XCORE.AI
TECHNICAL OVERVIEW

WHITEPAPER

7 FEBRUARY 2020

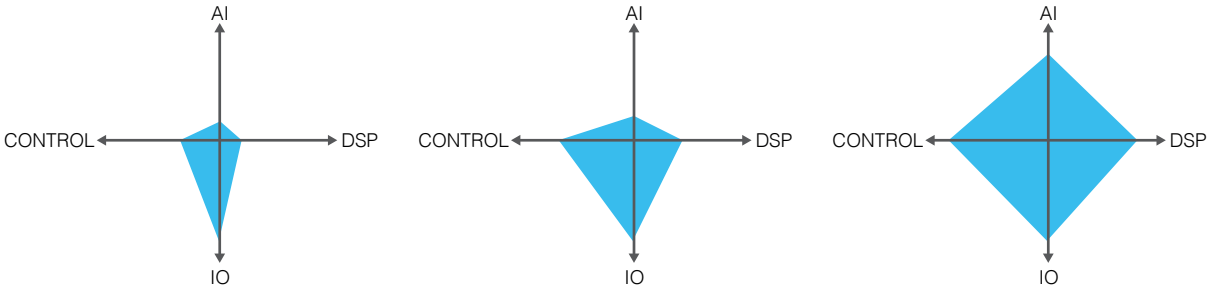
XMOS

1. INTRODUCTION

This white paper presents an introduction to the third generation of xcore architecture. In 2005, xcore was developed to deliver “FPGA-like” IO flexibility with significant control processing for embedded software engineers - enabling them to create differentiated products versus reproducing reference designs. The success of the first-generation architecture was demonstrated by hundreds of design wins in applications that bridge between IO protocols, most notably USB Audio Class 2 solutions and application-specific interfaces like S/PDIF – all of which could be implemented in software.

The second generation xocre added significant additional control and DSP performance through the addition of a dual-issue pipeline and other supporting enhancements. This enabled integration of signal conditioning capabilities to many of our existing customer solutions and gave XMOS a platform to become one of the most prominent suppliers of far-field voice processing solutions to the smart home.

Our latest xcore development – xcore.ai is a general-purpose crossover processor that increases performance across all of the original IO, control and DSP processing classes, as well as adding a completely new machine learning capability.



In summary, xcore.ai supports the following features:

- Vector arithmetic up to 38 billion multiply accumulates per second.
- Complex arithmetic up to one million 512-point FFTs per second.
- 2 x 512 kByte on-chip SRAM, with LPDDR interface for optional external memory
- 128 GPIO, on-chip USB transceiver and MIPI receiver

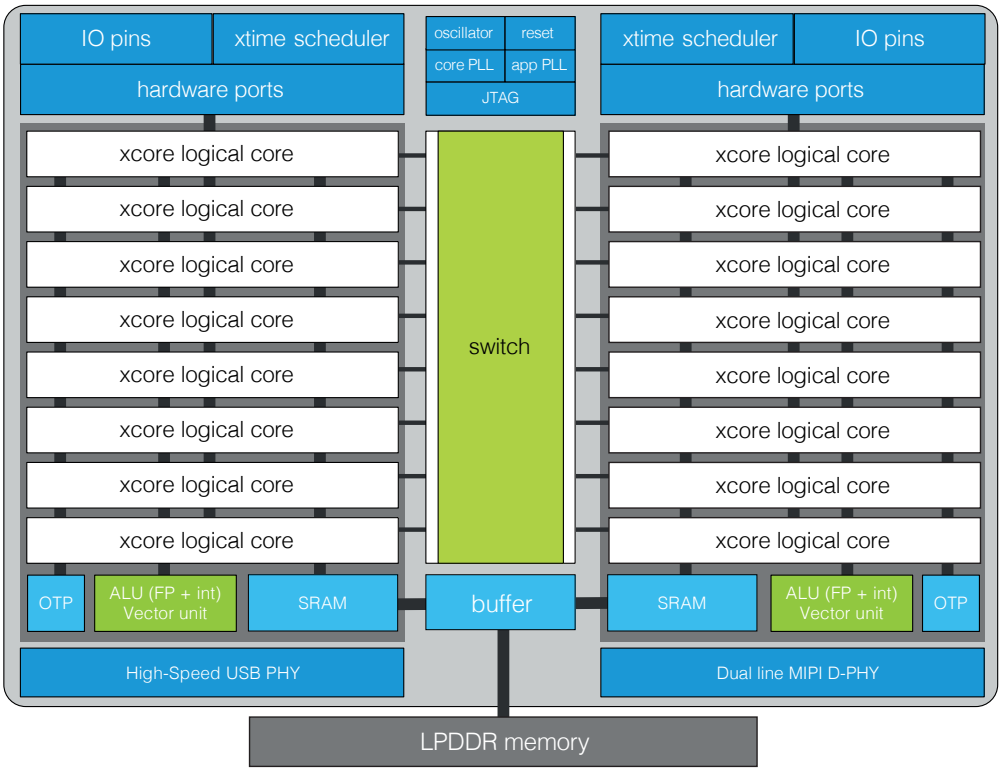
This white paper starts with an introduction to the underlying xcore architecture, a dual-issue RISC style processor running concurrent hardware threads. It then outlines the functions that make xcore.ai particularly special compared to our previous processors: the ability to operate on vectors efficiently. Finally the paper moves on to ways of mapping a problem onto the xcore.ai architecture, and how to extract maximum performance.

2. THE XCORE ARCHITECTURE

The xcore is a scalable, multi-core, general purpose crossover processor[1]. The building block of the xcore is a tile, containing a RISC core with a tightly coupled SRAM. In the second and third generation architecture, each processor has a dual-issue execution unit capable of executing instructions at twice the pipeline clock frequency. Execution is split over up to eight concurrent hardware threads which are each capable of running software tasks that execute IO, control, DSP and AI processing.

xcore implements a standard RISC-like instruction set, with instructions for loading and storing values in SRAM, and instructions for arithmetic on 32-bit integers and (in the third generation architecture) 32-bit floating point values. Additionally, there is a set of instructions that enables software tasks to interact with the IO pins of the device, and to communicate and synchronise with each other with no more than a few nanoseconds of latency.

Enhancements to the xcore architecture in the second and third generations have greatly improved the performance of Digital Signal Processing and Artificial Intelligence (inference and classification). In all cases the instructions and capabilities required have been integrated into the original architectural framework, enabling instantaneous sharing of data between software tasks of the four different processing classes through tightly coupled memory.

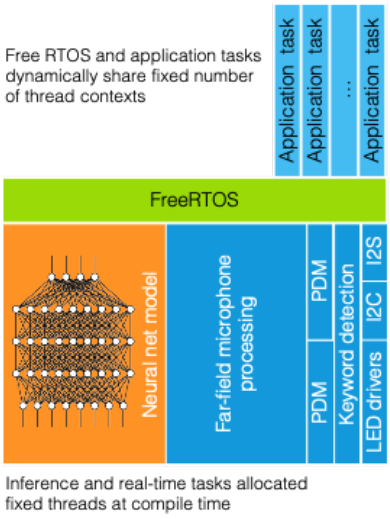


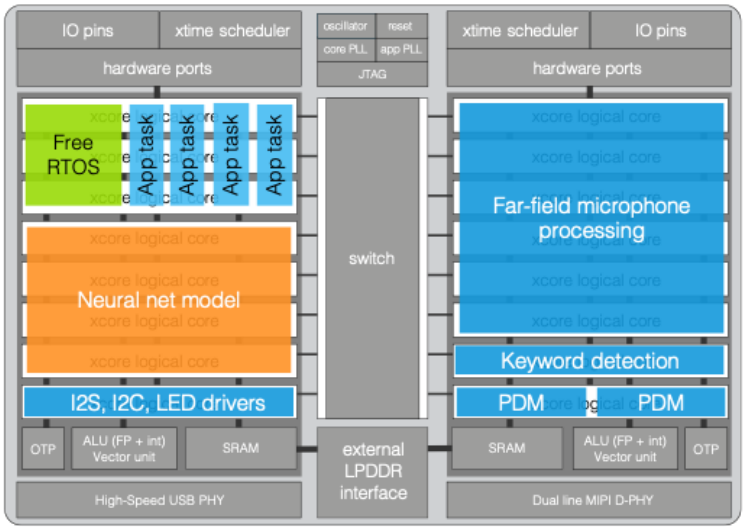
The xcore architecture is scalable. A standard device like xcore.ai contains two tiles (1 MByte of memory, up to 38.4 GMACC/s), two such devices can be put side by side on a PCB and programmed as a single system, providing 2 MByte of memory with up to 76.8 GMACC/s. This

[1] Cross-over processor delivers the performance of an Applications Processor (typically including one or more Cortex A-series processors) and ease-of use of a microcontroller, running a Real-Time Operating System (RTOS).

scalability enables system designers to avoid a common issue with embedded design where a solution doesn't fit into a fixed amount of memory or compute resource provided by the chosen platform. On xcore, this scalability is provided by the xconnect interconnect, providing up to 2 Gbit/s interconnect between adjoining chips.

A typical xcore program executes all control, IO, DSP and AI / inferencing code on the same hardware, employing different software tasks mapped onto one or more hardware thread contexts each. For example, a typical voice program may look something like the following:





The flexible support of all classes of compute means that fully integrated system solutions can be supported on xcore, which drive maximum economy on the bill of materials.

3. XCORE.AI FUNCTIONS

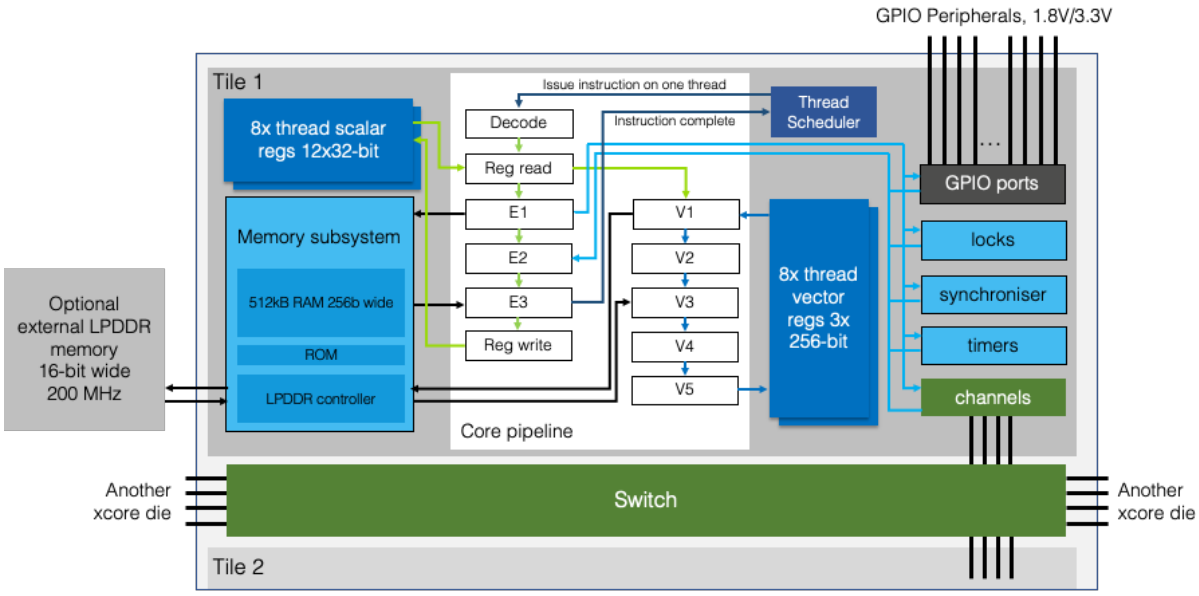
The xcore.ai family of products adds fast vector integer arithmetic to the processor. This is performed in a vector unit that operates on 256 bit wide vectors. Vector arithmetic supports five datatypes:

- 8-bit vector arithmetic, 32 parallel multiply accumulates
- 16-bit vector arithmetic, 16 parallel multiply accumulates
- 32-bit vector arithmetic, 8 parallel multiply accumulates
- 1-bit vector arithmetic (XNOR), 256 parallel "multiply" accumulates
- 32-bit complex vector arithmetic, 2 parallel multiply accumulates

The vector unit has a vector-register set dedicated for each of the eight hardware thread contexts. In a single cycle, an instruction can:

- load a vector from memory, and
- perform a pointwise multiplication with a pre-loaded vector, and
- accumulate all results.

This allows xcore.ai to make the best use of bandwidth of both the memory and the multipliers by loading data whilst the device is simultaneously operating on it.



As the vector multiply operation is executed in a single cycle, a 100 MHz task can achieve a maximum throughput of 100 million vector operations per second. At 8-bit resolution, this is a peak of 3.2 billion loads, multiplies, and accumulates per second (the overhead for instruction pre-fetching lowers this to a sustainable 2.4 billion vector ops/s). With eight tasks active on a core, this sums to 19.2 billion loads, multiplies, and accumulates per second. A typical xcore.ai device consists of two tiles, with two cores that adds up to a grand total of 38.4 billion MACCs per second. Note also that each core can issue instructions on both the scalar and the vector pipeline, enabling very efficient pointer management.

As stated before, each hardware thread is capable of running AI, DSP, control or IO tasks. In an example application one may use one task to obtain data from the sensor, one or two tasks to preprocess the data, perhaps 10 tasks to run a classifier, and then another 3 tasks to interpret the output. If each task is mapped to its own hardware thread[2], this would dedicate 10/16-th of the power of the processor to vector operations, for a maximum of 10/16 x 38.4 = 24 billon 8-bit MACCs/s.

In deep networks, it is typically the inner layers where most of the multiplications occur and can benefit the most from 8-bit arithmetic. This saves memory and increases multiplication throughput compared to 16-bit or 32-bit operations. Higher accuracy layers can be supported through 32-bit and 16-bit arithmetic where necessary.

For even higher multiplication throughput, the inner layers of a network can be designed to use single bit operations (values +1 and -1) – so called Binarised Neural Networks (BNN). In this mode, a device running at 800MHz can reach a total of just over 300 billion operations per second.

The 32-bit complex numbers are primarily included to support computations in the frequency domain, and to support transformations between the time and the frequency domain. The operations supported are complex multiplication and complex addition/subtraction. With these operations, a 512-point FFT can be computed in just over 4,000 instruction cycles.

[2] A 1:1 mapping of software tasks and hardware threads is not necessary, but makes a simpler example.

This totals to just over two instruction cycles per butterfly operation. If the whole chip is dedicated to FFT operations, an 800MHz third generation device can execute more than one million 512-point FFTs per second.

Beside the aforementioned multiply-accumulate instructions, operations are provided for common vector arithmetic (pointwise addition, subtraction, and multiplication), and many operations that support the other layers of an artificial neural network, such as Batch Normalisation or ReLU (Rectified Linear Unit).

4. NEURAL NETWORK MODEL DEPLOYMENT ON XCORE.AI

The development and training of deep neural networks takes place in high level frameworks such as TensorFlow, PyTorch, or MXNet. During training, the model weights and training data are typically stored as 32bit floating point numbers, and floating-point operations are performed by a CPU or GPU. In order to deploy these models on embedded devices with limited resources the neural network is quantised and optimised for the target device.

Typically the network is trained without taking the effect of quantisation into account, and quantisation is performed post-training. This involves the following steps:

- Use of a representative dataset (usually a subset of the training dataset) to estimate distributions of neuron activations.
- Use of the information about the activation distributions to convert the weights and activations from floating point to block floating point (or fixed point) integer numbers;
- If applicable, enforcing effects of saturating integer arithmetic on the quantised activations.
- Optimizing the network architecture for memory footprint and execution time (e.g. by fusing operations/layers for more efficient computations, reordering weights for improved memory I/O utilisation)

Post-training quantisation can have a negative effect on the network's predictive performance, manifesting in lower classification accuracy or higher regression error. The most effective mitigation for this is training the network to be quantisation-aware by quantising the weights and activations of the network during each step of the training, hence forcing it to learn the best weights under the effects of low precision integer arithmetic. In practice, quantisation-aware training is difficult due the lack of support for it in deep learning frameworks (although some promising new features are developed actively).

On the other hand, post-training quantisation is well supported in deep learning frameworks, albeit in a platform-independent manner. To make this task convenient for xcore.ai users, xmos has developed a set of tools that enable users to apply post-training quantisation and xcore.ai specific model optimisations easily. These optimisations include rearrangement of weight/activation arrays for optimal VPU and memory bus utilisation, custom fusing of network layers to reduce runtime memory footprint, and various model transformations to maximize efficiency of parallel execution. The tools are complemented by an extensive neural network library featuring hand-tuned assembly implementations for the most important kernels.

Once the model is quantised and optimised for xcore.ai, there are two ways to deploy it on the chip:

1. Build a TensorFlow Lite micro runtime. The converted model will run out of the box, and can easily be tested and debugged. This supports a wide variety of operators.
2. Build the network inference code "by hand"[3]. This will lead to a smaller program and can have some stronger real-time guarantees. It will also gives an opportunity to optimise memory allocations.

Whilst the architecture supports 1-bit, 8-bit, 16-bit, and 32-bit integers 8-bit arithmetic is generally recommended since it is comparatively dense and widely supported by various frameworks. However, 32-bit and 16-bit arithmetic are particularly useful in the first stage of data preparation, as the feature extractors may produce 16 or 32 bits wide data.

5. THE SIZE OF THE NETWORK

The xcore.ai chip is a low-cost chip designed to be used on the edge; that is, it will be located adjacent to the sensors, interpreting data without communication with the cloud, probably in a very low power environment. The constraints imposed by low-cost and low-power inevitably limits the size of solutions that can be supported.

This section explores these limits, which mostly centre around the on-chip RAM. In xcore.ai, each core has 512 kBytes (that is 524,288 bytes) of single cycle memory. This memory provides a bandwidth of just over 200 Gbits/second per tile (400 Gbits/s for two tiles). The chip has a low power DDR (LPDDR) interface that enables external memory to be connected. However, the interface to external memory is 60x slower at 6.4 Gbits/s. As such, external memory should be used carefully to ensure that overall performance remains acceptable.

Best performance is always achieved when all coefficients of the network, and all intermediate data are stored in the internal memory. If this doesn't fit, adding a second xcore.ai may be the most efficient in terms of performance.

5.1. CONVOLUTIONAL NEURAL NETWORKS

A good compromise for larger models is to keep the intermediate data in internal memory, and to store the coefficients in external memory. For convolutional layers, this approach will result in performance comparable to a model that fits completely in internal memory. The reason for this is that each coefficient in a convolutional layer is used many times, so that it can be pre-loaded once in internal memory, and then used many times. If the coefficients are constant, then these may be stored in flash. If the model is too large for internal memory, more bandwidth is required and/or the coefficients are to be modified, then an LPDDR memory must be used.

Convolutional neural networks are one way in which the vector unit can operate at maximum performance. Given a set of inputs (of any dimensionality), the vector unit multiplies the inputs with a set of coefficients in order to calculate a set of output features. This is performed by loading the input once, and multiplying it in sequence with different coefficients. Since it is a convolutional network, each coefficient is used a large number of times, and hence any cost associated with loading the coefficients from either flash or external memory over a large number of multiplications is ameliorated.

5.2. BINARISED NEURAL NETWORKS

There are two factors that determine the performance of an xcore.ai device: its clock speed, and

[3] The code isn't entirely built by hand, it is a matter of gluing together the various operations.

the amount of on-chip memory. To make best use of both, xcore.ai has an option to use single bit coefficients, to implement what is known as a “binarised neural network” (BNN), also known as an “XNOR” network.

The fundamental operation of a neural network (whether convolutional or not) is to implement an inner product between two vectors: the sum of a large number of products, each product multiplying a data value with a coefficient. In a typical network, both data and coefficient are real values, and the final sum is run through a non linear function. In a binarised network this is the same, with the one exception that all coefficients and data values can only take the values +1.0 and -1.0. This means that the product is either +1.0 (+1.0 x +1.0, or -1.0 x -1.0) or -1.0 (+1.0 x -1.0, or -1.0 x +1.0). The sum of products now becomes a value that is between +N and -N, where N is the number of terms in the sum. A non-linear function is applied to this sum (eg, a ReLU), and then the value is mapped to -1.0 or +1.0 for the next layer.

A BNN has two benefits. First, the multiplication of two 1-bit numbers is trivial compared to the multiplication of two real numbers, and takes orders of magnitude less energy. Second, storing a coefficient of +1.0 or -1.0 takes only a single bit, and hence saves 8x memory compared to storing 8-bit coefficients. It also saves 8x the memory bandwidth to retrieve the coefficients from memory. This means that 8 times as many coefficients and data, or 8-times as much data and coefficients can be stored in the same memory in the same low-cost xcore.ai microcontroller.

When replacing real numbers with the values +1 and -1, one cannot expect the same network performance. Comparing a network with N real coefficients with a network with N binary numbers will see a drop in accuracy. To compensate for this, the network needs to be bigger – perhaps 2x to 3x times more coefficients in a binary network when compared to its 8-bit equivalent. Overall, this means that we can fit a binarised network in xcore.ai with 2.6x to 4x more efficiency than its 8-bit counterpart.

To create a binarised network, the training procedure must be aware that the data and coefficients will be stored as 1-bit values (one cannot simply quantise the coefficient after training!). As such, special quantisation aware 1-bit training engines exist.

6. FLEXIBLE DATA ACQUISITION – THE IO ADVANTAGE

The xcore architecture can interface to a wide variety of data sources. It has built-in physical layers for USB and MIPI (receiver), and using GPIO pins it can communicate over SPI, QSPI, MII, I2S, I2C, PDM and many more interfaces.

This enables the xcore to collect and process data from a wide variety of sensors, including but not limited to:

- Cameras with a MIPI interface or a parallel interface
- Time-of-flight sensors with MIPI or parallel interfaces
- Radar chips with SPI, or MIPI interfaces
- Audio data over PDM, I2S, and S/PDIF.
- Isochronous, Synchronous, or Bulk USB end points.
- Ethernet and WiFi PHYs

Because of the concurrent nature of the xcore processor, the data can be captured, preprocessed and stored by one hardware thread, whilst one or more other hardware threads may be processing a previous frame of data.

In the case of MIPI data, the xcore MIPI interface can unpack common data formats, and perform basic operations such as biasing of the data. All other interfaces are slower than MIPI, and the software task receiving the data can do the basic pre-processing.

If feature extraction is required, then that will again take place in a separate software task, with the option to use the vector unit. For example, xcore.ai can calculate the frequency domain representation of an audio signal in order to generate a set of Mel-frequency cepstral coefficients (MFCC) features; the FFT part of this computation would be taken care of in the vector unit, whereas the final part of the feature extraction (calculating a log) may be performed in the scalar unit.

7. SUMMARY

xcore.ai is designed to deliver an optimum mix of processing for a broad range of IoT applications at a cost and power consumption that is accessible for even the most demanding markets. It combines four classes of processing; AI, DSP, control and IO in a single pipeline structure enabling embedded software designers the maximum flexibility to choose how to partition their systems without the usual concerns associated with fixed performance hardware and/or data-copying.

xcore is fully programmable in 'C' using industry standard tools (LLVM compiler, Tensorflow Lite) and runtime firmware (for example, freeRTOS). XMOS also provides a rich set of libraries, for high-performance DSP and accelerated machine-learning.

xcore.ai heralds an entirely new generation of embedded platform, it's the most versatile, scalable, cost-effective and easy-to-use processor on the market today. With its fast processing and neural network capabilities, xcore.ai enables data to be processed locally and actions taken on-device -within nanoseconds. In the rapidly evolving AIoT ecosystem, this enables manufacturers to build smarter sensing technology that fits seamlessly into our lives.

8. FIND OUT MORE

Designers, investors, media and interested partners can register their interest at xcore.ai to receive all the latest news and announcements.

XMOS will be showcasing xcore.ai at events starting June 2020.

Copyright © 2020, All Rights Reserved.

Xmos Ltd. is the owner or licensee of this design, code, or Information (collectively, the "Information") and is providing it to you "AS IS" with no warranty of any kind, express or implied and shall have no liability in relation to its use. Xmos Ltd. makes no representation that the Information, or any particular implementation thereof, is or will be free from any claims of infringement and again, shall have no liability in relation to any such claims.