Optimizing Embedded Applications using DMA

By Sachin Gupta, Applications Engineer Sr, and Lakshmi Natarajan, Applications Engineer Sr, Cypress Semiconductor Corp.

Just like a human body is composed of many individual systems, an embedded system comprises multiple functions. Even a basic mobile phone in today's world includes calling facilities, messaging facilities, entertainment options like games, music player, radio, camera, Bluetooth connectivity, and so on. Including so many different functions makes these systems quite complex.

The CPU handles the arithmetic and decision-making operations. Most real-time systems might include one or more processors to handle these functions with each processor interacting with the others. On the one hand, we add more CPUs to share the load; on the other, the CPUs may waste time transferring data between them.

An efficient real-time system is one where the CPU is used for the maximum number of tasks to yield better real-time responsiveness and lower power consumption while still being flexible enough to support future enhancements. In order to reduce the CPU time that is being wasted in data transfers, many systems include a peripheral which can do data transfer operations without including the CPU. This peripheral is called the Direct Memory Access (DMA).

The DMA helps to yield better results in terms of CPU usage and hence higher system throughput. Earlier, the concept of DMA was limited to computer applications like PCs, servers, etc. Due to complexity of modern electronics and the need to transfer a lot of data, it has become integral part of embedded applications as well.

In recent days, most high-end processors used for larger real-time systems like automotive, avionics, and medical applications integrate a DMA peripheral. This DMA can be used for the following type of data transfers, with availability varying from controller to controller:

- Memory to memory transfer
- Memory to peripheral transfer
- Peripheral to memory transfer
- Peripheral to peripheral transfer

Data transfers might occupy a large percentage of CPU time in a system. Let's consider a car dashboard system as an example. A dashboard has multiple data to displayed which comes from various sub systems. Figure 1 shows the block diagram of a dashboard system with sub-systems.

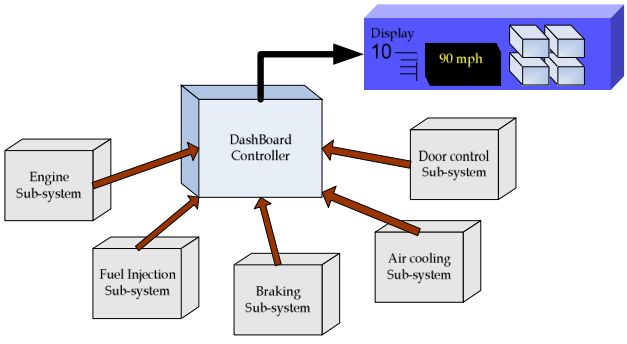

Figure 1: Example of a real-time system and sub-systems

Each subsystem transmits its data to the central processor which controls the display. In this case, the central processor controlling the display has to receive lots of data and then determine which subsystem has transmitted the data and update the display accordingly. If CPU has to handle reception of data, decision-making, and display update, system response time will decrease, resulting in a sluggish system.

Published in EE Times Design (http://www.eetimes.com) November 2010

To improve system performance, developers can use a high-frequency CPU. This, however, comes at a tradeoff in terms of power consumption and system cost. Alternatively, a DMA can free the CPU from these numerous data transfer operations. The DMA handles all the data transfers, leaving the CPU to handle all the other tasks. The above example gives an initial idea of why DMA is required in complex systems.

In a processor, all the peripherals including memory will be connected to the CPU using buses. As shown in Figure 2, these buses are the routes across which data is transferred between the peripherals (including memory) and the CPU. A processor including the DMA will have 2 masters for the bus – CPU and DMA. For instance, a simple C statement like "a = b + c" involves CPU access to data in the memory. The CPU has to access the memory to read variables b and c, calculate the sum, and update the value in the memory location for variable "a". When the CPU wants to access the memory, it submits a request to the bus which then processes the request, retrieves, and then updates the data in the memory location through the bus.

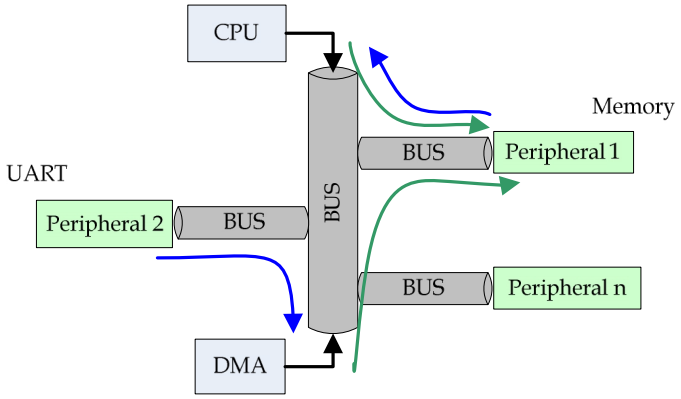

Figure 2: Example of a real-time system and subsystems

When DMA is involved, the DMA can access any peripheral based on the controller features. For instance, consider a system where DMA is used to move received UART data to memory. In this case, the DMA has to access the UART registers and the memory location through the bus. Since the DMA and CPU both can access the bus, there is usually an arbitration mechanism to handle the bus access.

Let's take a simple example of reading 10 bytes of data from an array in Flash to an array in SRAM to illustrate how much time the DMA can save. When the function is done using a CPU alone, the following steps will be executed:

1. Copy the value in the Flash memory location and store the value to a GPR (General Purpose Register e.g.: R0)
2. Copy the GPR value to the Accumulator register
3. Copy the Accumulator value to the SRAM memory location
4. Check if all the bytes have been copied
5. If yes, end
6. If no, Increment the Flash pointer
7. Increment the SRAM pointer
8. Go to Step 1

A simple Flash-to-SRAM copy uses many instruction cycles which involve CPU.
If the same example is handled using DMA, the DMA hardware will handle the process thereby reducing the execution time. Figure 3 shows how the DMA handles the transfer:

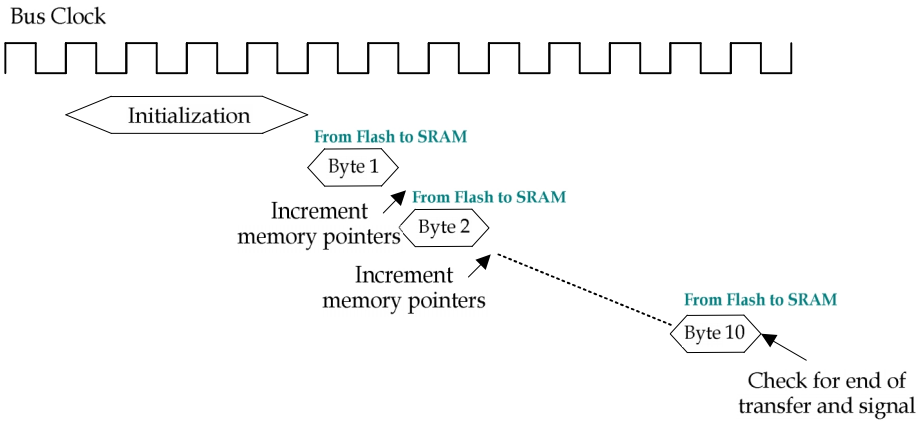

Figure 3: DMA handling of transfer

The DMA takes a few cycles for initialization, then the DMA hardware automatically handles the memory pointer increments. The DMA hardware also checks for the completion of data transfer and signals the completion.

Advantages of using DMA

- The longer the CPU runs, the more power consumed. Using DMA to offload the CPU thereby reduces power consumption.
- The DMA works in parallel with the CPU, thereby simulating a multi-processing environment and effectively increasing the CPU's bandwidth.
- Offloading the CPU results in more idle time for the CPU. This frees up available processing capacity for future product enhancements.

Real-time examples for using DMA

Wave generation using a SoC can be a perfect example where DMA makes a tremendous difference. A wave generator needs to generate waves continuously without interruption. Wave generation can be done by updating the digital-to-analog convertor's (DAC) data register with the values stored in a look-up table. In processors which do not have a DMA, the DAC's data register is updated by the CPU. Using the CPU consumes instruction cycles to copy the value from the look-up table to the DAC.

Instead, DMA can be used to copy the samples from memory to the DAC at the required interval of time. Unlike the CPU, the DMA is not interrupt driven. This means that the latency incurred due to high-frequency interrupts in CPU-based implementations is eliminated. As the frequency and number of samples of the wave increases, the system consumes more CPU bandwidth and hence more power. When a DMA is used, CPU does not need to do anything as long as wave is being generated.

If we look at power consumption, generally the CPU consumes more power compared to a DMA controller. When the CPU is not used, the CPU can be powered down, saving the static and well as dynamic power consumed by it. Thus, the DMA helps in optimizing the power consumption of the system where the major portion of work is to transfer data. If we consider a system where four waves need to be generated as a subset of its functionality, the CPU bandwidth needed to achieve this functionality with and without DMA will look as shown in Figure 4. Power consumption is shown in Figure 5.

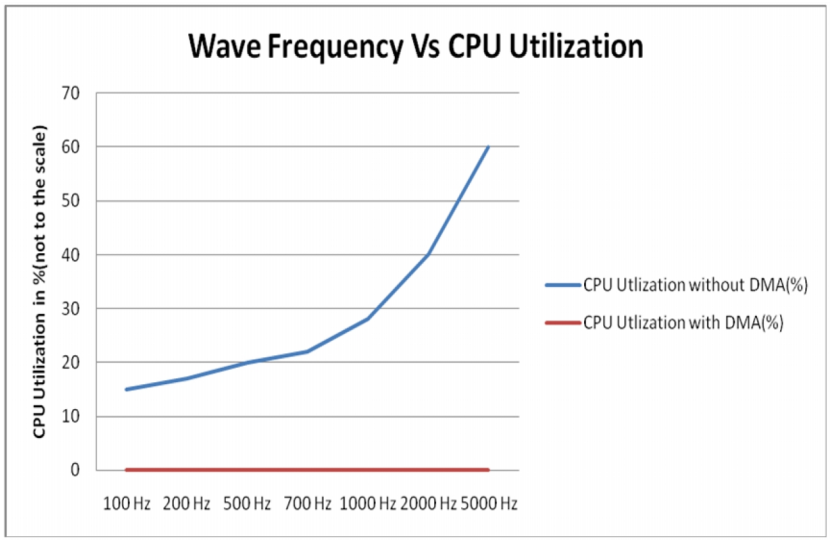

Figure 4: CPU Utilization

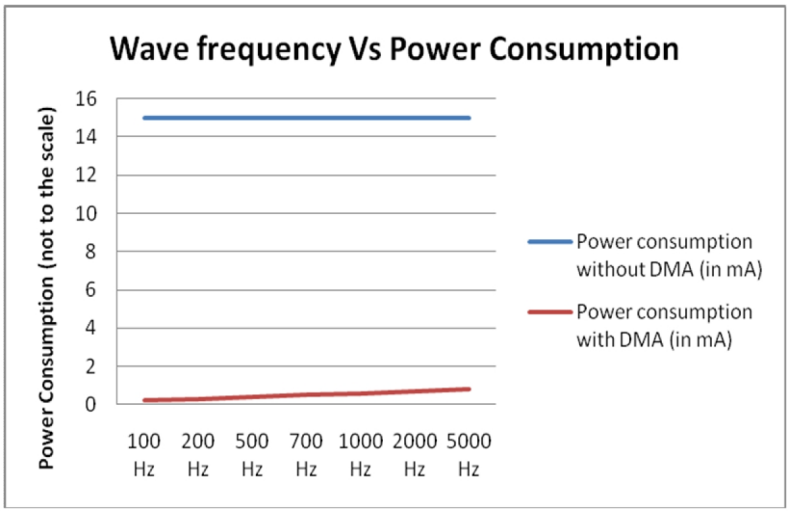

Figure 5: Power consumption

Another example is the buffering of analog-to-digital convertor (ADC) data. Many times, there is need for scanning analog signals with an ADC, storing them to memory and then filtering the ADC data. Using a CPU to read the ADC data and copy it to memory reduces the efficiency of the system. In this case, the CPU is required not only to handle the data transfer but also to filter the data. If sample rate is of the order of 100's of ksps or in Msps, it may completely consume the CPU's bandwidth. To reduce the CPU load, the DMA can be used to store the ADC data to memory leaving the CPU to do the filtering. This leads to the concept of double buffering, where the CPU can process the data in one buffer while the DMA fills another buffer.

Most systems use communication protocols. Transferring data from or to a communication block using DMA is a common usage. In a diagnostic system, there will be a large amount of data transmitted and received through the communication protocol. A DMA handling these data transfers allows the CPU to be free to do other operations.

With systems becoming more and more complex, the need for DMA is continuously increasing. A processor with DMA can provide multiple channels for the DMA, each configured for a selected peripheral. The general features of a DMA peripheral include:

- Access to multiple peripherals in the processor
- Multiple DMA channels in the processor
- Different priorities for each channel
- Handling the source and destination address
- Handling the increment/ decrement of the source and destination address based on the configuration
- Checking for the completion of the transfer of configured number of bytes for a DMA channel
- Burst transfers which split the whole data transfer into multiple blocks
- Generate an interrupt when required

In System of Chip (SoC) based implementation where almost all peripherals are integrated onto a single chip, a DMA proves to be helpful resource. From a system perspective, source and destination and the way data has to be transferred between them vary from application to application. This means that a fixed configuration or fixed mapping of a DMA channel might leave the DMA unusable for the application. When looking at the product development life cycle of modern systems, requirements are not always clear during initial development phases. Developers need a flexible DMA controller in order to use it for all the peripherals and memory with programmable configuration. The PSoC3 and PSoC5 from Cypress Semiconductors have a highly configurable DMA peripheral meeting the needs of the today's systems. The DMA peripheral in these SoCs includes multiple DMA channels (maximum 24) and transaction descriptors (maximum 255). The transaction descriptors contain the description of the data transfer to be completed. It includes source address, destination address, and number of bytes to transfer, enabling an interrupt after transfer and handling byte swapping as well as incrementing the source and destination address. The transaction descriptors can also be chained to perform multiple operations. Each DMA channel can use any of the transaction descriptors. Once a DMA channel is triggered, the DMA channel processes the linked transaction descriptor.

- DMA channels can be configured for any peripheral. No fixed routing between channel and peripheral
- Availability of multiple transaction descriptor
- Transaction descriptor can be chained
- DMA channels can be chained
- Each DMA channel can do maximum of 64 K bytes transfer
- The DMA channel can handle endian swapping
- The DMA channel can generate interrupt after completion of every transaction descriptor

Consider multiplexing 4 ADC channels. Each channel data needs to be written to separate buffers. In this case, one DMA Channel and four transaction descriptors are required. Each transaction descriptor can be used to transfer ADC data to each consecutive buffer. The transaction descriptors are chained so that after completion of a channel, the next channel is automatically handled. This operation would have involved the CPU for moving data to different buffers in case of traditional processors with fixed routing DMA channel. With a flexible DMA, CPU involvement is completely removed.

A DMA is one of the integral assets in today's embedded systems. The use of DMA can enhance the utilization of CPU significantly, thereby helping developers to design optimized, high-performance systems. Based upon a system's complexity, variation in peripheral integration from application to application, and changing requirements and specifications, developers need a flexible DMA controller which empowers designer to make the best of the DMA for the task at hand.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
Phone: 408-943-2600
Fax: 408-943-4730
http://www.cypress.com