Processor Architectures for Data-intensive Applications

When evaluating microcontrollers engineers should look closely at the controller's ability to move large amounts of data without consuming all of the CPU cycles.

By Dany Nativel and Jacko Wilbrink, Atmel®

Broadband access has become commonplace, paving the way for a new content-rich Web experience that now includes both audio and video. The microcontrollers

[1]

that drive these applications must store, process and move massive amounts of data at very high rates between the peripherals, the memory and the processor. For example, digital cameras now have multi-million pixel sensors with huge bandwidth and memory requirements to process and store the vast amount of data. On the other hand, voice and music require less bandwidth. However, streaming content adds real-time constraints to the communications channel.

Before the advent of data-centric applications, the limiting factor in most applications was the ability of the CPU to process small amounts of data quickly. Recent innovations in controller architectures, particularly the addition of DSP extensions to the instruction set and much faster clocks, have overcome the processing challenges. Controllers, such as those based on ARM's 926EJ-S core, can execute a huge processing load. Unfortunately, communications with, on- and off-chip memories have not kept pace.

Conventional 32-bit processors directly manage all communication links and data transfers. They first load data, received by a peripheral, to one of their internal registers and then store it from this previously loaded register to a scratchpad stored in on-chip SRAM or external SDRAM. The CPU must then process the data and copy it back, through an internal register, to another communication peripheral for transmission. This Load Register (LDR) Store Register (STR) scheme requires at least 80 clock cycles for each byte transferred.

An ARM9&#153 processor, running at 200 MHz with an internal bus at 100 MHz, reaches its limit when a peripheral transfers data at about 20 Mb/s &#151 not enough to service an SPI or SSC, much less handle 100 Mb/s Ethernet transfers. If the memory management unit (MMU) and the instruction and data caches are disabled, the ARM9 controller is limited to only 4 Mb/s, not enough to handle even a high-speed UART. The traditional solution to this severe bandwidth limitation has been to increase the processor clock frequency, also increasing both power consumption and heat dissipation. However, even the highest available frequency may not be sufficient to achieve the required bandwidth of today's applications.

Current applications may integrate high-bandwidth peripherals such as 100 Mb/s Ethernet, 12 Mb/s USB, 50 Mb/s SPI, a VGA LCD controller and a 4+ megapixel camera interface. With the advent of these high-speed peripherals, even a 1 GHz processor does not have enough processing power to handle all the data transfers. At 100 Mb/s, the CPU does nothing but move data because there simply isn't any processing power left to do anything else. Thus, although processors can easily achieve the computational throughput to execute an application, they are not capable of moving data fast enough. The challenge is no longer computational; it is bandwidth.

Manufacturers have tried to solve this problem by adding FIFOs to their on-chip peripherals. Unfortunately, FIFOs do not increase bandwidth, they just lower data transfer peaks by spreading the bus load over time. The archaic LDR/STR processor architecture requires the CPU to execute each and every one of those byte transfers, robbing it of cycles needed for processing.

A new approach to processor architecture that includes the use of simple, silicon-efficient DMA (Direct Memory Access) inside the individual peripherals and the addition of dedicated busses between high-throughput elements on the chip provides a lower cost, lower power solution to this problem.

Peripheral DMA

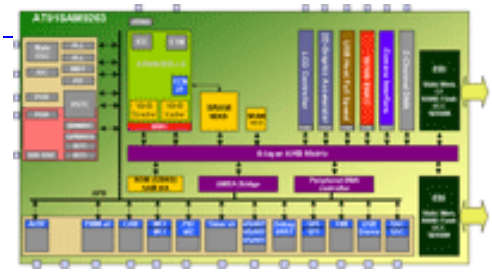
The use of DMA is a natural evolution for embedded architectures that have seen the number of on-chip peripherals and data transfer rates growing exponentially. DMAs solve part of the problem by allowing direct peripheral-to-memory transfers without any CPU intervention, thus saving valuable CPU cycles. DMAs can transfer data using one-tenth as much bus bandwidth as is required by the processor. However, DMA controllers are designed primarily for memory-to-memory transfers. Such DMAs offer advanced transfers modes like scatter-gather and linked lists that are very effective for memory-to-memory transfers but are not useful for peripheral-to-memory data transfers. This adds unnecessary software overhead and complexity to the system design. A better approach is to use an optimized peripheral DMA between the peripherals and the memory. Peripheral DMAs require 90% less silicon than memory-to-memory DMAs, making them cost-effective to implement dedicated DMA channels for each peripheral. Moving the DMA channel configuration and control registers into the peripheral memory space greatly

simplifies the peripheral drivers. The application developer needs only to configure the destination buffer in memory and specify the number of transfers. The software overhead is minimal.

The peripheral DMA frees the host CPU to focus on the computational tasks it was designed for without wasting cycles on data transfers. In fact, a peripheral DMA controller (PDC) can be configured to automatically transfer data between the peripherals and memories without any CPU intervention at all. Additionally, the PDC automatically adapts its addressing scheme according to the size of the data being transferred (byte, half word or word).

Multi-layer Bus

Another problem facing data-intensive applications is on-chip bus bandwidth. When multiple DMA controllers and the processor push massive amounts of data over a single bus, the bus can become



[2]

overloaded and slow down the entire system. A 32-bit bus clocked at 100 MHz has a maximum data rate of 3.2 billion bits per second (Gb/s). Although that sounds like a lot, in data-intensive applications, there may be so much data that the bus itself becomes a bottleneck. Such is the case with internet radio where audio quality is a direct function of the ability to receive and process streaming content in defined timeslots, or GPS navigation involving interactive vector graphics.

This situation can be avoided by providing multiple, parallel on-chip busses and a small amount of on-chip scratchpad memory.

External Bus Interface

When an application shares external memory between the processor and peripherals, the external bus interface limits the bandwidth. The next step to increase bandwidth is to provide two parallel external bus interfaces connected to the internal multi-layer bus: one for system memory and one that supports a high-speed peripheral or co-processor. In embedded applications with man-machine interfaces, the required amount of memory is so huge that it is not cost-effective to put it on the controller. For example, a 24-bit color VGA panel requires a frame buffer of 900 KB. An LCD controller with this much SRAM would be prohibitively expensive so the frame-buffer must be stored in external RAM. The refresh rate is typically 60 frames per second. With a VGA (640 &#215 480 pixels) panel in 24-bit true-color mode, the CPU needs to fetch 7.2 Mb of data 60 times per second, or 432 Mb/s. A conventional 200 MHz ARM9 processor cannot possibly achieve this level of throughput.

Bandwidth can be increased by adding a second EBI and a 2-D (or other) graphics co-processor. The second EBI is connected to a second external memory that is used as an LCD controller frame buffer which is directly connected to the on-chip 2-D graphics co-processor that offloads line draw, block transfer, polygon fill, and clipping functions from the CPU. The performance gain achieved from a second external bus interface is application dependant but can be expected to be in the range of 20 to 40%.

Atmel, for example has developed new 32-bit architectures that exploit these techniques to address the performance, scalability and cost issues discussed above. The company has added peripheral DMA to all its ARM7 and ARM9-based microcontrollers to allow high data rates and maximize CPU throughput. ARM9 microcontrollers with a multi-layer bus architecture provide dedicated busses for the CPU instruction, data cache controllers, as well as all high-bandwidth peripherals. Depending on the number of on-chip peripherals, an Atmel SAM9 microcontroller will have between five and eleven independent 32-bit busses, and a maximum on-chip data rate of between 16 to 41.6 Gb/s. Finally, Atmel has implemented dual external bus interfaces (EBI) for applications with human interface that require intensive graphics processing or large data buffers.

The result of these architectural enhancements is that Atmel’s SAM9 microcontrollers can achieve a data transfer rate of 20 Mb/s and still have 88% of the CPU’s cycles available for processing when executing from shared memory. In contrast, a conventional ARM9 MCU would be stopped in its tracks by a 20 Mb/s data rate. The available MIPS on a SAM9 microcontroller can be increased even further, to 100%, by providing separate memories for the CPU and PDC.

The combination of a nine-layer bus, dual EBIs and peripheral DMA controller allow an ARM9 with LCD controller to execute the above-referenced VGA refresh 60 times a second with 100% CPU cycles still free for other functions. This relatively simple, silicon-efficient addition of DMA, busses and external memory interfaces to the microcontroller architecture turns a processor that effectively has no MIPS for application execution into one that can transfer all the data and still have 200 MIPS remaining for applications execution.

Conclusion

While chip vendors have addressed processing challenges with high throughput CPU cores that have DSP extensions, they have not done enough to accommodate the massive amounts of data that must be transferred between the peripherals, memories, the CPU and any on-chip co-processors. Engineers should look beyond raw MIPS when evaluating microcontrollers. Rather they should verify that the controller’s ability to move massive amounts of data without gobbling up all the CPU cycles. Engineers should look for architectures that off-load data transfers between the peripheral and the memories with a peripheral DMA controller. Dedicated busses that service on- and off-chip memory, the CPU and any high bandwidth peripherals will also help eliminate the possibility of bus bottlenecks. Finally, adding multiple external bus interfaces will allow simultaneous, parallel processing of data from external memories by both the CPU and on-chip co-

processors, thereby realizing the full processing potential of advanced cores such as the ARM926EJ-S&#153.

About the Author
Dany Nativel technical marketing manager, ARM-based Microcontrollers, Atmel Corp.; dany.nativel@rfo.atmel.com; and Jacko Wilbrink is marketing manager for ARM-based Microncontrollers, Atmel Corp.; jacko.wilbrink@rfo.atmel.com.

Links:
[1] http://www.wirelessdesignmag.com/sites/wirelessdesignmag.com/files/legacyimages/0705/feat1_1_lrg.gif
[2] http://www.wirelessdesignmag.com/sites/wirelessdesignmag.com/files/legacyimages/0705/feat1_2_lrg.gif