GREENWAVES TECHNOLOGIES

GreenWaves Technologies GAP9

Enabling new dimensions in ultra-low power edge computing

January 26, 2022

Version 1.0

Over the past 8 years rich signal processing has been revolutionized by new deep neural network approaches. These have consistently outperformed classical algorithms for classification, detection, and transformation algorithms on input sources such as sounds, images, radar, biosignals and more.

There is a significant amount of interest in bringing these approaches to highly energy constrained edge devices. These include IoT sensors such as image-based building sensors, wearable devices such as wristbands or smart glasses and hearable products such as earbuds and headsets.

Figure 1. GAP9 Use Cases

In all of these products power is extremely limited, either because the product needs to operate for its entire lifetime on a preinstalled battery or because the size of the product greatly restricts the capacity of the battery that can be embedded in it. We see this new need as the third wave of AI, and we have been focused on it for 5 years:

1) The first wave was neural network training and inference on large servers in data centers

2) The second wave was moving AI from cloud to high powered edge devices such as mobile phones, security cameras and edge servers

3) **The third wave is bringing ultra efficient AI inference and signal processing to highly energy constrained devices**

The new algorithms available to designers, be they based on deep neural networks or traditional digital signal processing, make large demands on embedded processors, which **need to deliver bursts of significant processing and meet stringent latency requirements while controlling power consumption**. Moreover, just adding a neural network accelerator does not resolve the intricate power issues in power constrained devices. In a tiny, power

constrained device all the parts of its operation must be designed and optimized to save energy.

The designers of these new intelligent, power constrained devices come from diverse backgrounds. The data scientists are working with tensor processing frameworks such as TensorFlow or PyTorch. The DSP algorithm designers are working in analytical packages such as MathWorks while the embedded application designers expect familiar C/C++ tools. In many cases, and particularly with data scientists, designers may not be familiar with the constraints of deeply embedded systems. Next generation processors must be easy to use and provide the tools that developers are expecting so they can keep their way of working while ensuring that code produced is as efficient as possible. They must also ensure that it is easy and cost effective to integrate legacy code for elements such as communications protocols that are less demanding from an energy perspective.

All of this needs to occur in a highly fluid environment where state of the art in signal analysis and inference using machine learning is changing every week! Flexibility to adopt the latest available techniques is key.

GreenWaves' GAP9 processor has been designed from the ground up to address this multi-dimensional computing requirement. While GAP9 shares some system components with traditional microcontroller units (MCUs) its architecture is quite unique, enabling a revolution in the performance of extremely energy constrained devices. In this document we look at GAP9 from an application perspective. What is different? How does this help the device developer? We will also give an overview of the tools that are provided in the GAP SDK and how these ease development and allow designers with different backgrounds to exploit GAP9 to the maximum.

The building blocks

GAP9 is made of 3 fundamental building blocks, an MCU type controller that we call the Fabric Controller, a smart peripheral controller (the µDMA) integrating a sample by sample audio Smart Filtering Unit (SFU) and a parallel compute engine that we call the Cluster. The Fabric Controller (FC) is responsible for managing peripheral devices and controlling application execution on GAP9. The µDMA allows autonomous, low energy peripheral management and on the fly calculation through specialized processing blocks such as the SFU for ultra-low latency tasks such as Adaptive Noise Cancellation. The cluster provides a flexible, on demand, high performance programmable calculator for any task that demands significant compute resources such as Digital Signal Processing or Machine Learning algorithms.

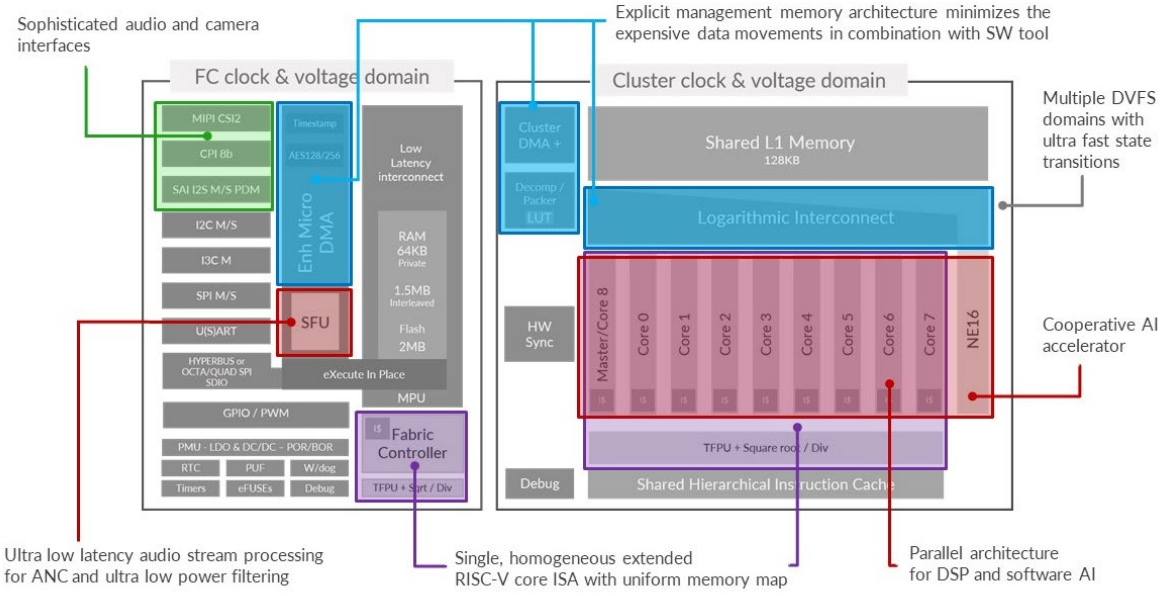

Figure 2. GAP9 Architecture Overview

GAP9 makes extensive use of Dynamic Frequency and Voltage Scaling (DVFS) in multiple different zones (known as Domains) of the chip. This allows elements of the chip to be entirely switched off when not in use but also the actual capabilities and energy consumption to be precisely tuned to the requirements of the task being executed.

When a voltage domain is active GAP9 uses automatic clock gating to stop clocking smaller functional blocks when they are not in use further reducing power consumption.

GAP9 has a rich set of peripheral interfaces including a 2 lane CSI2 interface, parallel camera interface and 3 Serial Audio Interfaces capable of handling up to 16 TDM channels and incorporating 3 input and 1 output PDM channels per interface.

The final building block for GAP is an SDK, which follows two fundamental philosophies:

- Use tools that are familiar to the targeted developer
- Avoid black boxes that the developer cannot properly debug

Let’s look at these elements in more detail.

The Fabric Controller

The FC contains a single core that is responsible for coordinating the activity on GAP9. All GAP9’s cores in the FC and cluster are based on the RISC-V ISA extended with custom instructions for operations such as MACs, zero-cycle loops, saturation and clipping operations, bit level arithmetic and lightweight 8-bit and 16-bit vector instructions. The use of a single core design simplifies development - only a single compiler toolchain is required.

All the cores in GAP9 include a Transprecision Floating Point Unit handling IEEE 32-bit, 16-bit and BFloat16 floating point numbers. The vector unit in the cores can handle vectors of IEEE-16 or BFloat16 operands.

The FC area of the chip also contains the main memory made up of 1.5MB of RAM, which can optionally be retained in blocks when in low power modes and a 2MB area of non-volatile memory using state-of-the-art eMRAM technology. eMRAM, as opposed to Flash memory, provides high speed read and write access, which gets close to the performance of RAM. It's ideal for storing filter coefficients for neural network tasks. It also can be used to store firmware enabling reduced system cost when no external memory is necessary.

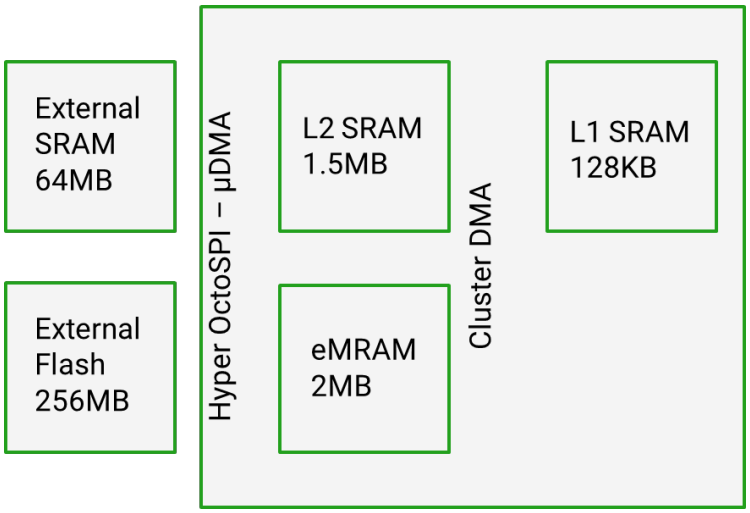

Figure 3. GAP9 Memory Architecture

GAP9 has two[1] external memory interfaces that can be individually configured to support Octo- or Quad-SPI, HyperBus or SD memories. The total bandwidth of each external memory interface is 370MB/sec. GAP9 incorporates a virtual memory mapping capability that allows both external and internal memories to be mapped into GAP9's memory space with caching in the L2 area. This allows execute-in-place execution of code contained in external memory.

All of the power supplies and oscillators necessary for GAP9 are controlled by the FC via an integrated power management system. The FC can enter a deep sleep state consuming a few µA of current from which it can wake up in a few milliseconds. It also has a low power state consuming under a mW from which it can quickly configure peripherals to start acquiring data before the chip is fully awake.

[1] Only 1 is available in WL-CSP package

The µDMA

The FC talks to peripherals via an intelligent peripheral controller that we call a Micro-DMA (uDMA). This differs from a classic DMA unit in that it is not executing transactions over a shared system bus. The uDMA has dedicated channels to the L2 memory and a series of dedicated DMA channels, individually connected to and controlled by peripherals. This reduces contention between peripheral dataflow and internal activity improving performance and reducing power consumption.

The uDMA is capable of autonomously executing complex transactions with an ability to process information passing through it on-the-fly. This processing capability ranges from timestamping information arriving from multiple interfaces and on-the-fly AES 128/256 encryption and decryption to complex, ultra-low latency sample by sample audio filtering that can be used to implement state-of-the-art Active Noise Cancellation. We call this audio filtering unit the Smart Filtering Unit and will explain it in more detail later in this document.

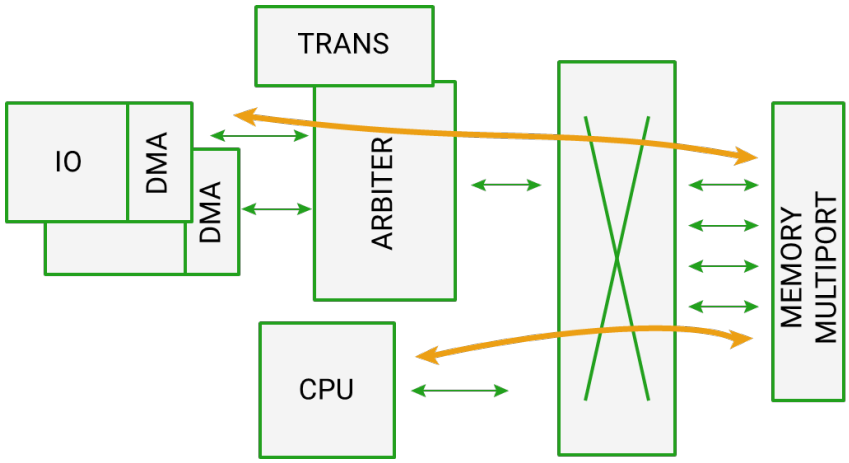

Figure 4. MicroDMA Architecture

The uDMA reduces the power consumption of data acquisition and transmission by reducing the need for (or entirely eliminating) intervention by the processing cores in GAP9. It also enables ultra-low latency on-the-fly data processing even directly between two interfaces.

The MicroDMA is connected to a rich set of interfaces including ones specialized for sound sources and sinks (SAI) and cameras (MIPI CSI2 & CPI) and general purpose interfaces such as SPI, UART, I2C & I3C and GPIO.

The Cluster

GAP9's cluster includes 9 RISC-V cores (identical to the FC core) coupled with a shared L1 memory area. GAP's cluster efficiently exploits the parallel nature of most DSP and Machine Learning algorithms to allow them to be run at a lower clock rate for the same performance. This, in turn, allows GAP9's core voltage to be reduced bringing a quadratic reduction in energy consumption. The cores all have their own program counter and can run different code giving a large choice in parallelization strategies.

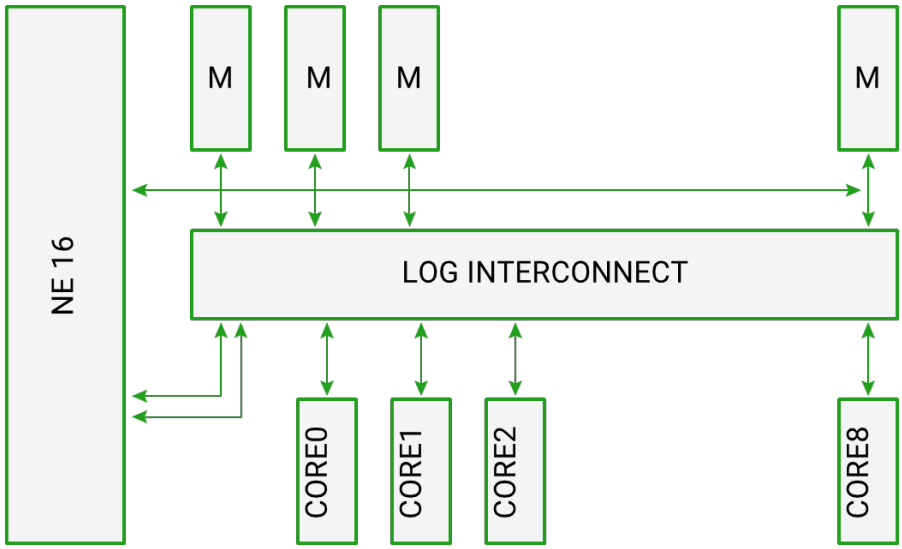

Figure 5. Cluster Architecture

To efficiently implement parallel algorithms all the synchronization primitives such as forks, joins & semaphores are implemented in hardware. This allows a task to be forked to all cores in under 5 cycles. When individual cores in a group complete a task, they enter a wait state in one cycle where they are immediately clock gated. When all the cores complete in one cycle they all start running again. This allows parallelism to be exploited in algorithms that require frequent synchronization between different threads.

GAP's cluster architecture allows dedicated hardware accelerators to be combined with the cores. GAP9 includes a dedicated accelerator, Neural Engine 16 (NE16), for the streamed multiply/accumulate (MAC) operations that characterize neural networks. NE16 has the same shared access to the L1 memory as the cores and communicates with them via a sophisticated event based controller. This allows the energy and speed benefits of a dedicated hardware block to benefit from the flexibility of general purpose cores. NE16's design can be focused on delivering the best energy performance for the most critical operations without sacrificing functionality filled in by the cluster cores.

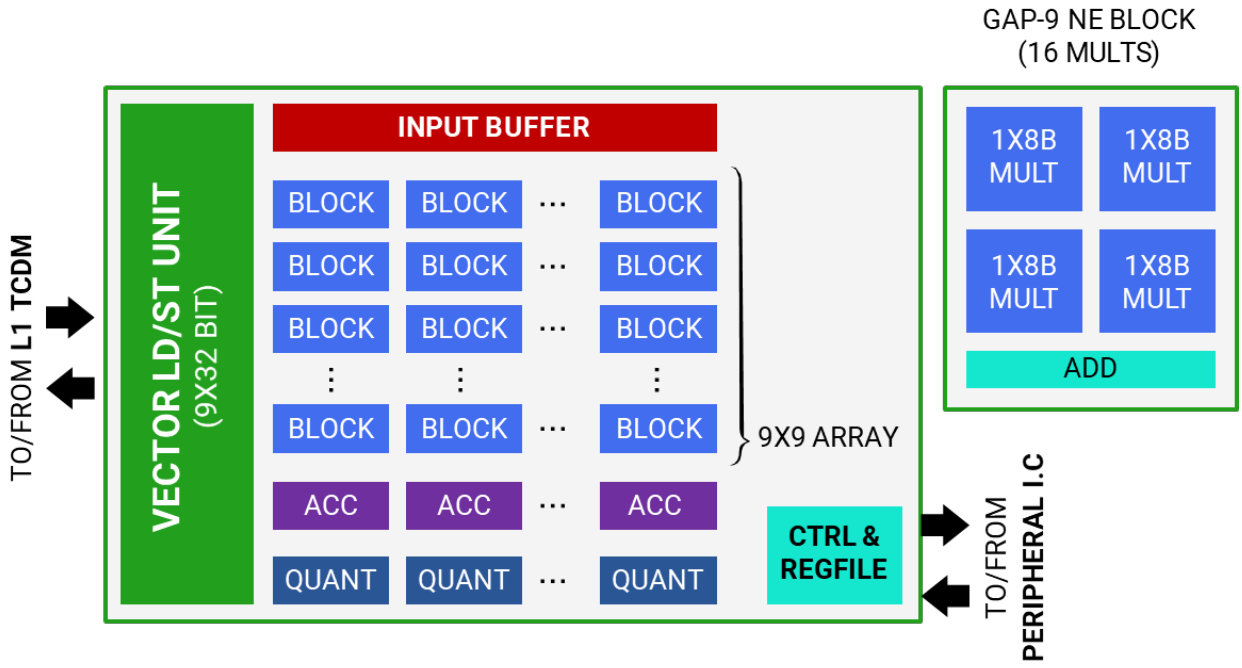

Figure 6. NE16 Architecture

NE16 is however highly flexible, it can handle CNN, RNN or vector/matrix multiplications with 16- or 8-bit features and from 8 to 2 bit weights. It natively supports asymmetric, scaled quantization. The varying precision can be allocated to individual layers. If float execution is needed for some layers these can be executed by software kernels on the cluster cores with vectorized IEEE16 or BFloat16 features and weights. Our kernel library includes a wide range of neural network and signal processing algorithms fully optimized for the cluster, ready for use.

This tightly coupled combination of a programmable multi-core compute cluster and dedicated hardware accelerator is unique in the market and is flexible enough to adapt to the latest, state-of-the-art signal processing and AI techniques. We have found that while it is possible to produce architectures that give greater efficiency on some synthetic example models, in real life applications the cluster's flexibility provides the optimal balance of network accuracy and energy performance.

Explicit Memory Movement

Instruction and data movement incurs a large energy cost. GAP9's cluster incorporates a hierarchical instruction cache that ensures that a sequence of instructions used by more than one core is only loaded once; however, GAP9 incorporates no data caching.

All the cores in GAP9 see exactly the same memory map; however, to improve performance, data movement should be hidden behind processing as much as possible. Classically this is achieved through some form of data cache. However, for GAP9's workload, essentially

streams of images, sounds, and so on whose dimensions are already known at compile time a data cache would be highly inefficient. The expected cache hit rate, how often loaded data would actually be in the cache, would be in the order of 30%. This would result in stalls and misloads causing loaded data to be thrown away - clearly a useless energy cost. In GAP9 sophisticated Direct Memory Access (DMA) units are used to explicitly move data.

While this cuts down on energy consumption, the problem of correctly positioning data elements for an operation or sequence of operations and properly scheduling data movement between external L3 and internal L1 and L2 memory areas to bring data as close as possible to the processing element using it is difficult to solve manually.

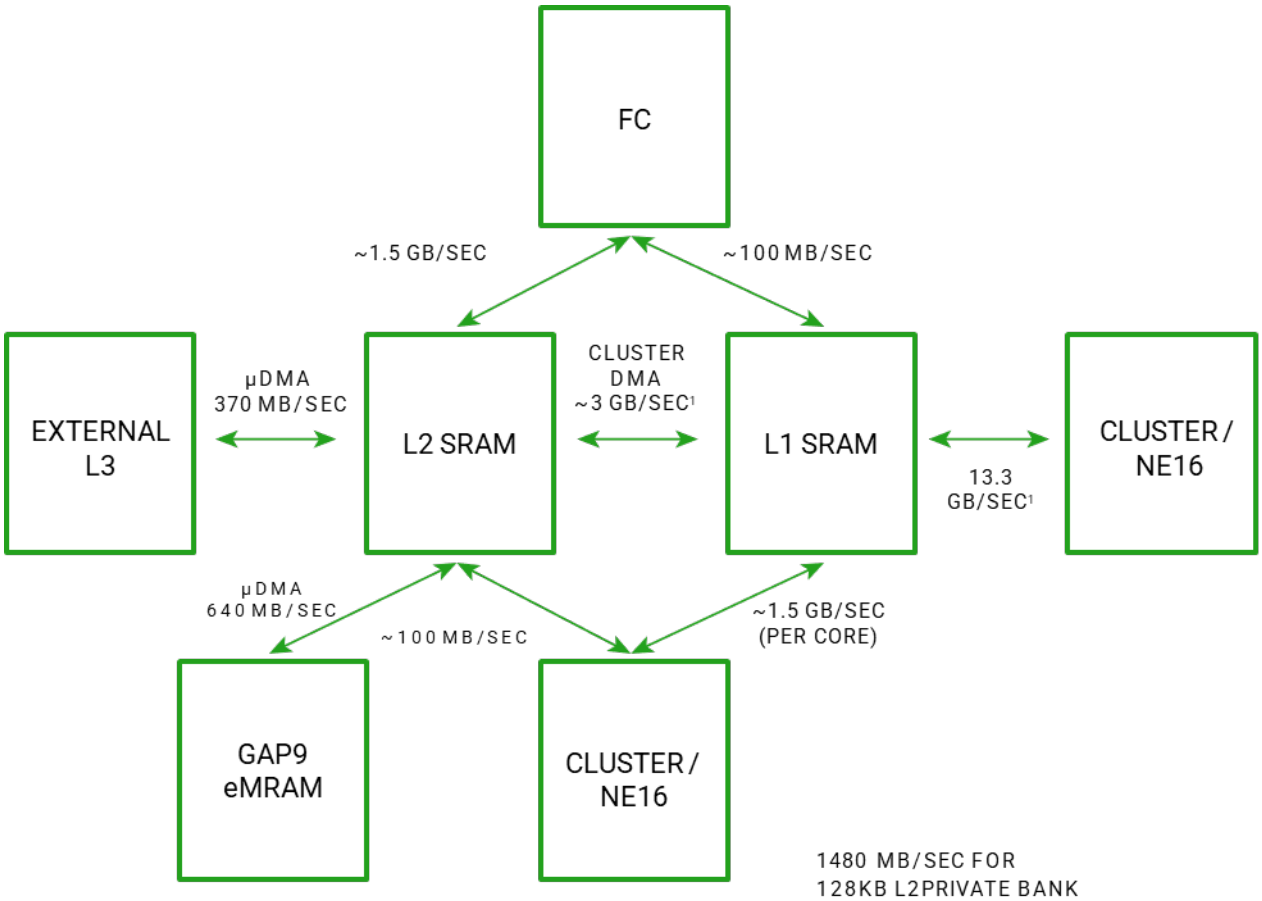

Figure 7. GAP9 Memory Bandwidth

We resolve this problem using a sophisticated optimization and code generation tool called the GAP AutoTiler. The AutoTiler takes as input a series of models of a computation graph (this could be a neural network graph for example) and the kernels that implement its operations and configurable memory constraints and discovers a solution for parameter placement and movement that minimizes the amount of times a piece of data is moved while ensuring that it is in the cluster L1 when it is needed.

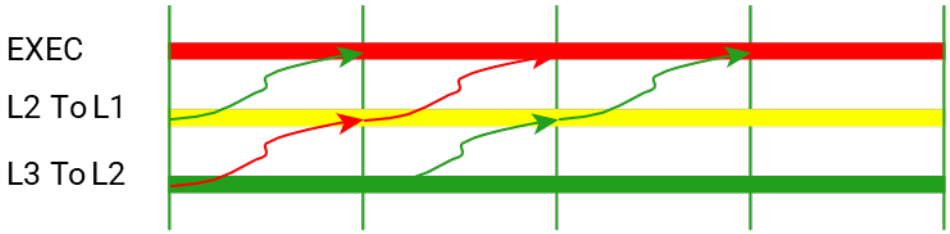

Figure 8. Overlap mem movement with computation

Once the AutoTiler has found a solution it produces human readable C code that does all the data movement and calls the kernels in the correct sequence to implement the graph. It produces three C functions that are called by the programmer: a graph constructor, a graph runner and a graph destructor.

```
static void Conv_Layer0
(
    signed char * In,       // input L3 vector
    signed char * Weights,  // input L3 vector
    signed char * Bias,     // input L3 vector
    signed char * Out,      // output L3 vector
){

    //tile sizes of In, Weights, Bias computed offline
    //L1 buffer allocated to handle double buffering
    //  two L1 memory buffers for double buffering

    uDMA load first tiles to L2 memory buffer

    DMA load first tiles to L1 memory buffer

    for any tile of In, Weights, Bias tensors:

        uDMA load next next tiles to L2 memory buffer

        DMA load next tiles to L1 memory buffer

        ParConv() on L1 tile
        ParReLU() on L1 tile
        ParPool() on L1 tile

        DMA write results (Out) to L2

        uDMA write prev results to L3
}
```

Figure 9. Generated user kernel pseudocode

The AutoTiler can be used by an algorithm designer to automate the production of code for all data movement allowing them to concentrate on the parallelization of their algorithm as though all data was available in the L1 cluster memory. It also makes up a fundamental building block of GAP's neural network toolchain that can take graphs in ONNX or TensorFlow Lite and automatically produce highly optimized C code.

One of the unique elements of the AutoTiler solution is that parameters are moved on demand to the cluster (mostly hidden behind computation) which means that there is no 'load' time for a network. The network constructor created by the AutoTiler allocates L1 and L2 memory for the

graph and can preload certain parameters for faster execution but the memory used by it is entirely configurable so multiple models can run side by side with minimal to no ‘start up’ time.

Familiar Development Tools

The AutoTiler is one in a series of tools that makes up the GAP SDK, available from our GitLab repository. Since GAP is based on RISC-V cores the RISC-V GCC compiler is used to compile code for GAP9. There is no assembly language used, no esoteric and difficult to learn languages or tools. As much as possible we try to support a path from the tools most familiar to our audiences: TensorFlow and Pytorch for machine learning, Matlab for signal processing, C and C++ for embedded development. Along with well known RTOS environments like FreeRTOS for application control.

The GAP9 SDK also includes a simulator, GVSOC, that allows code to be run on the developers workstation. Full visibility of the activity inside the simulated GAP9 is provided through signal traces in the GAP profiling tool. Many peripherals are simulated by GVSOC allowing complete applications to be run. This provides truly unique observability and allows problems to be tracked down quickly reducing development time.

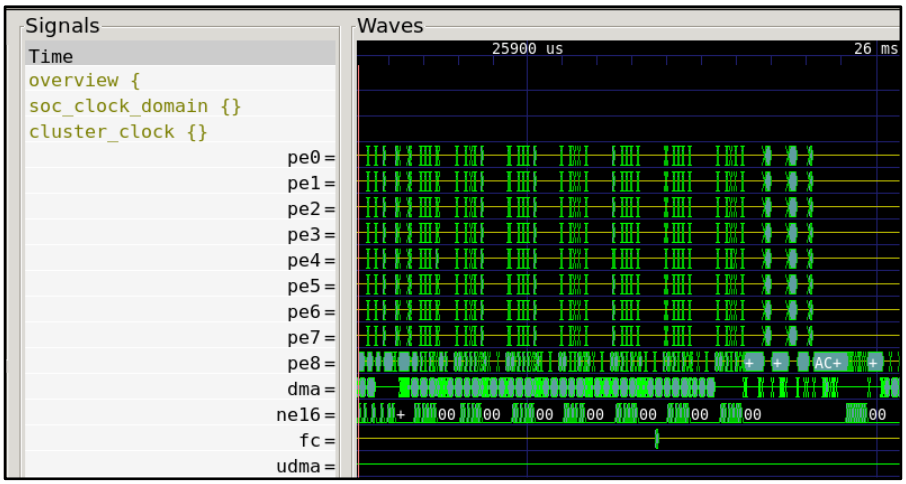

Figure 10. View of signals inside GVSOC simulation

Developing Machine Learning Applications with GAP9

Machine learning engineers generally like to start working with well known neural networks and then tune them for their applications. This can be a problem with energy constrained devices since often they incorporate dedicated hardware accelerators that lack the flexibility to implement these networks precisely.

The GAP9 SDK includes a flexible translator called NNTOOL that can take standard TensorFlow Lite or ONNX graph descriptions and produce highly optimized C code that executes the graph using the extensive GAP kernel library. NNTOOL takes the graph as input and produces a model for our AutoTiler tool that generates the C code.

NNTOOL can cope with quantizing the graph or using quantization information already present in the exported graph. It optimizes the graph so it matches the available kernels and can be used to analyze performance and accuracy of the graph in its translated form. NNTOOL supports selective quantization of different layers of the graph with a choice of different bit widths (including sub byte quantization) in both fixed and floating point formats. This multi-precision capability ensures that accuracy can be maintained in even the most demanding networks.

Our NNMenu repository contains a wide range of example graphs and applications that provide an excellent starting point for a developer working on a new project. It includes open source licensed implementations of graphs for object or person detection, attention detection, face identification, sound analysis and more.

Some of those networks and their performance on GAP9 is shown in the table below. The MobileNet performance is for maximum image size 224x224 and no channel scaling. The SSD performance is for a 300x300 input and no or 75% channel scaling.

CNN	N.MACs [M]	CYCLES [M]	MAC/CYC	LATENCY [MS]	POWER@1fps [mW/fps
MobileNetV1	574	14.80	38.83	37.00	1.01
MobileNetV2	307	12.40	24.80	31.00	0.85
MobileNetV3	213	9.10	23.48	22.75	0.62
ResNet18	1815	62.50	29.90	156.25	4.28
KWS (tinyML)	2.70	0.24	11.40	0.60	0.02
VWW (tinyML)	3.28	0.41	8.29	1.02	0.03
SSD MobV1 1.0	1241	39.40	31.63	98.50	2.69
SSD MobV1 0.75	731	24.90	29.45	62.25	1.70
Wav2Letter (ARM)	3494	192.50	18.14	481.25	13.17
YoloX-NANO	30	5.57	5.40	13.93	0.38
YoloX-NANO (HARD)	30	3.33	9.50	8.33	0.23

Figure 11. GAP9 NN Performance

Next generation hearables will need to combine algorithms such as Active Noise cancellation demanding ultra-low latency, sample by sample filtering, 3D sound algorithms demanding complex frequency domain filtering and room physics simulation and noise suppression or acoustic environment detection requiring state-of-the-art neural network technology. And all of this needs to happen at an energy level that doesn't immediately drain a tiny battery. The GAP9 cluster covers the frequency domain filtering, physics modelling and neural network applications; however, sample by sample filtering at ultra-low latency requires dedicated hardware.

This is where the GAP9 Smart Filtering Unit (SFU) comes in. The SFU provides a flexible block that can implement sample by sample filtering on audio streams coming from and going to PDM Audio interfaces, peripheral interfaces (including SAI, SPI, I2C) and L2 memory. Highly configurable, dedicated hardware implements a wide range of filtering types along with splitters, mixers, limiters and so on. The SFU integrates multi-channel / multi-tracker asynchronous sample rate conversion and flexible PDM modulator and demodulators. All these blocks can be combined into entirely user-defined multiple audio filtering graphs which can be run simultaneously. Furthermore, filter coefficients can be dynamically updated without stopping the graph and individual graphs can be loaded and unloaded.

The SFU's internal precision can be set to two levels: 32 bit input + 32 state into 64 bits or 32 bit input + 64 bit state into 96 bits. This is considerably more precision than most DSPs, which simplifies filter design. Also, since the SFU is a sub-system that you configure and then run, there is no instruction load penalty since there are no instructions to load. This has both a performance and energy benefit.

We see two primary scenarios for using the SFU: as a continuous filter graph between two PDM interfaces for applications such as ANC and as a filtering coprocessor for the FC and Cluster between blocks of samples in L2 for equalization and other sound effects.

The SFU enables real time operation on individual samples at a 768KHz sample rate, a structural latency of 1.35µS. Since it sits in its own DVFS domain, its performance and energy consumption can be precisely tuned to the task it is executing.

GAP9 incorporates 3 sophisticated SAI interfaces supporting up to 16 TDM inward, outward or forwarded slots on each interface. Each interface also incorporates two 2 channel PDM interfaces selectable as input or output. Each channel can be put into a clockless differential PDM mode which can be directly connected to an analog, minimal latency sigma-delta amplifier.

The GAP SDK includes our AudioTools framework providing a familiar interface via Mathworks plugins for filter design and our GraphTool utility allowing integration of SFU filters, Cluster Filtering and Neural Networks and graph elements running on the FC.

Our neural network development toolchain includes support for state-of-the-art networks including recurrent elements and temporal convolutional networks.

We also have tight partnerships with companies working at the forefront of audio algorithmic design for noise cancellation and suppression, speech enhancement, 3D sound and more.

Security

Security is an important consideration for any edge device. GAP9 incorporates several features to ensure that firmware is not tampered with and that confidential algorithms and neural network parameters are protected.

GAP9 incorporates a bank of one time programmable eFuses that can be used to store keys and to enable and disable firmware features. GAP9 protects system memory with a memory protection unit and hardware machine and user level privilege modes. It also integrates a physically unclonable function (PUF) hardware block that can be used to generate a unique chip ID and seed a strong random number generator for cryptographic functions. Finally the MicroDMA incorporates a hardware AES-128/256 encryption and decryption engine.

These hardware features enable the implementation of secure boot of encrypted firmware, on-the-fly decryption of network parameters fetched from external memory and compartmentalized application code execution.

Summary

Machine learning and particularly deep neural networks are being used in many applications. GAP processors focus on bringing significant capabilities to highly power constrained applications. Power constrained applications are generally devices that have to last for years on battery like IoT sensors or devices that have a very small battery such as hearable or wearable products. GAP processors are being used in smart building sensors that can detect people’s location in images, wearable devices that locate objects in images and respond to speech commands and hearable products that combine features such as sound environment aware noise reduction or 3D sound rendering. In all these applications it is GAP9’s unique combination of simple to use, flexible, ultra-low power and latency processing capability that makes what seemed impossible in battery operated devices possible, now.