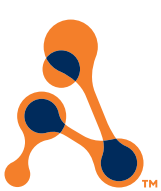
WHITEPAPER

AI on the Edge: Don't forget the Memories

Bard Pedersen, Adesto Technologies Corporation



Adesto

www.adestotech.com

Introduction

As consumers get used to having their invisible servants available within shouting range at all times, the demand for electronic systems that can deliver this service is increasing exponentially. Hidden in our walls or lying on our coffee tables are gadgets with surprisingly high levels of sophistication, capable of detecting and interpreting a wide range of human speech. Until recently, the gadgets themselves were just messengers, incapable of doing anything but detecting the simplest of commands without relaying everything they heard to their overlords in the cloud. And yet, they required so much power that they needed to be wired to the nearest power source continuously. The next generation of gizmos are expected to do a lot more on their own, without calling for help, while just sipping a little power from a battery. This calls for new ways of building these devices and we begin with the embedded systems within.

1. What is an Embedded System Anyway?

An embedded system is typically a computer-based system designed for a specific task. It can be as simple as the controller operating a washing machine or a room thermostat, or as complex as a network router or an industrial machine controller.

Embedded systems are generally not set up to run a wide range of different software applications the way a standalone PC is, they typically run specific firmware designed for the system at hand. A cell phone is an example of an application that moved from being an embeddded system to now having all the charactristics of a general purpose computer.

Still, embedded systems consist of the same basic units as any other computer:

- Central Processing Unit (CPU), the brain of the computer
- Code storage (memory for storing programs/applications/instructions)
- Data storage (memory for storing received and calculated data)
- Input units (which can range from simple switches and sensors to video cameras)
- Output units (from LED lamps to printers, video screens and RF links)

Code storage connected to the CPU can be either volatile or non-volatile memory. If a volatile memory is used for code storage, like the Random Access Memory (RAM) memory used on most PCs, the code must be copied from a non-volatile memory before the code can be run.[1] A hard disk or solid-state drive serves this purpose on a PC; in embedded systems it is usually a type of flash memory.

Data storage can be temporary storage of internal data used by the CPU, or it can be data recorded by the system and stored for later use. It can also be permanent data like graphical objects or configuration/calibration data; this type of data is handled similar to the code storage. Short-term temporary storage will typically use volatile memory, and data that must be available after a power-down / power-up cycle will use non-volatile memory.

2. Code Storage in Embedded Systems

Over the last decade, system designers have only been given a few options for code storage when building an embedded system. For most systems with large amounts of code, the choice has mainly been between various types of external boot memory or using MCUs with embedded flash. But now, with the advent of modern XiP systems, designers have another option.

2.1 Shadow Flash (or Boot Memory) Systems

In a system using shadow flash, the non-volatile code storage is an external memory device. For small memories under one megabit, this is typically an EEPROM device. For devices from one megabit to one gigabit, the dominant memory type is serial NOR flash. Above one gigabit, NAND flash memory types dominate.[2]

Before the CPU can start running, its code is copied from flash to RA data storage. Small systems use on-chip SRAM, while larger systems use external DRAM or PSRAM .[3]

Larger systems may use a simple file system or a more complex operating system (e.g. Linux) to load files from the flash as they are needed. The flash device may in this case be much larger than the RAM. This is similar to what is done on a PC with RAM and disk drive, just smaller.

2.1.1 Shadow Flash (or Boot Memory) + Embedded SRAM

This is the typical setup for small, fast SoC (system-on-chip) as shown in figure 1.

At power-up, all the code is copied into RAM, and the NV memory is typically not read again until next time the system powers up. The system will have additional RAM for data storage.

If the system does not require Over-The-Air (OTA) updates, the size of the flash image is typically the same size as the code storage RAM. If the system requires OTA, the flash size will typically be 2x, 3x or even 4x the RAM size, so multiple versions of the code image may be stored. The flash memory in such systems will typically hold the original factory code image, the currently used code image and space for storing an incoming OTA updated image.

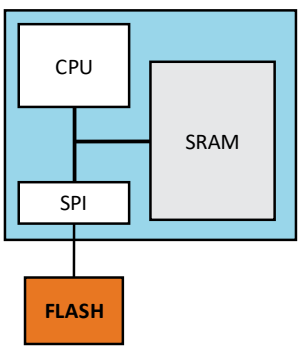


figure 1 - Shadow flash + Embedded SRAM

This configuration is cost-effective for devices with a lot of logic and relatively small memory, but fixed memory size and power consumption need to be considered.

This memory configuration has many advantages. There is no NVM on the SoC, so the chip can be built in an advanced geometry on a standard logic process. The CPU can take advantage of the small geometry and run very fast and the SRAM can be read at the speed of the CPU. It also means that this system achieves the best dynamic power consumption possible (least amount of power required to get the job done).

This configuration is cost-effective for devices with a lot of logic and relatively small memory. The SoC can be built in very deep process nodes, which means the die can get very small. And as it is using a standard logic process, there is no cost adder for NVM.

Of course, there are some disadvantages to this configuration as well. Cost, fixed memory size and power consumption need to be considered.

Cost quickly becomes a problem. On-chip SRAM is the most expensive memory available (in cost per Mbit). SRAM is a relatively huge memory cell (typically 100 F^2), often 8X larger than stand-alone NOR flash and 25X larger than NAND flash.[4] And you have to pay for the same amount of external flash on top of that. This configuration therefore limits itself to small systems only, typically up to a few Mbytes.

Another problem with on-chip memory is that the memory size is fixed. A different memory density means a new device has to be designed and put into production, and the cost of this can be substantial in fine geometries.

Power consumption is also a problem. Devices running from on-chip SRAM have low dynamic power compared to the alternative options, but they suffer greatly from increased standby power (the amount of power consumed when the device is not doing anything useful). SRAM will lose its memory content when power is turned off, so it either must be kept alive all the time, or the SRAM must be reloaded every time the device wakes up from sleep.

The total standby power varies depending on how often the system has to wake up and how much memory has to be reloaded each time it does so. But in either case, the total standby energy consumption may end up being much higher than the active power consumption for devices that spend a lot of time in sleep. An additional drawback when going for the reload option is that the time it takes to reload the memory may make the system response time too slow.

Typical applications using this configuration are Wi-Fi, Bluetooth and other RF SoCs, which take advantage of the low power consumption and high execution speed the on-chip SRAM offers. Many of these applications can be made with a combination of on chip SRAM + external boot flash, or they can be built with embedded flash.

2.1.2 Shadow Flash (or Boot Memory) + DRAM / PSRAM

When memory densities get too large, the cost of SRAM gets too high. In comes DRAM to the rescue. Figure 2 shows a configuration that swaps out expensive on-chip SRAM with off-chip DRAM or PSRAM. On the plus side, DRAM provides much lower cost per Mbit than SRAM. This is both due to the much smaller memory cell (6-8 F^2 for DRAM compared to >100 F^2 for SRAM) and due to the cost of DRAM wafers being less expensive than logic wafers in the same node.

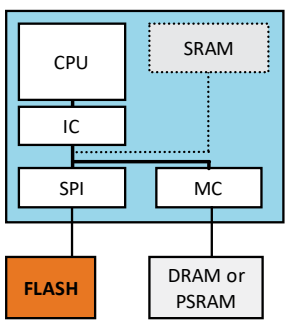


figure 2 - Shadow Flash + DRAM / PSRAM

DRAM provides lower cost per Mbit than SRAM but you may pay for memory you don't need and it requires continuous refresh.

But as always, nothing is perfect.

The lowest cost DRAM is approaching a minimum size of 1 Gbit; smaller devices are usually made in an older process and do not cost much less. Therefore, even if your system does not need that much memory, you end up paying for it.

Another problem with DRAM is that it requires continuous refresh.[5] The charge of the DRAM cells will otherwise leak out within a fraction of a second. This means that DRAM needs a memory controller that will continuously monitor the memory. It also means that DRAM has higher power consumption than any of the other code storage options, as every refresh cycle consumes some power. On top of all that, DRAM has destructive read. Just reading the memory contents of a cell will erase it, and it has to be written back afterwards.

Pseudo-SRAM (PSRAM) makes life easier for system designers. All the difficulties of handling DRAM are hidden internally in the memory chip, and the interface is more or less the same as for a static SRAM device. Of course, the power consumption is still high as the refresh cycles are required. In terms of cost, PSRAM is typically more expensive than DRAM, but not as high as SRAM.[6]

External SRAM costs about the same as embedded SRAM (it is using a similar process), so this is not a more cost-effective option for code storage. It may be used if an application requires slightly more SRAM than the amount offered on the MCU/SoC. If large amounts of additional RAM are needed, PSRAM is typically preferred over SRAM due to cost.

And speaking of cost: You will have to pay for every byte of DRAM or PSRAM used for code storage at least twice. An external flash device of at least the same size as the DRAM/PSRAM is required to provide non-volatile storage. A system using an operating system like Linux may have a much larger flash memory, just like a PC typically has a disk drive that is much larger than the system RAM.

2.2 Embedded NVM

Being able to add rewriteable non-volatile memory to the same die as the CPU and SRAM is probably one of the most important innovations happening to the MCU industry over the last quarter century. Embedded NVM for code storage makes life easier for system designers in many ways. This is the preferred code storage option for a wide variety of MCUs and SoCs from almost all device manufacturers.

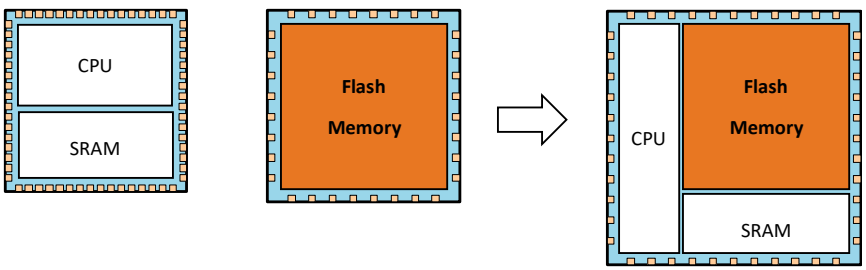


figure 3 - Moving from external flash to embedded NVM memory

The memory type used for code storage is typically embedded flash (e-flash); for small devices under 128kbytes (1Mbit) embedded EEPROM may be used. Newer NVM types like FRAM and MRAM are also being used, but these do not represent a large portion of the market.

In the configuration shown in figure 3, the non-volatile code storage is on the same die as the CPU, which saves both time and power.[7] The physical distance between the memory and CPU is short, the address and data do not have to go through two (or more) sets of I/O pads, and the bus width is not constrained by how many pins you can use on a package. The internal memory interface of new embedded NVM devices typically operates on a bus width of 32, 64, or 128 bits.

Code is either running directly from flash, or the device is using a small read buffer or instruction cache (IC). Except for the cache memory, RAM is typically only used for data storage. An external EEPROM or flash device may be used for OTA updates and/or data logging. If the system is implementing OTAs, the external flash size will typically be 2x, 3x or 4x the internal flash size, so multiple versions of the flash image can be stored.

Embedded flash offers many more advantages to the system.

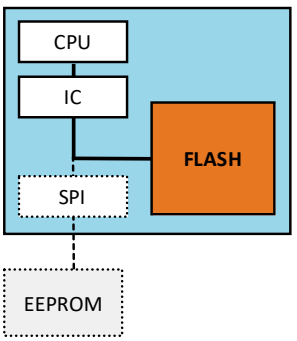


figure 4 - Embedded NVM

This solution offers the best standby power. However, embedded flash can be expensive.

This solution offers the best standby power, as the system may be turned (almost) completely off while still offering fast startup from sleep. Sleep currents may be less than 100nA while at the same time having wakeup times of a few µs. And the code is ready to run at wakeup since there is no reloading of RAM before execution can start. (The system may reload some configuration settings and a cache line or two, but rarely more than that.)

As with any other memory configuration, embedded NVM for code storage also has its limits.

Again, cost quickly becomes the limiting factor. Embedded flash is an expensive process. A much higher number of masks are needed for implementing flash compared to standard logic. A typical number is 10 or more extra masks, which leads to 20-30 additional processing steps. The increased production cost due to the extra process steps represents a larger problem. The wafer cost adder may end up being over 40% compared to standard CMOS. The cost adder varies with the technology node and tends to be higher for smaller geometries. This makes everything on the MCU more expensive – not just the memory. The CPU is also 40% more expensive in this case compared to the same CPU built in a standard CMOS process at the same node.

In addition, the embedded flash memory cells are much larger than their external counterparts. Embedded flash cells are 25-40 F^2 (the F^2 number goes up as the node size goes down), compared to ~12 F^2 for external flash. Of course, this is still better than the ~100 F^2 for SRAM, which explains why embedded flash remains the preferred choice for a wide range of devices. But the upper limit for most embedded NVM devices is still around 2Mbytes (16Mbits), and this limit is moving upwards extremely slowly. SoCs with larger embedded memory size exist, but these tend to be more expensive devices targeting very high-end applications.

SoC/MCU vendors have a limited availability of process options if they want to use embedded flash. Pure logic processes offer more options than the processes including flash. An embedded flash process is a compromise between what is needed for memory and what is needed for logic. This makes it a more complex process compared to a standard logic process or a standalone memory process.

The most advanced logic processes do not offer embedded flash. Flash is only offered in an older/larger technology node. A device with embedded flash that would otherwise benefit from a more advanced technology node will end up being built in a less cost efficient node. And to add insult to injury: In the most advanced nodes which do offer embedded flash, it is really only the logic circuitry that is using the smallest transistors. The flash memory has to be built using larger transistors. As a result, valuable high-density die area is used for low-density memory.

In addition, the coarser process nodes that support embedded flash are not as power efficient as the latest process geometries, making it difficult to build SoCs that require high level of compute power while operating from batteries.

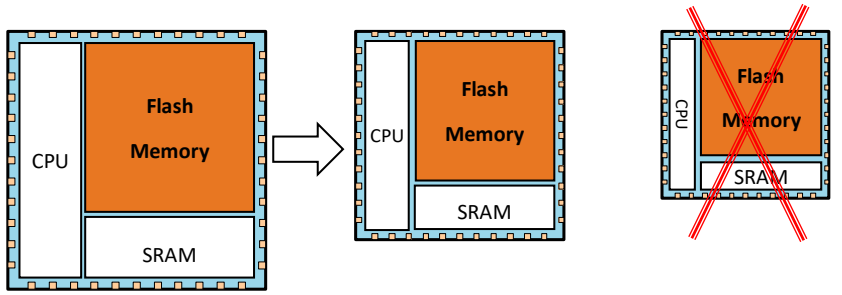


figure 5 - An embedded device may be shrunk, but only so far...

There is yet another cost adder when using embedded code storage: There is no memory size flexibility for the device. This same limitation applies to both embedded NVM and embedded SRAM. The amount of memory available is what you have and what you have to pay for regardless of how much memory your design is actually going to use. Most MCU customers do not like to pay for much more memory than their design needs, therefore, MCU vendors typically have to offer a family of devices with different memory densities.

2.3 Adding AI - The Challenge that None of the Above Solutions Can Solve

With all of the different memory options presented so far, it would make sense that at least one of them would be a good choice for any system. However, there are still some systems where none of these memory configurations provide the code storage solution that the system needs.

The problem occurs with certain devices that require intelligent local data processing capability, like many of the systems that now start adding Artificial Intelligence (AI) to their toolbox. This is a class of devices that require a relatively large program memory while at the same time are running high-performance time-critical applications.

Embedded flash only addresses low- to mid-range products, while suffering from limited scalability, limited memory, restricted performance, relatively high power consumption and high cost. External DRAM addresses only the highest end products. Existing solutions are power-hungry, performance-limiting and expensive.

What these systems really need is something that has the fast wakeup time and low standby current consumption of a large embedded flash, but without the exorbitant cost that a multi-megabyte embedded flash would demand today.

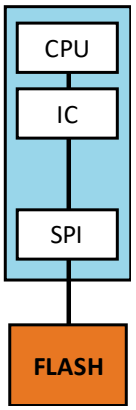
2.4 Execute in Place (XiP)

To solve this problem, execute in place is once again becoming a viable option. Where the old XiP solution was based on a parallel interface with a lot of pins directly addressing the memory, the modern solution saves a lot of pins by using serial flash devices that communicate via a Serial Peripheral Interface (SPI) bus.

XiP memory can either be used as program extension for SoCs with embedded Flash, or it may be used as the main program storage for SoCs built on a standard CMOS process.

In both cases, the CPU is built with an instruction cache (IC).[8] Refer to figure 6. Every time the CPU fetches an instruction, it will first check in the cache memory to see if that address location has already been loaded into the device. If it has, the CPU will load the instruction from cache, and if not, it will start a read operation from the flash to fill the missing cache line. As most software uses a lot of loops, the chance of finding the next instruction in the cache is actually quite high, and it is easy to achieve cache hit rates in the range of 95%-99%.[9]

With hit rates that high, it is easy to assume that the few percent misses are not important to the overall performance of the system, and that almost any type of flash memory could be used as the external flash memory. However, nothing is further from the truth. The difference in access times between the on-chip cache and the external memory can be so high that the system can easily spend more than half the time waiting for missing instructions to be loaded from the cache.



The biggest advantage of a XiP system is that vendors of small-to-mid sized MCUs/SoCs are no longer limited to processes that offer embedded NVM.

figure 5 - An embedded device may be shrunk, but only so far...

A cache controller (with its dedicated cache memory) does not have to be a large block to be useful. Typical cache memory sizes for these types of MCUs may be in the range of 8 to 32 Kbytes, which is tiny compared to the SRAM memories found on many boot+SRAM devices.

The advantages of an execute in place (XiP) system are many. The biggest advantage is that the vendors of small- to mid-sized MCUs/SoCs are no longer limited to processes that offer embedded NVM. They can choose whichever process is the best available for the device they want to make. So instead of being limited to 40nm or above, they can now go down to 28nm, 20nm or lower. And even if the design stays at a higher node, the number of mask layers goes down significantly, which reduces both mask costs and cost per wafer.

This reduces the cost of the on-chip circuitry considerably, which means that overall chip cost will go down, or much more advanced MCUs can be built for the same cost. And dynamic power consumption will also go down with finer geometries. We also see that CPU solutions previously built as application processors, with external DRAM and boot memory used for code storage, are now built as XiP systems at a significantly lower cost.

Another very important cost advantage for XiP systems is that one chip can cover a very wide range of applications. There is no longer any need to have multiple versions of the same device to cover multiple code sizes. This further reduces the mask cost. Since the vendor only needs one mask set, they can now afford the higher up-front cost of a mask set in a more advanced node, getting all the performance improvements and cost reductions that can give them.

Even though an XiP system uses a separate die for flash, it has many of the same features as an embedded flash device. The standby or sleep power can get down to the same sub-µA range as embedded flash devices, as there is no longer any need to keep DRAM or SRAM code storage alive.[10] At the same time, the wakeup times will be in the range of µs to a few ms, just like systems using embedded flash. (The actual wakeup of the MCU and memory will be in the µs range. Longer wakeup times will result from startup times of oscillators and the time the MCU spends reloading parameters for internal registers.) There is no need to reload large blocks of code before execution starts; execution starts as the first cache line is loading.[7]

This does not mean that embedded flash/EEPROM will disappear completely. For smaller systems, the advantages of embedded NVM outweigh the disadvantages. It is the mid-sized and large embedded flash systems where XiP becomes the preferred memory configuration. For systems larger than 4Mbytes (32 Mbits), XiP is a clear winner.

These are the systems that are too large to use embedded memory, but too small to need the memory sizes offered by DRAM. XiP is also likely to come out as a lower cost option for smaller systems currently built with 1 or 2 Mbytes (8 to 16 Mbits) of embedded flash.[11]

If it is done right, an XiP system is the equivalent of a device built using embedded flash, except that it will be faster, offer much larger memories, consume less power and cost less.

2.4.1 Limitations of Standard Flash for XiP Solutions

Now, getting an XiP solution implemented correctly is not that easy if you try to use standard flash for this purpose. Standard flash devices leave a lot to be desired for an XiP application. As their main purpose over the last decades has been to serve as boot memory for shadow flash systems, these memories have been optimized to serve that application very well, and that leaves a lot to be desired for their use as XiP memory.

Specifically, as XiP memory, standard serial flash delivers much lower performance than one would want to see, and the power consumption is high in both operational and sleep modes. In addition, since boot memory is read only once, power consumption has never been much of a concern for that application, but since XiP devices are read continuously, power consumption during read is critical.

An equally huge problem is the lack of performance seen when using standard SPI flash in an XiP system. Standard flash can only run at 104 or 133 MHz in SDR mode. (SDR is Single Data Rate mode, sometimes refered to as SIngle Transfer Rate or STR mode.) Even when running in a quad mode, the peak throughput is limited to 52 or 66 Mbytes/s. There are SPI devices that offer DDR operation (Double Data Rate)[12], but due to the delay of the data coming out of the memory chip compared to the clock signal from the MCU, these are usually limited to maximum clock speeds of 70-80 MHz or less. So, the data throughput from DDR standard flash is rarely much more than what can be achieved in SDR mode.

And even if this peak throughput could be sufficient for many MCUs with good cache controllers and high cache hit rates, the time it takes to get the first bytes from the flash after a cache miss is the real performance killer. Flash access time for both embedded and standalone flash is around 100ns. For embedded flash, there is a direct connection between the flash and the CPU (or instruction cache). For external SPI flash, there is additional time used for clocking commands and addresses in and data out.[13]

SPI flash vendors have added more communication modes such as various dual and quad interface options, however most primarily target faster boot memory performance – and do not address the delays the system experiences every time it has a cache miss and must read one or more cache lines from the flash. When an MCU with 16 byte cache lineshas a cache miss, it will typically read between 3 an4 cache lines each time (3.4 on average). Even the so-called "XiP mode " is an afterthought, as it decreases the total time for reading a cache line by just a few percent at a great cost of increased standby power.[14]

If an XiP system is utilizing over the air updates, standard flash presents another hurdle. It can only do one thing at a time: either read, program or erase. Every time a flash device is busy programming or erasing, the device cannot be read at all. This causes a serious problem if the algorithm being used for OTA is executing out of the same flash.

Even though many flash devices have "Suspend and Resume" operations, these take a lot of time. The programming will progress extremely slowly if every program or erase operation gets suspended every time the MCU needs to read a cache line. The same problem occurs if the system wants to write data to the flash. What an XiP system using standard flash ends up requiring is a different memory to run code from while updating the flash. That may lead to the need to use two flash devices instead of one.

Additionally, if the system needs the performance provided by an octal interface and not just a quad, the number of devices will double again, to four. Alternatively, the required functions for over the air updates may be implemented in ROM code on the host controller, or, if the system has enough on-chip SRAM, the device may be executing code from SRAM while the flash is busy writing.

3. Adesto's EcoXiP™ is Designed for Execute in Place Architecture

EcoXiP is a new line of application-specific non-volatile memory which sets a new standard for high performance, low cost and low power consumption for microcontroller designs using execute in place architecture.

Target systems include IoT edge devices, wearables, connected and wireless embedded systems, home controllers, smart speakers, medical monitors and point-of-sales controllers, just to mention a few.

The memory density of EcoXiP is available as low as 32Mbits (4Mbytes), providing a natural transition from embedded flash MCUs with 1-2 Mbytes of on-chip memory. Larger memory densities (64Mbit, 128Mbit) are also available.

EcoXiP is designed from ground-up with XiP systems in mind, and specifically targets low-power operation. It supports high-speed data speed transfer in both octal and quad modes and offers dual and single data rates for both. Additionally, EcoXiP has functions designed to provide better latency and throughput for cached CPU architectures.

A unique feature offered by EcoXiP is concurrent read and write with no additional delay on the read operations, as described in section 3.5. If a program or erase operation has been started in one part of the memory, the device can still read from another part of the memory without any delay. This significantly simplifies data logging operations and over-the-air updates for devices without on-chip program memory.

3.1 EcoXiP low-power operation

For a flash device designed to be used as boot memory or for data logging, power consumption of the memory device itself is of low importance since it will be inactive more than 99.9% of the time. However, flash read power is of very high importance for XiP devices, as it can be a significant portion of the overall power budget for the system.

The power consumption of read operations consists of two main parts: the amount of power consumed internally in the flash device by the read circuitry, and the amount of power required to transfer the data from the memory device to the host controller. The first of these is the responsibility of the memory chip designers, who in the case of EcoXiP spent a great deal of effort on minimizing the internal power consumption. The second part is equally influenced by the system designer.

The power consumed by data transfer is proportional to CV^2f, where C is the capacitance seen by the output driver, V is the voltage swing of the signals, and f is the average switching frequency of the signals. Lowering the switching frequency f will reduce the peak power, but assuming that the same amount of data has to be read, the total energy required will remain about the same -- the reads are just spread out in time. [15] The voltage and capacitance are the parameters where the choices of the system designer may make a larger difference.

3.1.1 Low voltage operation

The voltage is the obvious first choice to minimize, as this parameter is squared. Most XiP systems therefore operate at a 1.8V signal voltage. Even MCUs that operate at 3V usually have a separate 1.8V rail for the XiP memory. 3V operation is typically also limited to a lower maximum frequency than 1.8V operation, as the rise/fall times required for 3V operation at high speed require larger output drivers, which again increase power consumption even more. 1.2V operation is the natural next step to reduce the V^2 contribution further, and this has already been anticipated by the JEDEC xSPI specification.

3.1.2 Configurable strength I/O pins

Reduction of the capacitance will lead to reduced power consumption in two ways: The reduction of the C in the power formula above will directly reduce the power. Additionally, reducing the capacitance seen by the output driver also means that a weaker output driver can be used while still transferring data at high speed. EcoXiP devices have configurable drive strength I/O pins to minimize power consumption. If the system is designed with low capacitance in mind, the drive strength can be reduced to a lower setting. The best way to reduce the capacitance is to place the memory as physically close to the MCU as possible. This has the added benefit that the traces get so short that impedance control and length matching become less important.

3.1.3 Ultra-Deep Power-Down

In addition to these considerations, EcoXiP has features designed to decrease the power consumption even further. Adesto's patented Ultra-Deep Power-Down mode allows the device to reach sleep currents as low as 0.2 µA. This means that the host controller does not have to use an external switch (and waste an I/O pin controlling it) to turn off power to the flash when it is not in use. This leads to significantly longer battery life for battery operated systems that spend most of their time in sleep mode.

If the system wakes up frequently and the 50µs wakeup time from Ultra-Deep Power-Down becomes too long, the Deep Power-Down mode may be used instead. The wakeup time is then much faster (5 µs) while the power consumption is higher (4 µA). Between read operations, the device will be in Standby mode in which the wakeup time is 0, while power consumption is still only 35 µA.[16]

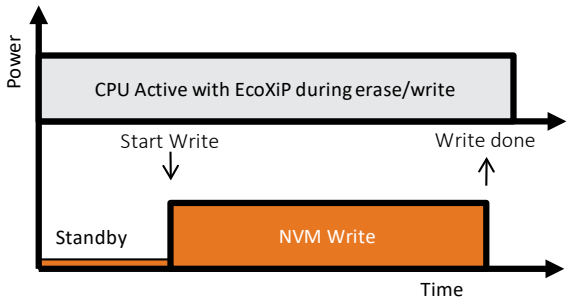


figure 6 - Operation without active interupt

3.1.4 Active Status Interrupt

Adesto's patented Active Status Interrupt means that the MCU does not have to babysit the memory device during a program or erase operation; the MCU may go to sleep to save power while the memory is finishing its write operation. Once the memory has finished the write operation, it can send an interrupt to wake up the MCU again, and the MCU may write more data.

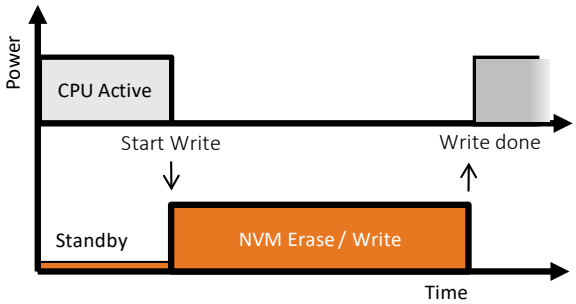


figure 7 - Operation with active interupt.

Alternatively, if there is no need for the MCU to wake up after the memory has finished writing, the host controller may enable Adesto's patented Automatic Power-Down upon completion of a program or erase function. This is a fire-and-forget function: the host will transfer the data to the flash and then go to sleep. Once the EcoXiP memory has finished its write operation, it will turn off power and go to sleep itself without any further interaction with the host controller.

3.2 Prefetching support for cached CPUs

Among the key differentiators of EcoXiP are the additional functions that specifically support cached CPUs. As previously described, CPUs designed for XiP have an instruction cache to avoid having to read every instruction from external memory. Even with a high cache hit rate, the delays experienced when reading from external flash are significant, so the behavior of the flash makes a huge difference in CPU performance. Most flash devices do nothing special to support the CPU during these critical reads, but EcoXiP was designed for this purpose.

The key function here is the improved Burst Read with Wrap function. This ensures that the next cache line is available when it is needed. The *traditional* Burst Read with Wrap implementation is shown in figure 8. It allows the CPU to request the critical word first, read to the end of the cache line, and then wrap around to the beginning to read the rest of the cache line so the cache controller can mark this cache line as residing in the cache memory.[17]

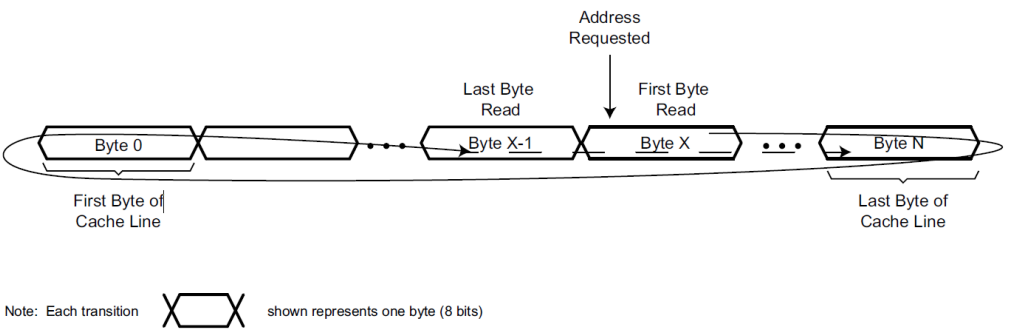


figure 8 - Operation with active interupt.

CPUs with instruction cache (IC) benefit from this feature, as it guarantees an efficient way of reading a whole cache line in one burst regardless of which byte in the cache line the read starts from. The CPU will receive the missing instructions immediately. This will improve the performance of the CPU compared to starting to read the bytes from the beginning of the line. However, the disadvantage of this function is that it only works for reading a single cache line. The system will have to issue another read operation for the next cache line.

Adesto's improved Burst Read with Wrap implementation supports Wrap-and-Continue as shown in figure 9. This function takes advantage of the fact that in most cases, the next cache line requested is the one following the one that was just read.[18]

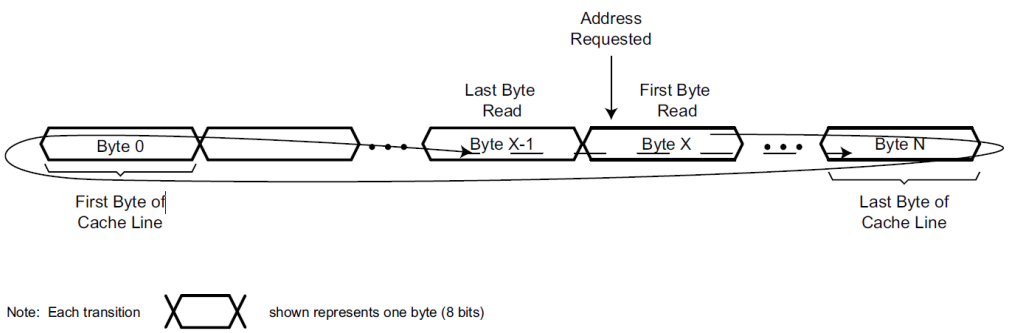


figure 9 - Traditional Burst Read with Wrap

Once the first cache line has been read, the EcoXiP memory is ready to immediately provide the next cache line. This next line will always be read from the beginning of the cache line. The CPU will then decide if the next cache line it requires is the one following the one that was just read, in which case, it can just continue clocking out the following cache line. If it is jumping to another address, it will end the transaction by de-asserting the Chip Select (CS) signal and start a new read operation.

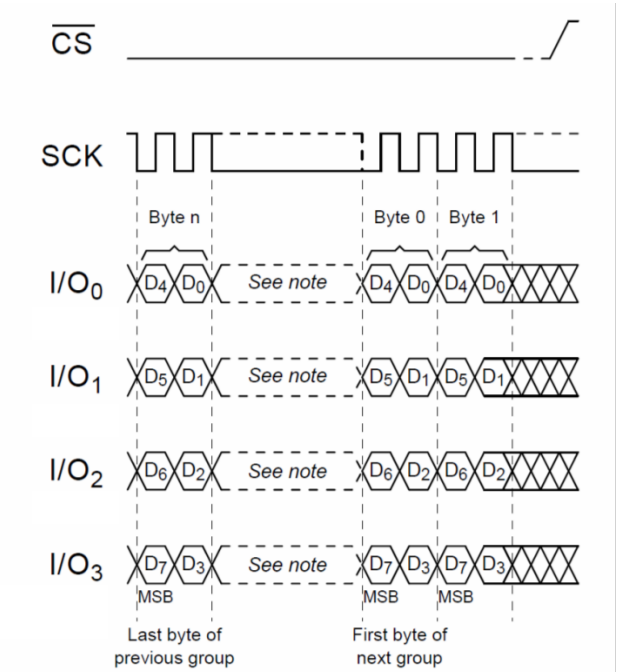


figure 10 - Timing diagem for two consecutive accesses (Quad SDR mode shown in tis case)

Note: If the CPU needs time to decide if the next cache line requires a jump or not, it can just stop the clock signal to the memory for as long as needed, as shown in figure 10. The memory is ready with new data without delay.

This function provides a significant reduction in access times for consecutive cache lines, as shown in figure 11. The figure shows that even though a standard octal DDR device (upper line) is capable of transferring commands, addresses (dark blue) and data (orange) at very high clock rates, there is still a significant delay caused by the flash access time.

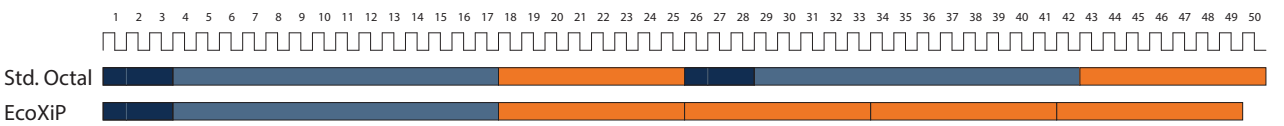


figure 11 - Fetch time for the next consecutive 16-byte cache line (133MHz bus with 100ns access time)

This means that dummy cycles (shown in grey) are required between the last byte of address and the first byte of data out. Both standard octal devices and EcoXiP need dummy cycles for the first cache line read. But EcoXiP with Wrap-and-Continue does not need additional dummy cycles for the 2nd, 3rd or nth cache line, and can output four cache lines in the time it takes other octal DDR devices to output just two.

On a typical 266MHz CPU with a 4% miss-rate, the CPU performance with Wrap-and-Continue is 1.4x higher than the same system without using this function. More details about this example can be provided by Adesto upon request. Compared to a traditional quad device, the performance is 2.5x higher.

Different CPU architectures use different cache line sizes. Adesto's EcoXiP devices support Wrap-and-Continue without any additional delays for 8, 16, 32 and 64-byte cache line sizes.

Some traditional quad devices and some octal devices include an "XiP mode" intended to reduce the access time for consecutive cache lines. However, this function only removes the CMD byte of the consecutive cache lines. (Cycle 26 in figure 11.) The address and dummy cycles are still required, which means the amount of time saved by using this mode is insignificant while the increase in standby power is significant.

3.3 Adding support for command fusing at the SPI Host Controller (HC)

Adding support for the Wrap-and-Continue operation requires a small update to an SPI host controller (HC) that does not already support this function. This is already supported by multiple commercial IP vendors, and can also be easily added by chip designers who want to update their existing host controller designs.

The update basically requires the addition of a register, an adder and a comparator, as shown in figure 12. Upon a CPU burst fetch request, the HC compares its LineAddress with an incremented value of the previous Line Address.[19] If the value is equal, the host controller continues the ongoing SPI command. If it is not equal, the HC terminates the ongoing SPI command by de-asserting CS and starts a new command with the address of the new request.

In addition, the HC stores the new Line Address in the register.

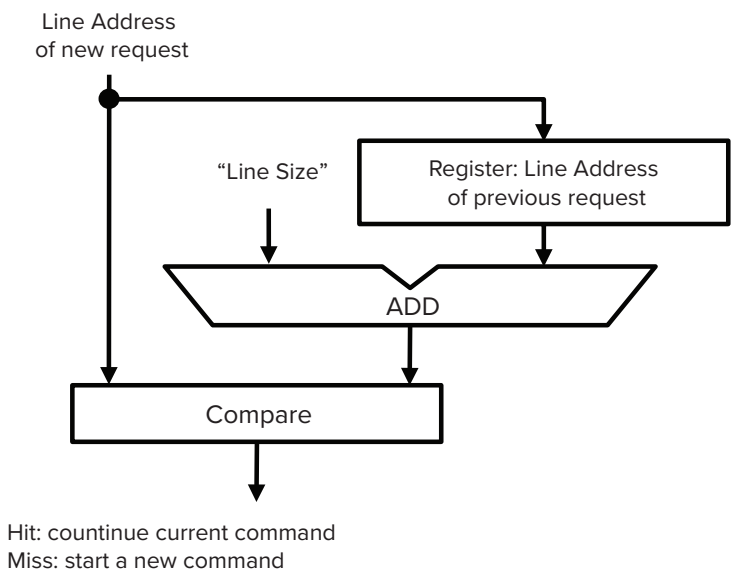


figure 12 - Command fusing at the SPI Host Controller (HC)

Ideally, the CPU should know the line address of a new request before the previous one is finished, but this is of course not always possible. As shown in figure 10, the CPU can then just stop the clock signal to the memory for as long as needed.

3.4 SPI XiP throughput and latency

XiP became possible with Quad SPI (QPI) Double Data Rate (DDR) devices at 80MHz. Quad SPI increases the data I/O lines from 1 to 4, and DDR operation transfers data on both edges of the clock. Contact Adesto for further details on SPI mode descriptions and protocol variations for serial memory devices.

Some suppliers added "XiP-mode" to increase performance, but the effects of this was limited, as shown in figure 13 and figure 14.

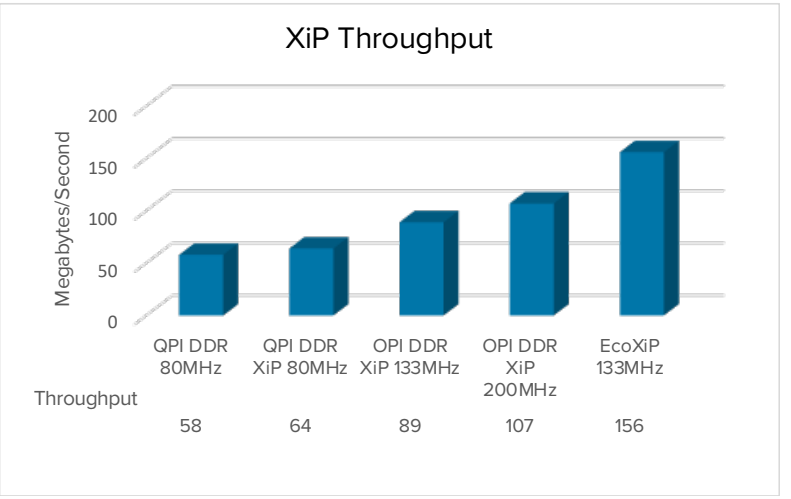


figure 13 - XiP throughput

New Octal (OPI) DDR protocols push the frequency up to 133MHz and some OPI DDR devices can run up to 200MHz. In OPI devices the number of data I/O lines is 8, directly doubling the nominal peak throughput of the interface. This however only increased XiP throughput by about 20%, with a corresponding CPU performance increase of about 12%.[20] This relatively modest performance improvement came at the cost of a much higher increase in power consumption. The XiP latency is at this stage dominated by the flash access time, shown as dummy cycles in figure 11.

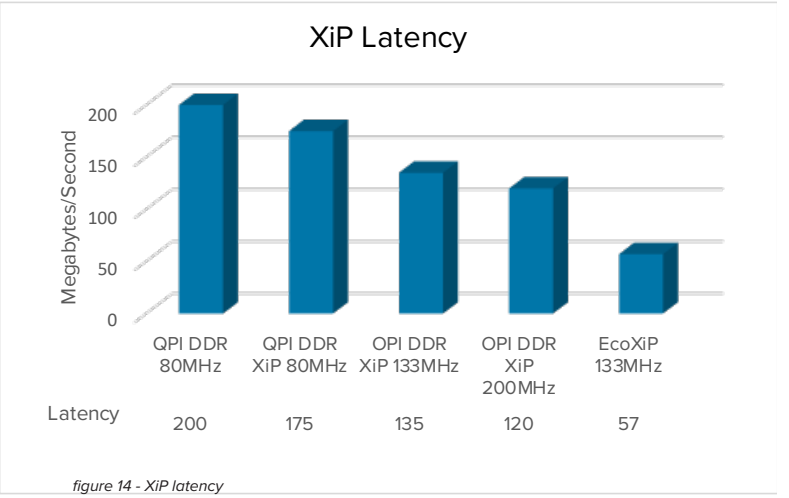


figure 14 - XiP latency

Now, EcoXiP is setting new levels of low latency, high throughput and improved CPU performance, achieved through a high-speed interface and a highly accurate pre-fetching scheme. [21]

3.5 Concurrent Read & Write

XiP operations typically require more or less continuous reads from the memory device. Write operations may not always get the same level of attention. However, write operations are essential even in systems that implement XiP. There are two main cases where writes become important: they are required for firmware updates (OTA and other methods), and also for logging and storing parameters and other data.

XiP systems implement complex schemes to allow occasional data writes. A common solution is to use two NVM devices: one to execute from while the other is being written to. This, however, adds both cost and board space, so system designers are always looking for solutions that allow them to use only one device.

For systems that can execute from on-chip memory (SRAM or flash), the code that needs to run while flash is written may be run from there, though the system functionality will be limited at these times to the code that resides on-chip. Many flash devices do allow program and erase operations to be suspended, so a read may happen in the middle of a write operation. But these operations also add significant delays, so code execution is slowed down.[22]

Adesto's EcoXiP devices support true read-while-write. If a program or erase operation has been started in one part of the memory, the device can still read from another part of the memory without any delay. The timing of read operations is the same regardless of whether there are write operations happening in the background. The program or erase operation is slowed down during read operation to keep the power consumption down. Of course, only one operation can take place on the SPI bus at a time, so there can be no reads just as the host controller is transferring data to the flash. However, this portion of the programming operation occurs very quickly. Transferring 256 bytes of data from the host to the write buffer inside the flash can take less than a microsecond in octal DDR mode.[23] It is the internal programming operation that takes a millisecond or more, and during that time the bus is available for reads.

EcoXiP also supports partial writes to the write buffer. If the host controller has urgent tasks (like fast interrupts) that cannot wait for a full 256-byte buffer to be transferred to the flash, it can use the write-to-buffer operation to transfer smaller blocks at the time and then use the buffer write command to transfer data from the buffer to the flash.[24]

The flash memory in Adesto's EcoXiP devices is divided into two or more memory banks. Adesto's scheme allows read operations from one bank while the device is busy programming or erasing another bank. For ATXP032, the device is configured into two banks: Bank A and Bank B. The border between the banks can be set with a granularity of 1/8th of the array size (4 Mbits).[25] Read commands to one bank can be done while a write is in progress in the other bank. For ATXP128, the memory is split into eight banks of 16 Mbits each. While a write operation is taking place in one bank, read operations may take place in any of the other seven. ATXP064 is similarly split into four banks of 16Mbits each.

3.5 EcoXiP Security Features

EcoXiP includes two 128-byte, One-Time Programmable (OTP) security registers that can be used for purposes such as unique device serialization, system-level Electronic Serial Number (ESN) storage, locked key storage, etc. The OTP Security Register is independent of the main Flash memory array and is comprised of a total of 256 bytes of memory divided into two portions. The first 128 bytes are factory programmed with a unique identifier. This unique identifier is fixed and cannot be altered at any time. The other 128 bytes are user programmable. However, this register is one-time programmable so each bit in the user-programmable register can be programmed once but cannot be erased. Software may lock the user-programmable register to preclude any further programming operations. The user-programmable register is locked when the last byte in the register is programmed.

These registers may be used to prevent third parties from copying the design by just buying the MCU and memory off-the-shelf. At production time, the system producer will read out the unique identifier of the flash device, read out the unique identifier of the host controller, calculate a hash value based on these two numbers (and any additional number, like a checksum of a key portion of the memory), write this hash value back to the user-programmable register and lock the register. At run time, the software can then confirm that the MCU currently running, and the memory it is running from, were programmed together at the original vendor's site. If someone tries to copy the system, the values will not match.

4. Example AI Application: Battery Powered Smart Speaker with Voice Control

Users want long battery life so the system must have ultra-low energy consumption while in an idle state. Users also expect immediate response from the device once they speak their command word.

As an example of an Artificial Intelligence (AI) application, let us look at a battery powered smart speaker with voice control. The device is voice activated, responding to a wakeup word like 'Allegra', 'Odessa' or 'Adesto'.[26] Once awake, a virtual assistant can handle further voice commands.

AI relies on two distinct modes of operation: The learning phase, which is where the system learns the characteristics of the types of data it is operating on, and the inference phase, where the system is using what it has already learned to interpret new data.

The learning phase consists of feeding the system large amounts of known data and building up a set of inference tables as the output. This requires a lot of computing horsepower and/or time. For an edge device, described here, the learning phase is not done on the device itself. It is done "once and for all" on a large computer system, and the resulting inference tables are stored as fixed data tables in the edge device.

The inference phase is the word recognition phase. This is repeated every time the device is receiving new data. For the wakeup phase, for instance, it is listening to a spoken word and using the inference tables stored on the device to determine if the word it just heard was its magic wakeup word.

Unlike many similar devices currently on the market, the goal for the smart speaker described here is to keep average power consumption low enough to allow the device to run from batteries. It should not be required to be connected to external power all the time. This therefore needs to be a power-efficient AI system.

Real-time AI algorithms require a great deal of CPU horsepower, so our system needs a large, powerful CPU with a lot of memory. At the same time, users want long battery life, so the system must have ultra-low energy consumption while the AI system is trying to detect the wakeup word. This means that the DRAM and SRAM memories must be powered off while the device is in sleep mode. And users also expect immediate response from the device once they speak their command word. The part of the system that recognizes the wakeup word therefore needs a very fast response time.

All of these requirements are difficult to meet by just one CPU core and one memory system. Two or more cores, with different memory systems, is likely a more cost-efficient solution. These may be two separate MCUs or a single-chip SoC with a multi-core CPU.

The main CPU, as shown in figure 15, will handle the bulk of the local AI processing. It may also call up to the mothership in the cloud as needed. In that case, a lot of the heavy processing will not take place inside the smart speaker itself, but in a remote system server in a data center.

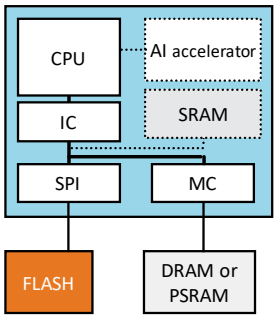


figure 15 - Main CPU

The main CPU handles the bulk of the local AI processing.

This main CPU may be a single MCU, a multi-core CPU, an AI accelerator, a GPU or any combination of these. The amount of memory required may be from a few hundred megabytes to several gigabytes. Systems limited to handle only a few spoken commands can operate on much less memory.

This means the CPU will boot from external flash (large NOR flash or NAND flash) and execute from external DRAM or PSRAM. To save power, the main CPU will be turned off when not in use. It will only be woken up after the wake-up word has been received and recognized. This necessarily means that there will be a certain delay each time it wakes up to reload the memories.

While the main CPU is sleeping, and during the time it takes the main CPU to reload its memories, the always-on CPU, shown in figure 16, will be in charge. Its main task is to listen to incoming sounds, detect and recognize the wakeup word and then alert the main CPU as required. At a minimum, it will require enough AI capability to recognize a single spoken word. Once it detects the right word, it will wake up the main CPU and let the big guy take it from there.[27]

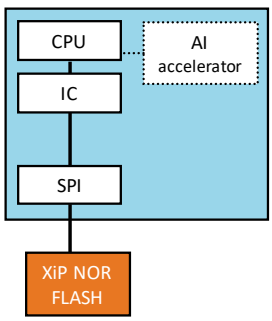


figure 16 - Always on CPU

The chief task of the always-on CPU is to listen to incoming sounds, detect and recognize the wake-up word and then alert the main CPU.

The always-on CPU may also handle other tasks that do not require advanced AI, for instance housekeeping and audio playback. This means that the main CPU can be put back to sleep relatively fast, while the always-on CPU holds the fort. The amount of memory the always-on CPU requires to handle its tasks is much less than what is required for the main CPU.

The always-on CPU may also be turned off when not in use, provided it can wake up quickly enough. For instance, it may sleep until the microphone detects sound, but must be able to start executing fast enough to catch the beginning of a spoken word.

This makes it difficult to use boot+SRAM as the memory configuration for the always-on CPU. The amount of memory required for AI applications means on-chip SRAM would be expensive, and the response time requirement means that there is no time to reload the memory. The SRAM would have to be kept on all the time, causing high power consumption. To reduce SRAM size, it would be possible to keep only the code in SRAM, and keep the AI inference tables in flash. But this option would still consume power.

Embedded flash could handle the wakeup time requirement, but the large memory requirement is a problem here, too. Additionally, using embedded NVM means that the always-on CPU cannot be put on the same die as the main CPU. Or, if we really wanted to keep both CPUs on the same die, the main CPU would be much more expensive than necessary if an embedded NVM process was used for the main MCU.

XiP memory as code storage is clearly the best option for the always-on CPU. With no embedded NVMs on either of the CPUs, it is then easy to add both to the same die and create an SoC.

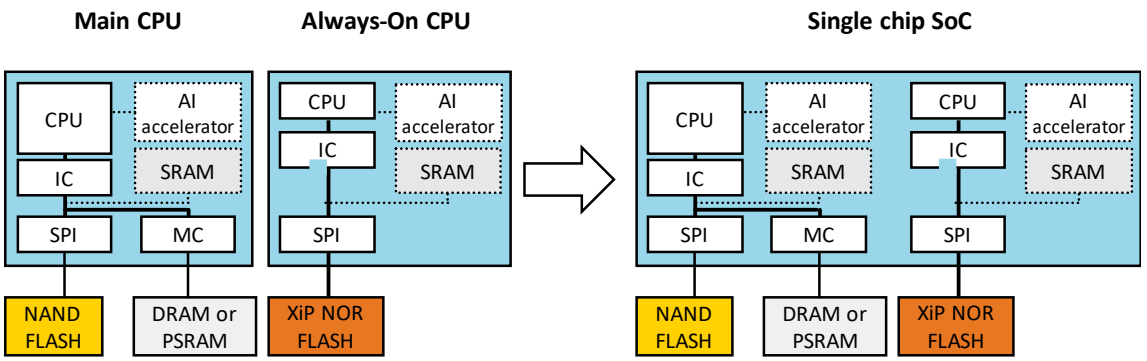


figure 17 - The most cost-efficient system is based on a single chip SoC without any internal NVM

AI applications need to go through a large amount of data in a very short amount of time. The inference engine needs to go through a large number of NVM-stored coefficients in a very short time and with minimal power consumption. The data sets are too large to fit in the cache, so the raw bandwidth of the interface between the flash and the host controller becomes the critical factor for maximum performance. This is where the octal DDR interface of the EcoXiP really shows a difference compared to a quad SPI device.

5. Summary

For AI applications on the edge, the performance and cost of the system will be directly dependent on the performance and cost of the memory. External DRAM is too power hungry, external quad SPI flash is too slow, internal embedded flash is too small or too expensive.

Adesto's EcoXIP is best suited to handle all the requirements of an always-on AI application. It is offering an external memory solution with all the benefits of embedded memory, at much higher memory densities than embedded memory can offer, but at a much lower cost.

Footnotes

[1] Volatile memory is memory that will lose its contents when power is turned off. When the device is powered on again, the device may be cleared, or it may contain random data.
Non-Volatile memory is memory that will keep its contents when power is turned off, code and data is available when the device is powered up again. Older devices used various forms of Read Only Memory (ROM). The first ROM devices required a change to the top metal layer of the device (mask ROM). Later came one-time programmable (OTP) ROM, (UV) erasable EPROM, electrically erasable EEPROM and Flash memories; first as stand-alone memory devices and later available as on-chip NV-memories for MCUs.
RAM = Random Access Memory. This is memory the CPU can read from and write to, at any location. It is typically a volatile memory type (but there are exceptions).

[2] Flash memory comes in two flavors: NOR flash allows random access reads of every byte and can be used as code storage accessed directly by the CPU, also known as eXecute-in-Place (XiP). NAND flash costs less per MB, but can only be accessed in blocks or pages, similar to a hard disk drive.

[3] SRAM = Static Random Access Memory. This RAM type will keep its memory content as long as it is powered on. SRAM is typically built using the same transistors that are used for the CPU, so adding SRAM to a CPU does not change the manufacturing process.
DRAM = Dynamic Random Access Memory. This RAM type must be refreshed at regular intervals to avoid losing data. It is more difficult to use than SRAM, but it is a lot cheaper. It requires a very different manufacturing process than a CPU, so DRAM is always a separate die. Embedded DRAM was attempted at some time in the past, but this is no longer considered a cost effective option.
PSRAM = Pseudo-Static RAM. Wolf in sheep's clothing. This is a DRAM device with additional interface circuitry that makes it look and behave (almost) like an SRAM, but at a much lower cost than a true SRAM.

[4] The relative sizes of memory cells for a given process node is usually provided in "F-squared" (F^2) where "F" is the minimum feature width in nanometers of the process node (F = 40nm 28nm 20nm…)

[5] Typical refresh intervals are 64ms for each page. For large devices with many pages this means that the interval from one page refresh to the next will get very short, as all the pages have to be refreshed within those 64ms. The device will appear to be busy refreshing a page somewhere in memory almost all the time.

[6] The one exception here are octal SPI PSRAM devices. These take advantage of the much lower pin count of an octal SPI device compared to a parallel PSRAM or DRAM device to build devices that are much smaller than other PSRAM devices.

[7] The left part of Figure 3 shows what was a common configuration for small MCUs in the days before embedded NVM: To be able to use a smaller package and save cost, he MCU could have some address bits multiplexed on the same pins used by the data. An external latch (not shown) would be used to hold the address value of those pins while the MCU used the pins to read or write data.

[8] The performance difference for XiP systems with and without an instruction cache is so huge that building an XiP system without a cache is in most cases done by mistake, and this is rarely done twice by the same design team. Instruction caches are useful also for the other memory architectures described in this document, but do not provide the same huge difference in performance for a device running from on-chip memory as for a device running from external memory.

[9] A cache line is the smallest memory block the cache controller is keeping track of. As it has to both keep track of where in the main memory the cache line is coming from, and if the data in the cache line is still valid, it does not make sense to store all that information (called a tag) for every single instruction in the cache memory. Instead, the cache operates on cache lines (or blocks) of 8 to 64 bytes, where 16 and 32 bytes cache lines are the most common. Every time the CPU has a cache miss, it has to read the entire cache line from flash. Using smaller cache lines mean fewer extra instructions have to be read each time, but more tags have to stored. Larger cache lines mean less memory used for storing tags, but more time spent reading additional instructions that may not be needed.

[10] Depending on the application, there may be a need to keep some SRAM/DRAM alive for data storage. Alternatively, some data may be written to flash, which will also consume power. However, for most applications optimized for low power consumption, the amount of data required to be kept will be much smaller than the amount of code storage used in a boot+RAM application.

[11] The exact point where an XiP system becomes lower cost than the same system using embedded flash will vary based which other digital and analog blocks the systems contains. A device where the rest of the device is large compared to the memory section will save more by going to a standard logic process compared to a device where the flash memory is a dominant part of the device.

[12] Dual Data Rate mode. Sometimes also called Dual Transfer Rate (DTR) mode. People sometimes confuse this with DDR DRAM, which is DRAM running in DDR mode.

[13] Parallel flash the way it was used in old systems can still be used to reduce the time it takes to transfer address and data. However, the high number of I/O pins required means that this solution becomes too expensive for most applications today.

[14] The "XiP mode" uses a mode byte inserted as one of the dummy bytes to tell the memory device that the next command will be the same type of read command, but without the command actually being transmitted. This changes the read mode from a 1-4-4 mode or 4-4-4 mode to a 0-4-4 mode, saving 8 or 2 clock cycles for each read transaction.

[15] The main way to reduce the amount of data read for an XiP system is to add a better cache controller and a larger cache memory on the host controller, so the cache hit rate goes up. This will increase performance and cost of the host controller while reducing power consumption of the flash memory. As always, the system designer will have to trade performance, cost and energy consumption against each other, improvement in one will deteriorate (at least) one of the others.
Operating the device in DDR mode instead of SDR mode leads to a slight reduction in power consumption, as the clock is only toggling half as many times for the same amount of data transferred. Additional savings come from transferring the data in shorter time, which means that the read circuitry of the device will be powered for a shorter amount of time.

[16] For most XiP systems, the standby current consumption is of little consequence, as it is so low compared to the read current. It is only relevant for battery powered systems that for some reason do not utilize the Ultra-Deep Power-Down mode or Ultra-Deep Power-Down modes to save power.

[17] There are XiP implementations that do not use the wrap-and-continue function. These will have to read every cache line from the first byte and then wait up to a full cache line before they get their critical word. This can be a significant drawback particularly for functions requiring fast response time. However, this implementation also allows reading multiple cache lines back to back without additional dummy cycles.
If the CPU does not read the entire line, it cannot mark it off as residing in the cache memory. It then risks having to re-read the same line again if it gets another request for code residing in the same cache line.

[18] The average distance between taken branches for this class of CPU is around 15.5 instructions and an average instruction is 3 bytes long. If the IC (Instruction Cache) cache line size is 16 Bytes, the average number of sequential cache lines fetched is 3.84. The average number of sequential cache lines fetched after the first is then 2.84.

[19] Note: Line address = the address of the first byte in a cache line.
For a cache line size of 16bytes, it will be the address on the AHB bus && 0xFFFFFFF0.
For a cache line size of 32bytes, it will be the address on the AHB bus && 0xFFFFFFE0.

[20] A more in-depth discussion of why doubling the SPI interface speed does not directly lead to doubling of the CPU performance can be provided on request from Adesto.

[21] EcoXiP numbers assume 16-byte Instruction Cache line-size and an average of 3.84 line fetches per IC miss

[22] A single suspend operation may by itself take several tens of microseconds. When multiple suspend operations are required during a write operation, which is likely for code running from external flash, a lot more time is spent waiting for suspend and resume operations than actually reading code. The net result is extremely low execution speed during flash programming.

[23] It can safely be assumed that the small software loop that does the data transfer will have no problems fitting inside the cache at this stage.

[24] In the datasheet, the command is called "Buffer to Main Memory Page Program without Built-In Erase". The opcode is 88h. The "Buffer Write" opcode is 84h.

[25] So, the 32Mbit memory array may be split 4-28, 8-24, 12-20, 16-16, 20-12, 24-8 or 28-4. Of course, the entire 32Mbit array may be treated as a single bank.

[26] See Saturday Night Live or https://www.youtube.com/watch?v=YvT_gqs5ETk for a list of supported wakeup words.

[27] A similar AI application can use face recognition instead of voice recognition for the wakeup function. The always-on CPU uses a low power camera to record low resolution images and use these images to detect the presence of a human face. Once a face is detected, the main system is kicking in to do a more thorough analysis to identify the owner of the face. The memory requirements for the always-on CPU in this application is similar as for the voice recognition.

Additional Information

Adesto is a leading provider of innovative, application-specific semiconductors and embedded systems that comprise the essential building blocks of Internet of Things (IoT) edge devices operating on networks worldwide. Our broad portfolio of semiconductor and embedded technologies are optimized for connected IoT devices and systems used in industrial, consumer, communications and medical applications.

Through expert design, unparalleled systems expertise and proprietary intellectual property (IP), our offerings enable customers to differentiate their IoT systems and product designs, leading to improved efficiency, greater reliability and security, integrated intelligence and ultimately lower cost.

Adesto

3600 Peterson Way, Santa Clara, CA 95054 USA | Phone: +1 (408) 400-0578 | www.adestotech.com | e-mail: info@adestotech.com