS-IOT kit's filesystem. As this verification sequence starts with a trusted bootloader stored in read-only memory (ROM), this process provides a chain of trust for running application firmware. Also, because code signing and verification can slow development, this verification process is designed to be bypassed during software design and debug. In fact, the SLN-VIZNAS-IOT kit comes preloaded with signed images, but code signature verification is bypassed by default. Developers can easily set options to enable full code signature

```
typedef struct {
    //max input image height, width and channel, min_face: minimum face can be detected
    int height;
    int width;

    //only valid for RGB images; for IR image, always GREY888 format
    OASISLTImageFormat_t img_format;
    OASISLTImageType_t img_type;

    //min_face should not smaller than 40
    int min_face;

    /*memory pool pointer, this memory pool should only be used by OASIS LIB*/
    char* mem_pool;

    /*memory pool size*/
    int size;

    /*output parameter,indicate authenticated or not*/
    int auth;

    /*callback functions provided by caller*/
    InfCallbacks_t cbs;

    /*what functions should be enabled in OASIS LIB*/
    uint8_t enable_flags;

    /*only valid when OASIS_ENABLE_EMO is activated*/
    OASISLTEmoMode_t emo_mode;

    /*false accept rate*/
    OASISLTFar_t false_accept_rate;

    /*model class */
    OASISLTModelClass_t mod_class;

} OASISLTInitPara_t;
```

Listing 1. Developers can modify software execution parameters by modifying the contents of structures such as the one shown here for Oasis Lite runtime initialization. Code source: NXP

```
typedef enum {
    /*indicate the start of face detection, user can update frame data if it is needed.
     * all parameter in callback parameter is invalid.*/
    OASISLT_EVT_DET_START,

    /*The end of face detection.
     *if a face is found, pfaceBox(OASISLTCbPara_t) indicated the rect(left,top,right,bottom point value)
     *info and landmark value of the face.
     *if no face is found,pfaceBox is NULL, following event will not be triggered for current frame.
     *other parameter in callback parameter is invalid */
    OASISLT_EVT_DET_COMPLETE,

    /*Face quality check is done before face recognition*/
    OASISLT_EVT_QUALITY_CHK_START,
    OASISLT_EVT_QUALITY_CHK_COMPLETE,

    /*Start of face recognition*/
    OASISLT_EVT_REC_START,

    /*The end of face recognition.
     * when face feature in current frame is gotten, GetRegisteredFaces callback will be called to get all
     * faces feature registered and OASIS lib will try to search this face in registered faces, if this face
     * is matched, a valid face ID will be set in callback parameter faceID and corresponding simularity(indicate
     * how confidence for the match) also will be set.
     * if no face match, a invalid(INVALID_FACE_ID) will be set.*/
    OASISLT_EVT_REC_COMPLETE,

    /*start of emotion recognition*/
    OASISLT_EVT_EMO_REC_START,

    /*End of emotion recognition, emoID indicate which emotion current face is.*/
    OASISLT_EVT_EMO_REC_COMPLETE,

    /*if user set a registration flag in a call of OASISLT_run and a face is detected, this two events will be notified
     * for auto registration mode, only new face(not recognized) is added(call AddNewFace callback function)
     * for manu registration mode, face will be added forcely.
     * for both cases, face ID of new added face will be set in callback function */
    OASISLT_EVT_REG_START,
    /*when registration start, for each valid frame is handled,this event will be triggered and indicate
     * registration process is going forward a little.
     * */
    OASISLT_EVT_REG_IN_PROGRESS,
    OASISLT_EVT_REG_COMPLETE,
    OASISLT_EVT_NUM

} OASISLTEvt_t;
```

Listing 2. The Oasis Lite runtime recognizes a series of events documented as an enumerated set in the Oasis Lite runtime header file. Code source: NXP

```
static void EvtHandler(ImageFrame_t *frames[], OASISLTEvt_t evt, OASISLTCbPara_t *para, void *user_data)

{

[code redacted for simplification]

    case OASISLT_EVT_QUALITY_CHK_COMPLETE:

    {

        UsbShell_Printf("[OASIS]:quality chk res:%d\r\n", para->qualityResult);


        pQMsg->msg.info.irLive  = para->reserved[5];

        pQMsg->msg.info.front   = para->reserved[1];

        pQMsg->msg.info.blur    = para->reserved[3];

        pQMsg->msg.info.rgbLive = para->reserved[8];


        if (para->qualityResult == OASIS_QUALITY_RESULT_FACE_OK_WITHOUT_GLASSES ||

            para->qualityResult == OASIS_QUALITY_RESULT_FACE_OK_WITH_GLASSES)

        {

            UsbShell_DbgPrintf(VERBOSE_MODE_L2, "[EVT]:ok!\r\n");

        }

        else if (OASIS_QUALITY_RESULT_FACE_SIDE_FACE == para->qualityResult)

        {

            UsbShell_DbgPrintf(VERBOSE_MODE_L2, "[EVT]:side face!\r\n");

        }

        else if (para->qualityResult == OASIS_QUALITY_RESULT_FACE_TOO_SMALL)

        {

            UsbShell_DbgPrintf(VERBOSE_MODE_L2, "[EVT]:Small Face!\r\n");

        }

        else if (para->qualityResult == OASIS_QUALITY_RESULT_FACE_BLUR)
```

```
        {

            UsbShell_DbgPrintf(VERBOSE_MODE_L2, "[EVT]: Blurry Face!\r\n");

        }

        else if (para->qualityResult == OASIS_QUALITY_RESULT_FAIL_LIVENESS_IR)

        {

            UsbShell_DbgPrintf(VERBOSE_MODE_L2, "[EVT]: IR Fake Face!\r\n");

        }

        else if (para->qualityResult == OASIS_QUALITY_RESULT_FAIL_LIVENESS_RGB)

        {

            UsbShell_DbgPrintf(VERBOSE_MODE_L2, "[EVT]: RGB Fake Face!\r\n");

        }

    }

    break;

[code redacted for simplification]

    case OASISLT_EVT_REC_COMPLETE:

    {

        int diff;

        unsigned id                 = para->faceID;

        OASISLTRecognizeRes_t recResult = para->recResult;


        timeState->rec_comp = Time_Now();

        pQMsg->msg.info.rt  = timeState->rec_start - timeState->rec_comp;

        face_info.rt        = pQMsg->msg.info.rt;

#ifdef SHOW_FPS

        /*pit timer unit is us*/

        timeState->rec_fps++;

        diff = abs(timeState->rec_fps_start - timeState->rec_comp);
```

```
        if (diff > 1000000 / PIT_TIMER_UNIT)

        {

            // update fps

            pQMsg->msg.info.recognize_fps = timeState->rec_fps * 1000.0f / diff;

            timeState->rec_fps          = 0;

            timeState->rec_fps_start    = timeState->rec_comp;

        }

#endif

        memset(pQMsg->msg.info.name, 0x0, sizeof(pQMsg->msg.info.name));


        if (recResult == OASIS_REC_RESULT_KNOWN_FACE)

        {

            std::string name;

            UsbShell_DbgPrintf(VERBOSE_MODE_L2, "[OASIS]:face id:%d\r\n", id);

            DB_GetName(id, name);

            memcpy(pQMsg->msg.info.name, name.c_str(), name.size());

            face_info.recognize = true;

            face_info.name      = std::string(name);

            UsbShell_DbgPrintf(VERBOSE_MODE_L2, "[OASIS]:face id:%d name:%s\r\n", id, pQMsg->msg.info.name);

        }

        else

        {

            // face is not recognized, do nothing

            UsbShell_DbgPrintf(VERBOSE_MODE_L2, "[OASIS]:face unrecognized\r\n");

            face_info.recognize = false;

        }
```

```
        VIZN_RecognizeEvent(gApiHandle, face_info);

    }

    break;
```

Listing 3. As shown in this snippet from a sample application in the NXP software distribution, an event handler processes events encountered during the face recognition sequence. Code source: NXP

verification for production.

Along with the runtime environment and associated sample application code, NXP provides Android mobile apps with full java source code. One app, the VIZNAS FaceRec Manager, provides a simple interface for registering faces and managing users. Another app, the VIZNAS Companion app, allows users to provision the kit with Wi-Fi credentials using an existing Wi-Fi or BLE connection.

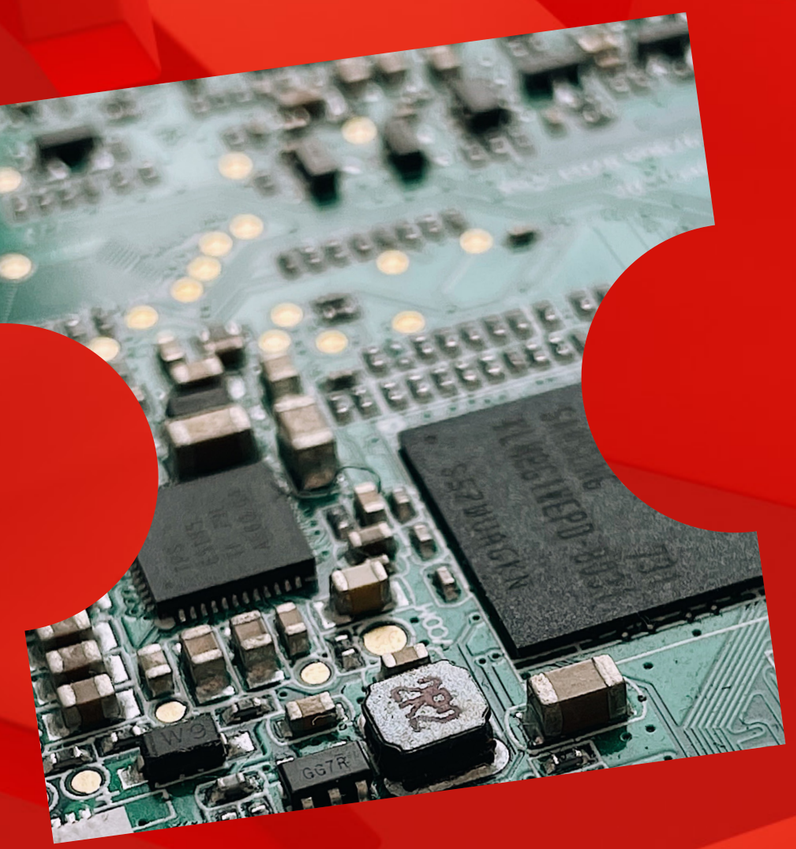
Conclusion

Face recognition offers an effective approach for authenticating access to smart products, but implementing it has typically required local high-performance computing or always-on high-bandwidth connectivity for rapid responses. It has also been a target of spoofing and is subject to concerns about user privacy.

As shown, a specialized processor and software library from NXP Semiconductors offer an alternative approach that can accurately perform offline face recognition in less than a second without a Cloud connection, while mitigating spoofing attempts.

Why and how to use Efinix FPGAs for AI/ML imaging – Part 1: getting started

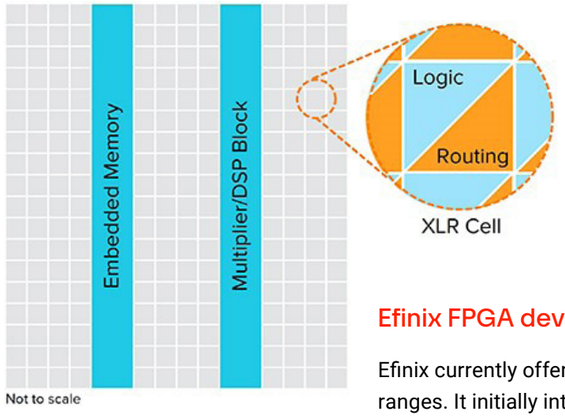
Written by: Adam Taylor



Editor's Note: New approaches to FPGA architectures bring finer-grained control and greater flexibility to address the needs of machine learning (ML) and artificial intelligence (AI). Part 1 of this two-part series introduces one such architecture from Efinix and how to get started with it using a development board. Part 2 discusses interfacing the development board to external devices and peripherals, such as a camera.

FPGAs play a critical role in many applications, from industrial control and safety to robotics, aerospace, and automotive. Thanks to the flexible nature of the programmable logic core and their wide interfacing capabilities, one growing use case for FPGAs is in image processing when ML inference is to be deployed. FPGAs are ideal for implementing solutions that have several high-speed camera interfaces. In addition, FPGAs also enable the implementation of dedicated processing pipelines in the logic, thereby removing bottlenecks that would be associated with CPU or GPU-based solutions.

Figure 1. What makes an XLR block unique is that it can be configured to function as either a logic cell with an LUT, a register and adder, or a routing matrix. Image source: Efinix



For many developers, however, their applications require more ML/AI functionality and finer-grained control or routing and logic, beyond what classic FPGA architectures with combinatorial logic blocks (CLBs) can provide. Newer approaches to FPGA architectures address these issues. For example, Efinix's Quantum architecture uses an eXchangeable Logic and Routing (XLR) block.

This article discusses key features and attributes of the Efinix FPGA architecture, emphasizing its AI/ML capabilities and introducing real-world implementations. It then discusses a development board and associated tools that developers can use to quickly get started on their next AI/ML imaging design.

Efinix FPGA devices

Efinix currently offers two device ranges. It initially introduced the Trion family, which offers logic densities from 4000 (4K) to 120K logic elements (LEs), and is fabricated using an SMIC 40LL process. The newest line of devices, the Titanium family, offers logic densities from 35K to 1 million (1M) logic elements, and is fabricated on the very popular TSMC 16 nanometer (nm) node.

Both offerings are based around the Quantum architecture, which is unique in the FPGA world. The standard FPGA architecture is based on CLBs which, at the simplest level, contain a look-up table (LUT) and flip-flops. The CLBs implement logic equations that are then interconnected via routing. Efinix's Quantum architecture moves away from distinct logic and routing blocks with the XLR block.

What makes an XLR block unique is that it can be configured to function as a logic cell with an LUT, a register and adder, or a routing matrix. This approach offers a finer-grained architecture that provides routing flexibility, enabling implementations that are logic heavy or routing heavy to achieve the desired performance.

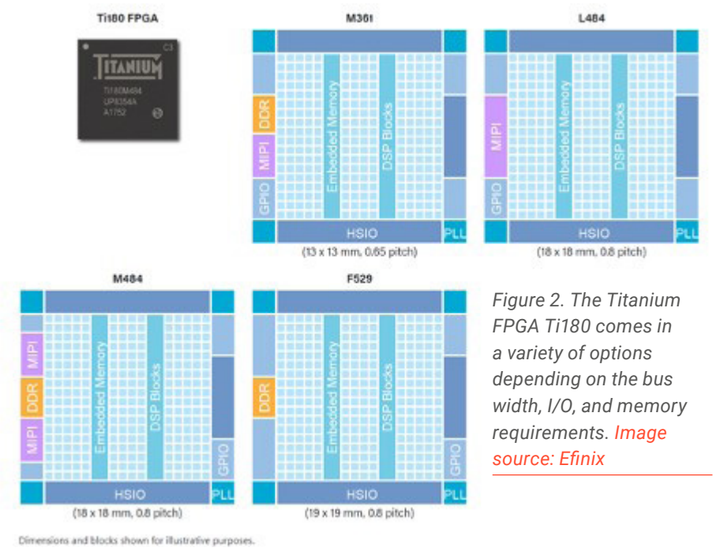


Figure 2. The Titanium FPGA Ti180 comes in a variety of options depending on the bus width, I/O, and memory requirements. Image source: Efinix

As the newest family, the Titanium devices offer the most advanced features for the developer (Figure 2). Along with the XLR core, they provide multi-gigabit serial links which operate at either 16 gigabits per second (Gbps) or 25.8 Gbps, depending on the device selected. These multi-gigabit links are crucial for enabling high-speed data transfer on and off the chip.

Titanium devices also provide a wide range of input/output (I/O) interfacing capabilities that can be grouped as general purpose I/O (GPIO), and that can support single-ended I/O standards such as low-voltage CMOS (LVCMOS) at 3.3, 2.5, and 1.8 volts.

For high-speed and differential interfacing, the Titanium devices provide high-speed I/O (HSIO) which supports single-ended I/O standards such as LVCMOS at 1.2, 1.5 volts, and SSTL and HSTL. Differential I/O standards supported by HSIO include low-voltage differential signalling (LVDS), differential SSTL, and HSTL.

Modern FPGAs also require closely coupled, high-bandwidth memory, which is used to store image frames for image processing applications, sample data for signal processing, and of course, to run operating systems and software for processors implemented within the FPGA. The Titanium range of devices provides the ability to interface with dynamic data rate four (DDR4) and low-power DDR4(x) (LPDDR4(x)). Depending on the exact Titanium device selected, the bus width support is x32 (J) or x16 (M), while some devices have no LPDDR4 support (L).

Titanium FPGAs are SRAM based and require a configuration memory, with the device configuration performed by either master/slave Serial Peripheral Interconnect (SPI) or JTAG. To ensure this configuration method is secure, the Titanium FPGA uses AES GCM encryption of the bitstream, along with AES GCM and RSA-4096 to provide bitstream authentication. Strong security like this is required since FPGAs are deployed at the edge where malicious actors could access and manipulate their behavior.

Development board introduction

Development boards form a critical element of the FPGA evaluation process since they can be used to explore the capabilities of a device and prototype applications, thereby helping to reduce overall risk. The first development board available to evaluate Titanium FPGAs and begin prototyping applications is the Ti180 M484 (Figure 3). The board features an FPGA Mezzanine Card (FMC) connector and four Samtec QSE connectors.

The Ti180 FPGA fitted to this development board provides 172K XLR cells, 32 global clocks, 640 digital signal processing (DSP) elements, and 13 megabits (Mbits) of embedded RAM. The DSP elements provide the ability to implement fixed point 18 x 19 multiplications and 48-bit multiplications. This DSP can also be optimized for single instruction, multiple data (SIMD) operations running in either a dual or quad configuration. DSP elements can also be configured to perform floating point operations.

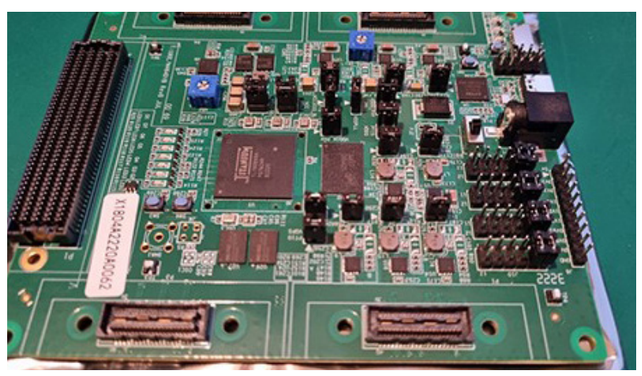


Figure 3. Along with a Titanium FPGA, the Ti180 M484 development kit features an FMC connector and four Samtec QSE connectors. Image source: Adam Taylor

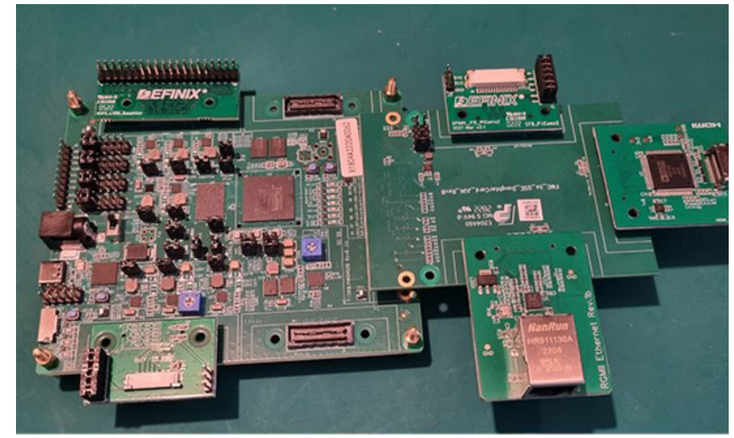


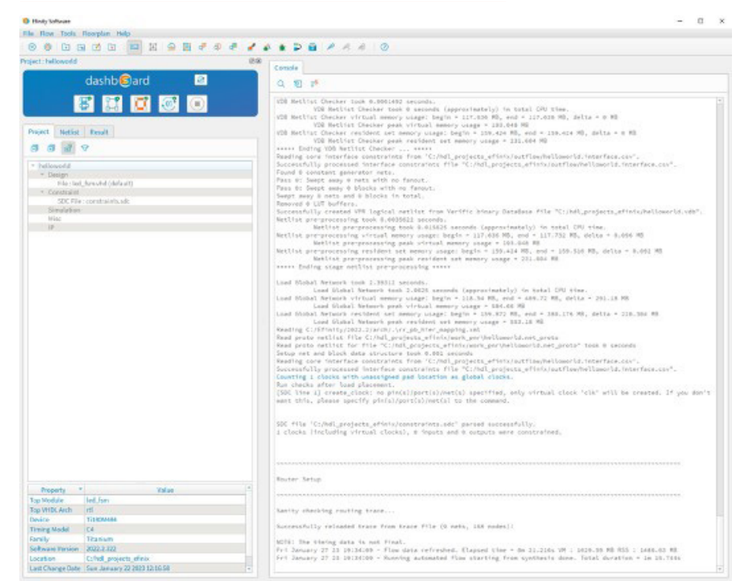
Figure 4. The Ti180 M484 development kit is shown with its versatile range of expansion options based on QSE and FMC connectors. Image source: Adam Taylor

Like most development boards, the Ti180 development board provides simple LEDs and buttons. Its real power, however, comes in its interfacing capabilities. The Ti180 development board provides a low-pin-count FMC connector that enables a wide range of peripherals to be connected. As it's a widely used standard, there are many FMC cards that enable interfacing of high-speed analog-to-digital converter (ADC), digital-to-analog converter (DAC), networking, and memory/storage solutions.

In addition to the FMC connection, the board provides four Samtec QSE connectors which enable the developer to add expansion cards. These QSE connectors are used to provide MIPI inputs and outputs, with each QSE connector providing either a MIPI input or output.

The Ti180 board also provides 256 Mbits of LPDDR4 to support the high-performance memory required in image or signal processing applications. In addition, the development board provides a range of clocking options at 25, 33.33, 50, and 74.25 megahertz (MHz), which can be used with the device phase lock loop (PLL) to generate different internal frequencies.

Figure 5. Within Efinity, new projects are created targeting the selected device. Image source: Adam Taylor



The ability to reprogram and debug live on the board during development is critical and requires a JTAG connection, which is provided on-board via a USB-C interface. Also provided is non-volatile memory in the form of two, 256-Mbit NOR flash devices that can be used to demonstrate the configuration solution.

The board is powered from a 12-volt universal power adaptor that is included in the box. Also included is an FMC-to-QSE break out, along with QSE-based expansion cards for HDMI, Ethernet, MIPI, and LVDS.

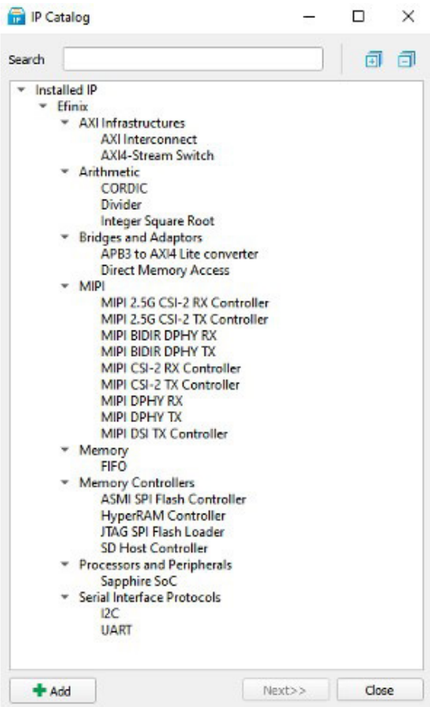


Figure 6. Efinity provides developers with an IP catalog that they can use to accelerate the design process. Image source: Adam Taylor

To demonstrate the Ti180 image processing capabilities, a dual RPI daughter card and two IMX477 camera cards are also provided.

The software environment

Implementing designs targeting the Ti180 development board use the Efinix software Efinity. The software enables the generation of a bit stream via synthesis and place and route. It also provides developers with intellectual property (IP) blocks, timing analysis, and on-chip debugging.

Note that a development board is required to gain access to the Efinity software. Unlike other vendors, though, the tool does not have different versions that require additional licensing.

Within Efinity, new projects are created targeting the selected device. RTL files can then be added to the project, and constraints created for timing and I/O design. It's within Efinity that developers are also able to implement the I/O design, utilizing the HSIO, GPIO, and specialized I/O.

A critical element of FPGA design is leveraging IP, especially for complex IP such as AXI interconnects, memory controllers, and softcore processors. Efinity provides developers with a range of IP blocks that can be used to accelerate the design process.

While FPGAs are excellent at implementing parallel processing structures, many FPGA designs include softcore processors. These provide the ability to implement sequential processing, such as network communications. To enable the deployment of the softcore processors in the Efinix devices, Efinity provides the Sapphire system-on-chip (SoC) configuration tool. Sapphire allows the developer to define a multi-processor system that has both caches and cache coherency across multiple processors, along with the ability to run an embedded Linux operating system. Within Sapphire, the developer can choose between one and four softcore processors.

The softcore processor being implemented is the VexRiscV soft CPU, which is based on the RISC-V instruction set architecture. The VexRiscV processor is a 32-bit implementation which has extensions for pipelining and offers a configurable feature set, making it ideal for implementation in Efinix devices. Optional configurations include a multiplier, atomic instructions, floating point extensions, and compressed instructions. Depending on the configuration of the SoC system, performance will range between 0.86 and 1.05 DMIPS/MHz.

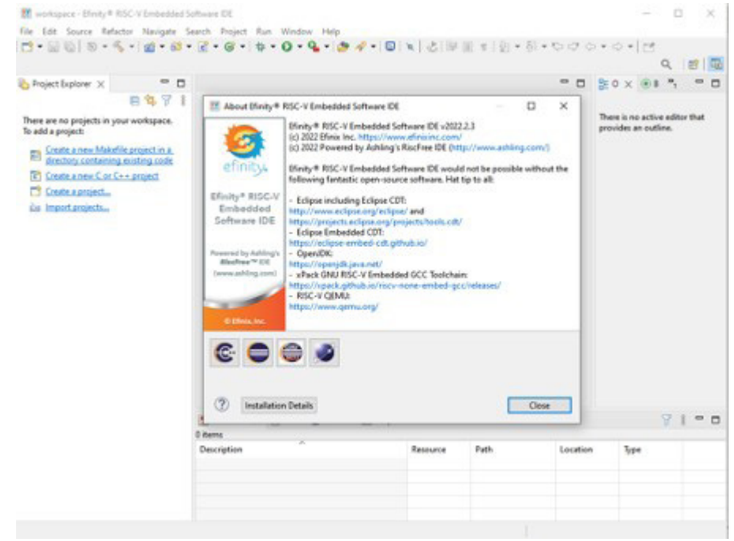


Figure 7. Ashling RiscFree is an Eclipse-based IDE that enables the creation and compilation of application software, along with debug on the target. Image source: Adam Taylor

Once the hardware environment has been designed and implemented in the Efinix device, the application software can be developed using the Ashling RiscFree IDE. Ashling RiscFree is an Eclipse-based IDE that enables the creation and compilation of application software, along with debug on the target to fine-tune the application prior to deployment.

If an embedded Linux solution is being developed, all necessary boot artifacts are provided, including First Stage Boot Loader, OpenSBI, U-Boot, and Linux using Buildroot. Alternatively, the developer can use FreeRTOS if a real-time solution is required.

AI implementation

Building upon the RISC-V softcore operation is Efinix's AI implementation. This leverages the custom instruction capability of the RISC-V processor to enable the acceleration of TensorFlow Lite solutions. The use of the RISC-V processor also enables users to create custom instructions that can be used as part of the pre-processing or post-processing following the AI inference, creating a more responsive and deterministic solution.

To get started on an AI implementation, the first step is to explore the Efinix model zoo, which is a library of AI/ML models optimized for its end technology. For developers working with the Efinix devices, the model zoo can be accessed, and the network trained using Jupyter Notebooks or Google Colab. Once the network has been trained, it can be converted from a floating point model to a quantized one using the TensorFlow Lite convertor.

Once in the TensorFlow Lite format, Efinix's tinyML accelerator can be used to create a deployable solution on the RISC-V solution. The tinyML generator enables the developer to customize the accelerator implementation and generate the project files. When deployed in this manner, the acceleration can range between 4x and 200x depending upon the selected architecture and customization.

Conclusion

Efinix devices provide developers with flexibility thanks to their unique XLR architecture. The toolchain provides the ability to not only implement RTL design, but also implement complex SoC solutions that deploy softcore RISC-V processors. Building on top of the softcore SoC is an AI/ML solution that enables the deployment of ML inference.

Why and how to use Efinix FPGAs for AI/ML imaging – Part 2: Image capture and processing

Written by: Adam Taylor

Editor's Note: New approaches to FPGA architectures bring finer-grained control and greater flexibility to address the needs of machine learning (ML) and artificial intelligence (AI). Part 1 of this two-part series introduces one such architecture from Efinix and how to get started with it using a development board. Here, Part 2 discusses interfacing the development board to external devices and peripherals such as a camera, and how to leverage the FPGA to remove image processing bottlenecks.

FPGAs play a critical role in many applications, from industrial control and safety to robotics, aerospace, and automotive. Thanks to the flexible nature of the programmable logic core and their wide interfacing capabilities, one growing use case for FPGAs is in image processing, where machine learning (ML) can be deployed. FPGAs are ideal for implementing solutions that have several high-speed camera interfaces thanks to their parallel logic structure. In addition, FPGAs also enable the use of a dedicated processing pipeline in the logic, thereby removing shared-resource bottlenecks that would be associated with CPU or GPU-based solutions.

This second look at Efinix's Titanium FPGAs will examine the reference image processing application that comes with the FPGA's Ti180 M484 development board. The aim is to understand the constituent parts of the design, and to identify where FPGA technology enables the removal of bottlenecks or enables other benefits to developers.

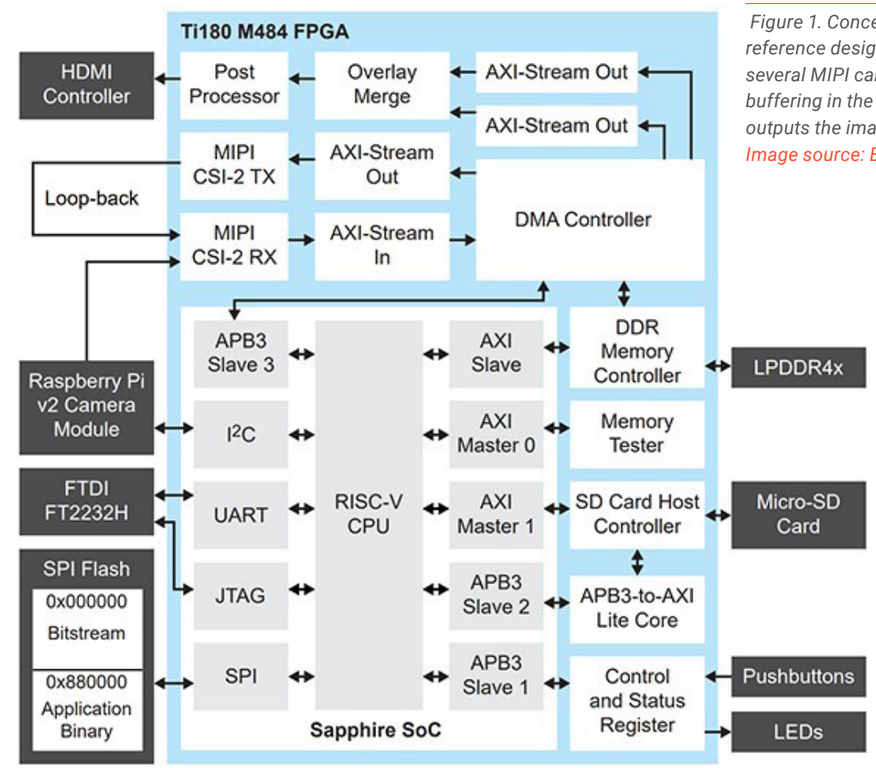


Figure 1. Conceptually, the Ti180 M484 reference design receives images from several MIPI cameras, performs frame buffering in the LPDDR4x, and then outputs the images to an HDMI display. Image source: Efinix

Figure 2. Sample images output from the reference design. Image source: Adam Taylor

The Ti180 M484-based reference design

Conceptually, the reference design (Figure 1) receives images from several Mobile Industry Processor Interface (MIPI) cameras, performs frame buffering in the LPDDR4x, and then outputs the images to a High Definition Multimedia Interface (HDMI) display. An FPGA Mezzanine Card (FMC) and four Samtec QSE interfaces on the board are used to provide the camera inputs and HDMI output.

The FMC to QSE expansion card is used in conjunction with the HDMI daughter card to provide the output video path, while three QSE connectors are used to interface with the DFRobot SEN0494 MIPI cameras. If multiple MIPI cameras are not available, a single camera can be used by looping back the single camera channel to simulate additional cameras.

At a high level, this application may appear to be straightforward. However, receiving multiple high-definition (HD) MIPI streams at a high frame rate is challenging. This is where FPGA technology is beneficial because it allows designers to utilize multiple MIPI streams in parallel.

The architecture of the reference design leverages both parallel and sequential processing structures with the FPGA. The parallel structures are used to implement the image processing pipeline, while a RISC-V processor provides the sequential processing used for the FPGA look-up tables (LUTs).

The image processing pipeline can be split into two elements within many FPGA-based image processing systems, namely the input and output streams. The input stream is connected to the camera/sensor interface, and processing functions are applied to the sensor's output. These functions can include Bayer conversion, auto white balance, and other enhancements. In the output stream, the image is prepared for display. This includes changing color spaces (e.g., RGB to YUV) and post-processing for the desired output format, such as HDMI.

Often the input image processing chain operates at the sensor pixel clock rate. This has different timing to the output chain, which is processed at the output display frequency.

A frame buffer is used to connect the input to the output processing pipeline, which is often stored in external high-performance memory, such as LPDDR4x. This frame buffer decouples between the input and output pipelines, allowing access to the frame buffer via direct memory access at the appropriate clock frequency.

The Ti180 reference design uses a similar approach to the concepts outlined above. The input image processing pipeline implements a MIPI Camera Serial Interface 2 (CSI-2) receiver intellectual property (IP) core, which is built upon the MIPI physical layer (MIPI D-PHY)-capable input/output (I/O) of the Titanium FPGA. MIPI is a challenging interface because it uses both single-ended and differential signaling on the same differential pair, in addition to low-speed and high-speed communications. Integrating the

The Ti180 M484 reference design clearly showcases the capabilities of Efinix FPGAs and the Ti180 in particular. The design leverages several of the unique I/O structures to implement a complex image processing path that supports several incoming MIPI streams.

MIPI D-PHY within the FPGA I/O reduces the complexity of the circuit card design while also reducing the bill of materials (BOM).

With the image stream from the camera received, the reference design then converts the output of the MIPI CSI-2 RX into an Advanced eXtensible Interface (AXI) Stream. An AXI Stream is a unidirectional high-speed interface that provides a stream of data from a master to a slave. Handshaking signals to transfer between a master and slave are provided (tvalid and tready) along with sideband signals. These sideband signals can be used to convey image timing information such as start of frame and end of line.

AXI Stream is ideal for image processing applications and enables Efinix to provide a range of image processing IP which can then be easily integrated into the processing chain as required by the application.

After being received, the MIPI CSI-2 image data and timing signals are converted into an AXI Stream and input into a direct memory access (DMA) module, which writes the image frame to the LPDDR4x and acts as the frame buffer.

This DMA module is operating under the control of the RISC-V core in the FPGA within a Sapphire system on chip (SoC). This SoC provides control, such as stopping and starting DMA writes, in addition to providing the DMA write channel with the necessary information to correctly write the image data to the LPDDR4x. This includes information on the memory location and the width and height of the image defined in bytes.

The output channel in the reference design reads the image information from the LPDDR4x frame buffer under the control of the RISC-V SoC. The data is output from the DMA IP as an AXI Stream, which is then converted from RAW format provided by the sensor to RGB format (Figure 2), and prepared for output over the on-board Analog Devices' ADV7511 HDMI transmitter.

The use of the DMA also enables the Sapphire SoC RISC-V to access the images stored within the frame buffer, and the abstract statistics and image information. The Sapphire SoC is also able to write overlays into the LPDDR4x so that they can be merged with the output video stream.

Modern CMOS image sensors (CISs) have several modes of operation and can be configured to provide on-chip processing, and several different output formats and clocking schemes. This configuration is normally provided over an I²C interface. In the Efinix reference design, this I²C communication to the MIPI cameras is provided by the Sapphire SoC RISC-V processor.

Integration of the RISC-V processor within the Titanium FPGA reduces the overall size of the final solution as it removes the need to implement both complex FPGA state machines that increase design risk, as well as external processors that add to the BOM.

Inclusion of the processor also enables support with additional IP to communicate with the MicroSD card. This enables real-world applications where images may be required to be stored for later analysis.

Overall, the architecture of the Ti180 reference design is optimized to enable a compact, low-cost, yet high-performance solution that allows developers to reduce BOM

Core Resources	
Inputs	1264 / 3706
Outputs	1725 / 4655
XLRs	73587 / 172800
Memory Blocks	508 / 1280
DSP Blocks	4 / 640

Table 1: Resource allocation on the Efinix architecture shows only 42% of the XLR cells are used, leaving ample room for additional processes. Image source: Adam Taylor

cost through system integration.

One of the key benefits of reference designs is that they can be used to kickstart application development on custom hardware, enabling developers to take critical elements of the design and build off it with their needed customizations. This includes the ability to use Efinix's TinyML flow to implement vision-based TinyML applications running on the FPGA. This can leverage both the parallel nature of FPGA logic and the ability to easily add custom instructions into RISC-V processors, allowing the creation of accelerators within the FPGA logic.

Implementation

As discussed in Part 1, the Efinix architecture is unique in that it uses eXchangeable Logic and Routing (XLR) cells to provide both routing and logic functionality. A video system such as the reference design is a mixed one that is both logic and routing heavy: extensive logic is required to implement the image processing features, and extensive routing is needed to connect the IP cells at the required frequencies.

The reference design uses approximately 42% of the XLR cells within the device, leaving ample room for additions, including custom applications such as edge ML.

Periphery Resource	
DDR	1 / 1
GPIO	22 / 27
HSIO	20.0 / 59
JTAG User TAP	1 / 4
MIPI RX	4 / 4
MIPI TX	4 / 4
Oscillator	0 / 1
PLL	4 / 8

Usage of the block RAM and digital signal processing (DSP) blocks is also very efficient, using only 4 of the 640 DSP blocks and 40% of the memory blocks (Table 1).

At the device IO, the DDR interface for the LPDDR4x is used to provide the application memory for the Sapphire SoC and the image frame buffers. All of the device-dedicated MIPI resources are utilized along with 50% of the phase lock loops (Table 2).

The general purpose I/O (GPIO) is used to provide the I²C communications along with several of the interfaces connected to the Sapphire SoC, including NOR FLASH, USB UART, and SD card. The HSIO is used to provide the high-speed video output to the ADC7511 HDMI transmitter.

One of the crucial elements when designing with FPGAs is not only implementing and fitting the design within the FPGA, but also being able to place the logic design within the FPGA and achieve the required timing performance when routed.

Long gone are the days of single-clock-domain FPGA designs. There are several different clocks, all running at high frequencies in the Ti180 reference design. The final timing table shows the maximum frequencies achieved for the clocks within the system. This is where the

Table 2: Snapshot of the interface and I/O resources used. Image source: Adam Taylor

Timing	
Worst Negative Slack (WNS)	0.182 ns
Worst Hold Slack (WHS)	0.026 ns
i_pixel_clk	211.909 MHz
tx_escclk	261.370 MHz
i_pixel_clk_tx	210.881 MHz
i_sys_clk	755.858 MHz
i_axi0_mem_clk	130.429 MHz
i_sys_clk_25mhz	234.577 MHz
i_soc_clk	187.231 MHz
i_hdmi_clk	233.918 MHz
mipi_dphy_rx_inst1_WORD_CLKOUT_HS	273.973 MHz
mipi_dphy_rx_inst2_WORD_CLKOUT_HS	262.881 MHz
mipi_dphy_rx_inst3_WORD_CLKOUT_HS	204.290 MHz
mipi_dphy_rx_inst4_WORD_CLKOUT_HS	207.598 MHz
mipi_dphy_tx_inst1_SLOWCLK	201.979 MHz
mipi_dphy_tx_inst2_SLOWCLK	191.865 MHz
mipi_dphy_tx_inst3_SLOWCLK	165.235 MHz
mipi_dphy_tx_inst4_SLOWCLK	160.823 MHz
jtag_inst1_TCK	180.505 MHz

Table 3: Timing implementation against the constraints shows the potential of the Titanium FPGA XLR structure to reduce the possible routing delay, thereby increasing design performance. Image source: Adam Taylor

```
# PLL Constraints
##################
create_clock -period 10.0000 mipi_clk
create_clock -period 10.0000 i_pixel_clk
create_clock -period 10.0000 rx_cfgclk
create_clock -period 50.0000 tx_escclk
create_clock -period 25.0000 i_pixel_clk_tx
create_clock -period 10.0000 i_sys_clk
create_clock -period 10.0000 i_axi0_mem_clk
create_clock -period 40.0000 i_sys_clk_25mhz
create_clock -period 10.0000 i_soc_clk
create_clock -period 6.7340 i_hdmi_clk
create_clock -period 7.5008 pll_ddr_CLKOUT0
```

Figure 5. Clock constraints for the reference design. Image source: Adam Taylor

requested timing performance can also be seen in the constraints (Figure 5), which have a maximum clock frequency of 148.5 megahertz (MHz) for the HDMI output clock.

Timing implementation against the constraints shows the potential of the Titanium FPGA XLR structure as it reduces the possible routing delay, thereby increasing design performance (Table 3).

Conclusion

The Ti180 M484 reference design clearly showcases the capabilities of Efinix FPGAs and the Ti180 in particular. The design leverages several of the unique I/O structures to implement a complex image processing path that supports several incoming MIPI streams. This image processing system operates under the control of a soft-core Sapphire SoC, which implements the necessary sequential processing elements of the application.

Powering the Edge: the evolution of AI from digital to neuromorphic systems for ultra-low power performance

Contributed By DigiKey's European Editors

There are many types of machine learning. AI implemented at the Edge of the network and embedded into devices is bringing significant advantages in performance and power consumption.

But there are also many other types of AI and machine learning algorithms being used in all kinds of different places, not just the data centre. The technology has been evolving from digital deep neural networks (DNN) and convolutional neural networks (CNNs) to transformer networks.

At the same time some of these embedded AI chips are using analog approaches for more performance at much lower power, particularly for processing signals from sensors locally without having to send the data to the Cloud.

All these digital and analog technologies are also coming together to provide lower power and more performance.

Image recognition and computer vision that has been a stalwart application of AI, reliably identifying defects on the production line at speeds beyond the human eye. This AI capability is moving further to the Edge of the network, down to sensors in the field.

Embedded microcontrollers from STMicroelectronics, NXP Semiconductors, Renesas Electronics, Infineon Technologies and Analog Devices have all kinds of different types of digital AI accelerator blocks alongside their CPU cores, often dedicated to particular applications, whether that's refining sensor data, correcting for errors or pattern recognition for images for spoken words.

Embedded algorithms are moving from digital signal processing (DSP) to CNNs to transformers for object recognition and detection and pose detection, whether that's for fault inspection on a production line, shelf monitoring in a warehouse or person monitoring in the smart home.

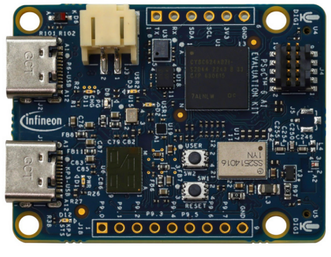


Figure 1: An Edge AI evaluation board from Infineon Technologies

One of the challenges is that data from sensors has a lot of zeros. This sparsity of data is a major challenge for digital AI chips, which previously have had to process data whether it is a 0 or a 1. The latest designs tackle this sparsity head on, reducing the amount of processing required and so the amount of energy used.

Femtosense in the US for example has designed an AI accelerator that is optimised for sparse networks, both reducing the amount of data and operating directly on the compressed data flow. This allows the AI framework to fit into memory on the chip, with 1Mbit of SRAM available in the first generation chip, slashing the power consumption of the edge AI operations.

Building the chip on a 22nm fully depleted silicon on insulator (FD SOI) process also helps reduce the power consumption even further. Rather than putting the chip down on a PCB it has been combined with a 40nm microcontroller in the same package to make it even easier for engineers to use without it being too expensive.

There are quite a few accelerators for all kinds of applications. Hailo and Ambarella are seeing success in driver safety systems, and self-driving cars and trucks, while Axelera has developed an architecture that handles the AI models in memory. This 'in memory compute' can dramatically cut the power consumption for edge applications. The chip, built on a 12nm process at TSMC, has been benchmarked at 480frame/s handling YOLO AI video analysis on 16 HD streams simultaneously for embedded security camera applications.

The Axelera Metis chip is now on M.2 boards for easy integration with controllers and Hailo has been working with Raspberry Pi on its Pi 5 AI Kit. This brings access to the Hailo 8 AI accelerator to both professional and enthusiast creators for home automation, security and robotics based on the Raspberry Pi 5 board.

The AI Kit is designed for the Raspberry Pi 5 and uses the M.2 HAT+ connection to add the Hailo-8L M.2 AI acceleration module. This provides 13 TOPS of edge AI inference for computer vision and other edge AI applications.

The key for developers is that the accelerator is fully integrated with Raspberry Pi's camera software stack and supports numerous out-of-the-box AI applications through Hailo's software suite and model zoo.

This enables Raspberry Pi's industrial customers to integrate AI into high-performance solutions that are extremely cost-effective and power-efficient. For enthusiasts, the AI Kit provides an accessible way to enhance their creative projects with AI.

Syntiant is also adding in AI accelerators for handling sensor data. Its Neural Decision Processors (NDPs) are specifically designed to run deep learning models, providing 100x the efficiency and 10/30x the throughput of existing low-power microcontrollers. These NDPs can be used for acoustic event detection for security applications to video processing in teleconferencing and equip almost any device with real-time data processing and decision making with near-zero latency and without the need for libraries or compilers.

Analog Edge AI

These are all digital AI implementations, but as every engineer knows, there is often another way.

One increasingly popular approach is neuromorphic, or spiking, AI. Neuromorphic is replicating

Figure 2: Combining a sparse neural network accelerator with an ARM microcontroller in a single package

the structure of the brain, with interconnected neurons. When a signal is detected, a spike of data propogates thorough the network. These spiking networks are much lower power as they only use the neurons in the path.

These can be used for always on audio detectors in chips from companies such as POLYN, or an image processor from Prophesee which is teamed with the Akida spiking neural network from Brainchip as IP that can be integrate into other chips.

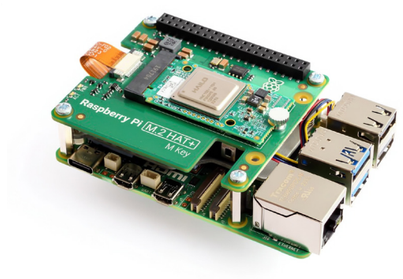
POLYN in Cambridge, UK, developed its Neuromorphic Analog Signal Processing (NASP) to handle any type of sensor and add all kinds of edge AI algorithms. The neurons in the chip are neurons are physically implemented as an analog circuitry elements according to the mathematical simulation of a single neuron, and optimised for TinyML algorithms.

TinyML cuts down the amount of data that needs to be processed with various techniques such as embeddings. An embedding is a function that can map a discrete list of values into a continuous vector to be processed by an edge AI engine and is easier to train.

This allows the AI computations can be performed directly on the device and do not require users to send data to the cloud or a remote server.

The NASP chips are true Tiny AI

Figure 3: The Hailo 8 is the first AI accelerator to be added to the Raspberry Pi 5 single board computer



implementations that improve latency and power consumption, and enable inference computations directly on devices like wearables, IoT sensors and more, increasing their functionality but also improving users' privacy as the data stays on the device.

Polyn has various implementations for its analog AI, handling vibration data to extract useful data from a sensor or to watch for and recognise a wake word or for voice control. Algorithm-based data compression does not work for noisy signals because of the fundamentally linear aspect of algorithms. Neural networks on the other hand can extract useful information even from very noisy data, due to a non-linear way they process data.

Some deep neural network architectures such as NASP prove to be exceptionally well suited for addressing vibration monitoring challenges.

Dutch neuromorphic AI chipmaker Innatera has combined an ultra-

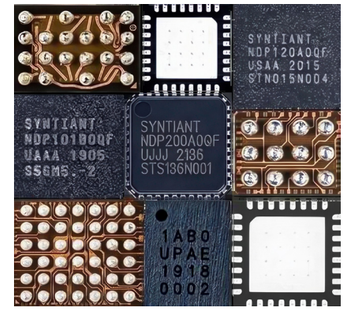


Figure 4: The Syntiant neural decision processor

low-power spiking neural network engine and a custom 32bit microcontroller core using the open RISC-V instruction set architecture (ISA) with 384 KB of embedded SRAM memory. This creates a single chip that processing sensor data quickly and efficiently with power consumption under 1mW. This is similarly being used for signal processing and pattern recognition tasks using spiking neural networks alongside DNNs and conventional processing in the same device. All of this fits into a 2.16mm x 3mm chip in a 35 pin wafer scale package

There is also a key trend to combining analog neuromorphic AI and digital AI technologies.

The first generation of the Akida neural spiking processor developed by Brainchip has been evaluated by NASA for handling sensors on space missions, and the second-generation now includes Temporal Event Based Neural Nets (TENN) spatial-temporal convolutions that supercharge the processing of raw time-continuous streaming data, such as video analytics, target tracking and audio classification. This can boost the analysis of MRI and CT medical scans for vital signs prediction, and time series analytics used in forecasting, and predictive maintenance to highlight when equipment is going to fail so that repairs can be scheduled..

The TENNs allow for radically simpler implementations by consuming raw data directly from sensors. Like the Polyn and Innatera approaches, this drastically reduces model size and operations performed, while maintaining very high accuracy. This can shrink design cycles and lower the cost of development for customers such as Renesas Electronics.

But Brainchip has also added in support for digital Vision Transformers (ViT) acceleration for image classification, object detection, and semantic segmentation. This allows it to self-manage the execution of complex networks like RESNET-50 completely in the neural processor without CPU intervention and minimizes system load.

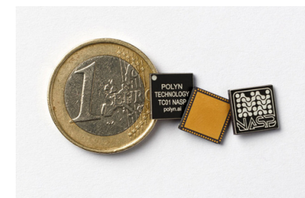


Figure 5: The NASP chip from POLYN

Figure 6: The Akida 1000 neuromorphic AI IP

The Akida IP platform can also learn on the chip, allowing continuous improvement and data-less customization that improves security and privacy. This is being used in secure, small form factor devices like hearable and wearable devices, that take raw audio input, medical devices for monitoring heart and respiratory rates and other vitals that consume only microwatts of power.

This can scale up to HD-resolution vision solutions delivered through high-value, battery-operated or fanless devices enabling a wide variety of applications from surveillance systems to factory management and augmented reality to scale effectively.

All of this marks the combination of the spiking and digital neural networks, with a focus on ultra low power. This combination of technologies is potentially a key step forward for scaling up the size and performance of all kinds of embedded AI systems without driving up the power consumption.

All about AI/machine learning

Contributed By DigiKey's
Mark Hughes

Ants are fascinating creatures. When they leave their nest in search of food, they initially wander randomly, leaving pheromones along their path. Once an ant finds food, it returns to the nest, reinforcing the trail with more pheromones. Other ants follow this strengthened trail and continue to add pheromones, making it even more prominent. Occasionally, ants wander off the trail and if they discover a shorter path to the food, the trail will gradually shift to follow this new route. Over time, this process transforms a weak and meandering trail into a streamlined ant superhighway.

The process of how ants explore their environment is similar to how brains learn, and machine learning algorithms operate: They start by exploring many possibilities; they identify successful outcomes; and then they optimize and reinforce pathways to make the connections more efficient over time.

This article covers how machine learning works, its relation to the ant analogy, and encourages you to consider using machine learning in your next project!

Machine learning basics

When laymen hear the term artificial intelligence, their thoughts often turn to the intelligent, personified machines and humanoid robots often seen in movies and television – machines capable of completing any task, who inevitably turn on their creators in a bid to take over the world.

Reality is quite different – machines are only able to learn one task at a time, and once their training stops, their ability to evolve stops with it. Machine learning 'intelligence' is restricted to the completion of a single task and is frozen in time. For example, a camera trained to detect vehicle license plate numbers, can only ever detect license plate numbers. It cannot 'evolve' to edit the grammar in written documents.

What is machine learning?

Machine learning is one type of artificial intelligence that uses advanced algorithms and a trove of data to teach computers to make predictions. Unlike the omnipotent robots of science fiction, machine learning models can only perform specific, isolated tasks, such as image recognition and classification, language translation, or trend detection. There are three types of machine learning: supervised learning, unsupervised learning, and reinforcement learning.

Types of learning

Supervised learning

Supervised learning involves pairing known inputs with known outputs. For example, how do computers learn to recognize handwritten digits when each person's handwriting is unique?

The MNIST database (http://yann.lecun.com/exdb/mnist/) contains 70,000 examples of handwritten digits (0-9) collected from the National Institute of Science and Technology Standard Reference Data Special Database 1 and 3. Researchers collected hundreds of thousands of handwritten characters and meticulously labeled each one with the correct digit. This dataset is used to train computer models to automatically recognize digits.

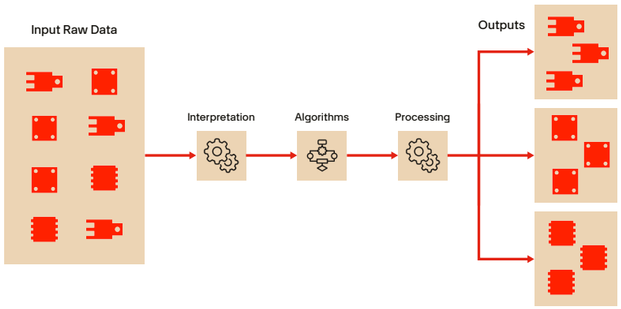
By using this dataset, computer scientists can train and test their algorithms with known correct answers. This approach ensures that the algorithm learns accurately without deviating from the correct path, which is the essence of supervised learning.

Unsupervised learning

Unsupervised learning algorithms identify patterns, similarities, and differences in large datasets that might not be visible to humans. Mathematicians discovered long ago how to determine correlation between a single dependent and independent variable, and even how to use advanced statistical techniques to find a relationship between two or three independent and one or two dependent variables. But sometimes the relationship between the variables is obfuscated by hidden variables. For those cases, machine learning takes over in order to determine hidden relationships.

By feeding the raw input data into the model and letting the algorithms experiment, it is possible to determine relationships,

Figure 1: The image illustrates unsupervised learning. In unsupervised learning, a large assortment of data is fed into an algorithm whose job it is to find patterns, coincidences, and anomalies. The algorithm sorts the data in a manner that suits it.



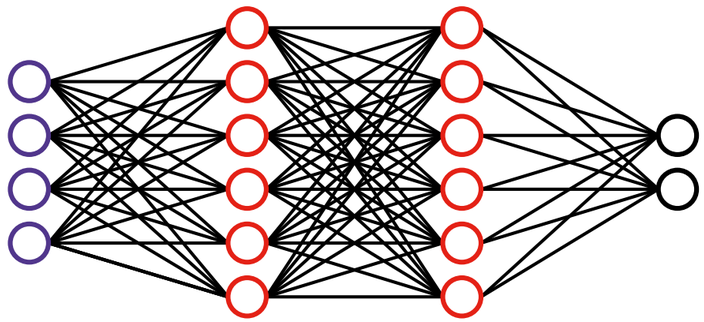


Figure 2: Neural networks are often shown as a diagram similar to the one pictured above. Data arrives and is stored in an input layer (the 4 neurons shown on left). From there, there are 'hidden-layers' that form connections to output layers. In this image there are two sets of 6 neurons in the hidden layers, and two neurons in the output layer. The training task determines how input layers are ultimately connected to output layers through the hidden layers

groupings, and outliers.

The image above illustrates unsupervised learning. In unsupervised learning, a large assortment of data is fed into an algorithm whose job it is to find patterns, coincidences, and anomalies. The algorithm sorts the data in a manner that suits it.

As an example: Every day, newspapers around the world produce articles on a wide variety of topics. These articles are often published online, which makes them available as raw input data for search engines. Using unsupervised learning, it is possible for Google and other websites to analyze the data and cluster similar articles together for a reader.

Reinforcement learning

Reinforcement learning is a process where positive feedback is given after a desired action to increase the likelihood of it happening again, and negative feedback is provided to decrease the likelihood of an undesirable action being repeated.

Learning piano

Imagine a student learning to play a piano, but the piano has all its strings cut. Without sound from key presses, there is no feedback, making it impossible for the student to know if they played the correct or incorrect key. Consequently, no reinforcement learning can take place. Now, picture the same student with a properly tuned piano. Each key press produces a sound. If the sound matches the student's expectations, it serves as positive reinforcement. If the sound is unpleasant or unexpected, it acts as negative reinforcement.

Ant trails

An ant that follows a pheromone trail to a food supply will strengthen the trail by depositing more pheromones. If the trail leads to an exhausted food supply, the ants will wander elsewhere and the pheromones will dissipate. With a higher pheromone concentration, there is an increased likelihood that more ants will follow. With fewer pheromones, the lower the likelihood that ants will follow that trail.

Early machine learning

In the mid-1990s, computer scientists developed TD-Gammon, an algorithm that learned to play backgammon through reinforcement learning. Initially, it made random moves and received feedback based on the outcomes of those moves. Actions that increased the game's expected score received positive reinforcement by increasing a reward variable, while actions that decreased the expected score received negative reinforcement by decreasing the reward variable. The algorithm was programmed to maximize the cumulative reward. Through trial and error, it gradually learned which decisions maximize the reward and win the game!

Reinforcement review

This basic process of positive and negative reinforcement – increasing or decreasing the likelihood of a behavior – is how humans, insects, and machines learn. Brains make neuronal connections, ants make trails, and computers increase or decrease the values of variables.

It is important to train using data that is appropriate for the task and free of unnecessary information when possible. This approach reduces the computational complexity of the training process.

Relation to machine learning

Machines interact with the world around them through a series of inputs and outputs. The inputs often come from sensors or data files, while the outputs can be screens, actuators, or data files. By training models on data obtained from sensors, you can develop systems that understand and respond to their environments in near real-time. For instance, in autonomous driving, sensor data helps vehicles navigate and make decisions safely and efficiently. In personalized medicine, machines analyze patient data to allow doctors to better diagnose and treat patients.

The ability to process and learn from diverse data sources is crucial for advancements in machine learning.

Real-world applications of machine learning

Algorithms need data, and there are a wide number of sensors available for use today. Often the outputs of a variety of sensors are combined – a process called sensor fusion. Sensor fusion combines the data from multiple sensors in different proportions to create a dataset product that is greater than the sum of its parts. For example, smartphones combine data from GPS, accelerometers, gyroscopes, magnetometers, and the camera to determine where it is in space. The result allows for a variety of augmented reality experiences, such as Pokemon Go!, or Google Maps indoor navigation.

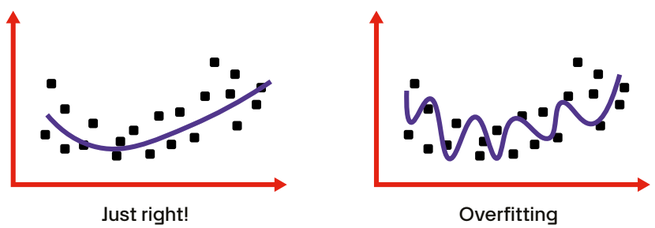
In automobiles

Modern automobiles have dozens of sensors, including transmission-speed, wheel-speed, steering-angle, suspension-height, tire-pressure, accelerometers, yaw-rate, microphone arrays, and more. I chose to list these sensors in particular because a company called Tactile Mobility fuses the data from these sensors to produce a very detailed picture of road conditions. Application specific algorithms are able to analyze all of the sensor data coming out of a vehicle and determine which vehicle-type produced the data, what condition the tires are in, what road the vehicle was traveling on, and even the specific lane of travel – all without using GPS.

In healthcare

Healthcare combines several data points together to provide a picture of patient health. A patient's vitals include: blood pressure, pulse rate, respiratory rate, and temperature. Other data, such as the results of a Complete Blood Count (7 tests) and a Comprehensive Metabolic Panel (14 tests) can identify which

Figure 3: An example of a model that's 'Just right!' and Overfitting



Machine learning is a powerful tool, but it is task specific and lacks the general intelligence seen in Science Fiction movies and fantasy novels.

organs might be functioning poorly and might require further testing. Doctors learn to interpret the data and generate results after years of intense education and practice. But researchers also train computers to fuse the data from the various sources and produce a list of likely causes of an illness.

As another example, amputees today have more options than the peg-legged pirate of centuries past. Modern prosthetics now can include accelerometers, gyroscopes, magnetometers, pressure sensors, force sensors, and electromyography (muscle) sensors to provide enhanced capabilities and natural movement. The data from the sensors allows a prosthetic to know where it is in space and interact with the nervous system without having to merge with the nervous system.

Looking forward

Now, it is time to learn how neural networks work, and hopefully demystify the process enough that you'll be confident enough to try it out on your own.

The learning process

Machine learning is an involved process that requires significant amounts of data and processing power. The steps include: data collection, error detection, and deployment.

Neural networks, a type of processor, feed sensor data into an input layer. The input layer then passes the data to a series of hidden layers containing neurons, and eventually to the output layer. The computationally intensive part of training involves determining the values of the individual neurons.

Neural networks are often shown as a diagram similar to the one pictured above. Data arrives and is stored in an input layer (the 4 blue neurons shown on left). From there, there are multiple 'hidden-layers' that form connections to output layers. In this image there are two sets of 6 neurons in the hidden layers, and two neurons in the output layer. The training task determines how input layers are ultimately connected to output layers through the hidden layers

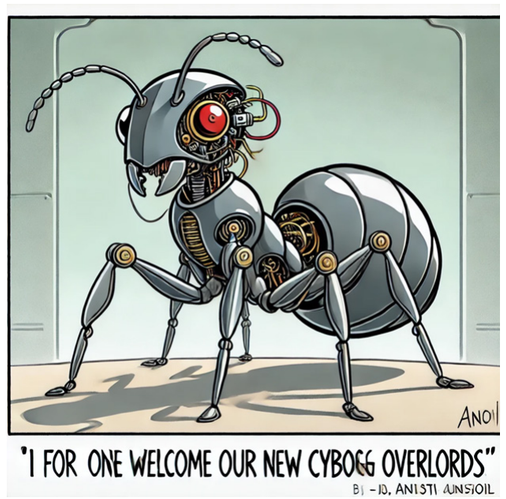
Data collection

Substantial amounts of data are required to properly train a model. In the handwriting example mentioned at the start of the article, sixty-thousand of the seventy-thousand handwritten digits were used to train a detection algorithm.

Data preprocessing

Computer scientists enjoy saying "Garbage in equals Garbage out." Which means the data fed into an algorithm will impact its later performance. For this reason, if you're training an algorithm to detect handwritten digits, you cannot accidentally include a few hundred photographs of your cat or newborn child.

You must evaluate the data for missing values, outliers, duplicates, etc. to ensure that the best possible dataset is fed to the computer



for training. Normalizing the data provides mathematical reductions in the computational complexity and reduces the amount of required memory since all values will fall in a known range.

Finally, split the data into two batches - one large batch for training the algorithms, and a second smaller batch for testing the quality of the training.

Training and deployment resources

Training often requires a substantial number of addition and multiplication calculations and a lot of available memory. At times you can train a model at home on powerful computing devices such as the NVidia Jetson series of Single-Board-Computers or graphics cards, and other times training involves server farms at data-centers. Training will take anywhere between a few hours and a few months depending on the complexity of the model and the available processing power.

One reason training is so computationally expensive compared to deployment is the sheer volume of data used in training as well as the need for both forward and backward propagation to determine the value of each neuron. Once training determines the value of each neuron, deployment is far more straight-forward. Remembering the ant analogy, lots of ants are initially needed to find a food source, many more are needed to refine and streamline the trail, but once the trail is established, a single ant can mindlessly follow it without much effort.

Training tips

For neural networks, the data has to be sufficiently shuffled to properly train. If you were to feed a digit-recognition model six-thousand images of the number 0, followed by six-thousand images of the number 1, then 2, etc. the neural net would not properly form, and no useful predictions could be made. Proper shuffling ensures that the model learns to generalize from the data rather than memorizing the order of the inputs.

The scatterplots above are often used to illustrate the goal of training. Algorithms attempt to find models that fit the data reasonably well without over-training. Overtraining means the model will detect the training data with high accuracy, and the testing data with very low accuracy.

It is important not to over-train a model. There is always a bit of statistical variation, gaussian noise, or error in the datasets. A properly trained model will ignore the noise of the training dataset. An overtrained model will learn that the noise is important, and part of the dataset. As an example, imagine you've got a list of temperatures, measured at noon at a single location over the course of a year. A properly trained dataset will fit a smooth sinusoidal curve to the scatterplot, while an overtrained dataset will resemble a scatter plot connected with sharp lines.

Summary

Machine learning is a powerful tool, but it is task specific and lacks the general intelligence seen in Science Fiction movies and fantasy novels. The models used to generate outputs are dependent on the quality of the training process and are not capable of generalization to new areas - a model trained to correct grammar cannot be used to identify license plate numbers in security footage. In contrast, the human brain has no such limitations - its ability to identify, abstract, and generalize the world around it is seemingly boundless. The hopes of having an Android companion are well beyond the limits of technology for the foreseeable future.

In recent years, advances in computers have made training and deploying models accessible and affordable to everyone, democratizing AI for everyone with even basic programming skills. To learn more, please see the excellent work done by our very own Shawn Hymel here at Digikey.com!

We've got the new products your ideas deserve

We have over 400,000 new, name-brand products in stock and ready to ship–with more added daily. If you can design it, we can help you build it.

Find what you need at digikey.com/new or call 1.800.344.4539

DigiKey

we get technical

DigiKey is an authorized distributor for all supplier partners. New products added daily. DigiKey and DigiKey Electronics are registered trademarks of DigiKey Electronics in the U.S. and other countries. © 2024 DigiKey Electronics, 701 Brooks Ave. South, Thief River Falls, MN 56701, USA

ECIA MEMBER
Supporting The Authorized Channel