Direct Memory Access

Direct Memory Access (DMA) is used by Symbian OS to offload the burden of high bandwidth memory to peripheral data transfers and allow the CPU to perform other tasks. DMA can reduce the interrupt load by a factor of 100 for a given peripheral, saving power and increasing the real-time robustness of that interface.

A DMA engine is a bus master peripheral. It can be programmed to move large quantities of data between peripherals and memory without the intervention of the CPU.

Multi-channel DMA engines are capable of handling more than one transfer at one time. SoCs for Symbian phones should have as many channels as peripheral ports that require DMA, and an additional channel for memory-to-memory copies can be useful.

A DMA channel transfer will be initiated by programming the control registers with burst configuration commands, transfer size, the physical addresses of the target RAM and the peripheral FIFO. This is followed by a DMA start command. The transfers of data will be hardware flow controlled by the peripheral interface, since the peripherals will always be slower than the system RAM.

In a memory to peripheral transfer, the DMA engine will wait until the peripheral signals that it is ready for more data. The engine will read a burst of data, typically 8, 16 or 32 bytes, into a DMA internal buffer, and it will then write out the data into the peripheral FIFO. The channel will increment the read address ready for the next burst until the total transfer has completed, when it will raise a completion interrupt.

A DMA engine that raises an interrupt at the end of every transfer is single-buffered. The CPU will have to service an interrupt and re-queue the next DMA transfer before any more data will flow. An audio interface will have a real-time response window determined by its FIFO depth and drain rate. The DMA ISR must complete within this time to avoid data underflow. For example, this time would be about 160 µs for 16-bit stereo audio.

Double-buffered DMA engines allow the framework to queue up the next transfer while the current one is taking place, by having a duplicate set of channel registers that the engine switches between. Doublebuffering increases the real-time response window up to the duration of a whole transfer, for example about 20 ms for a 4 KB audio transfer buffer.

Scatter-gather DMA engines add another layer of sophistication and programmability. A list of DMA commands is assembled in RAM, and then the channel is told to process it by loading the first command into the engine. At the end of each transfer, the DMA engine will load the next command – until it runs out of commands. New commands can be added or updated in the lists while DMA is in progress, so in theory my audio example need never stop.

Scatter-gather engines are good for transferring data into virtual memory systems where the RAM consists of fragmented pages. They are also good for complex peripheral interactions where data reads need to be interspersed with register reads. NAND controllers require 512 bytes of data, followed by 16 bytes of metadata and the reading of the ECC registers for each incoming block.

Power savings come from using DMA, since the CPU can be idle during a transfer, DMA bursts don’t require interrupts or instructions to move a few bytes of data, and the DMA engine can be tuned to match the performance characteristics of the peripheral and memory systems.