Lecture 13 objectives are to:

- Define the terms bandwidth, latency, and priority;
- Introduce the concept of DMA synchronization.

1. Introduction

As the performance requirements of our embedded system grow, there comes a point when busy-wait and interrupt synchronization methods of I/O interfacing are not fast enough. **Direct memory access** (DMA) is an interfacing technique that allows data to transfer directly from I/O device to memory, or from memory to the I/O device without going through the processor. Embedded system designers will need to use DMA when interfacing high speed devices.

Bandwidth, latency, and priority are quantitative parameters we use to evaluate the performance of an I/O interface. The basic function of an input interface is to transfer information about the external environment into the computer. In a similar way, the basic function of an output interface is to transfer information from the computer to the external environment. The **bandwidth** is the number of information bytes transferred per second. The bandwidth can be expressed as a maximum or peak that involves short bursts of I/O communication. On the other hand, the overall performance can be represented as the average bandwidth. The latency of the hardware/software is the response time of the interface. It is measured in different ways depending on the situation. For an input device, the **interface latency** is the time between when new input is available, and the time when the data is transferred into memory. For an output device, the interface latency is the time between when the output device is idle, and the time when the interface writes new data. A **real time** system is one that can guarantee an upper bound on the interface latency. The following table illustrates specific ways to calculate latency. In each case, however, latency is the time between when the need arises to the time the need is satisfied.

The time a need arises	The time the need is satisfied
new input is available	the input data is read
new input is available	the input data is processed
output device is idle	new output data is written
sample time occurs	ADC is triggered, input data
periodic time occurs	output data, DAC is triggered
control point occurs	control system executed

Table 13.1. Interface latency is a measure of the response time of the computer to a hardware event.

We can also define **device latency** as the response time of the external I/O device. For example, if we request that a certain sector be read from a disk, then the device latency is the time it takes to find the correct track and spin the disk (seek) so the proper sector is positioned under the read head. After we send an image file to the printer, the device latency is the time it takes to actually print the image.

If we consider the busy/done I/O states shown in Figure 13.1, the latency is the time from *busy to done* state transition to the time of the *done to busy* state transition. Sometimes we are interested in the worst case (maximum) latency and sometimes in the average. If we can put an upper bound on the latency, then we define the system as real time. A number of applications involve performing I/O functions on a fixed interval basis. In a data acquisition system, the ADC is triggered (a new sample is requested) at the desired sampling rate.

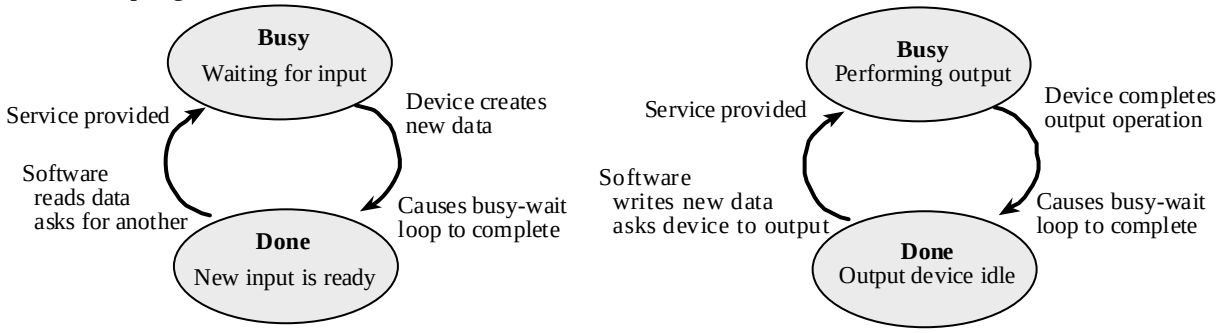


Figure 13.1. I/O devices set a flag when it needs software service, software clears the flag.

Checkpoint 13.1: *What is the difference between bandwidth and latency?*

Observation: *The same trigger flag you would have used for busy-waiting or interrupting will be used to initiate direct memory access.*

2. High speed applications

Before introducing the various solutions to a high-speed I/O interface, we will begin by presenting some high speed applications.

2.1. Mass Storage

The first application is mass storage including floppy disk, hard disk, tape, and CD ROM. Writing data to disk with these systems involves

1) establishing the physical location to write, record head at the proper block, sector, track…
2) specifying the block size
3) waiting for the physical location to arrive under the record head
4) transmitting the data from memory to the disk.

Reading data from disk with these systems is similar and involves

1) establishing the physical location to read, read head at the proper block, sector, track, …
2) specifying the block size
3) waiting for the physical location to arrive under the read head
4) receiving the data from the disk and transferring it to memory.

Under most situations the size of the data block transferred is fixed. The time to locate the physical location is called the **seek time**. The seek time has a significant impact on the disk performance. First the data must be found (seek), then the data must be transferred. The bandwidth depends on the rotation speed of the disk and the information density on the medium. The transfer rates vary according to the physics of the drive

A 7200 RPM hard drive can read at 70 megabytes/sec
A 52X CD-ROM has a peak bandwidth of 7.8 megabytes/sec
A 16X DVD writer can handle 22 megabytes/sec
A Class 2 SD card can write data at 2 megabytes/sec
An SATA channel can transfer up to 300 megabytes/sec

Original CD drives could read data at 150 kilobytes per second, and as faster drives arrived, manufacturers referred to their read speeds as a multiple of the original speed, referred to as X. Therefore, a 2X CD drive reads data at 300 kilobytes/sec. For DVDs the speeds are 9 times faster than CDs. I.e., a 1X DVD can read/write at 1,385,000 bytes/sec. Therefore, a 16X DVD can transfer at 16 times faster.

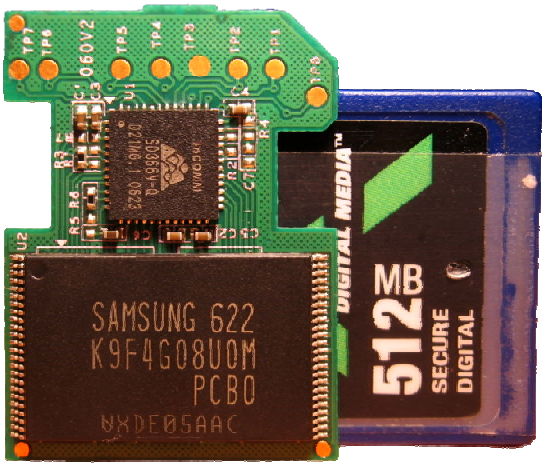


Figure 13.2. SD cards have a NAND flash and a controller..

SD Cards and SDHC Cards have Speed Class Ratings defined by the SD Association. The SD Speed Class Ratings specify the following minimum write speeds based on "the best fragmented state where no memory unit is occupied": [www. SDCard.org]

Basic	900 kBytes/sec	6x in CD units
Class 2	2 MByte/s	13x in CD units
Class 4	4 MByte/s	26x in CD units
Class 6	6 MByte/s	40x in CD units

There is a wide range of disk speeds, but it is important to note that for most situations, the disk bandwidth will be less than the computer bus bandwidth, but greater than the maximum bandwidth that a software-controlled interface can achieve. If the disk interface is not buffered, then the interface must response to each data byte at a rate as fast the peak disk bandwidth. For example in a disk read, once the data becomes available, the interface must capture it and store it in memory, before the next data becomes available. If we do not meet the response time requirement in the disk interface, the rotation speed will have to be reduced. Notice because of the seek time (time for the physical location to arrive under the head), the average and peak bandwidth will be quite different. Also notice that without buffering, the maximum interface latency will be inversely related to the peak bandwidth.

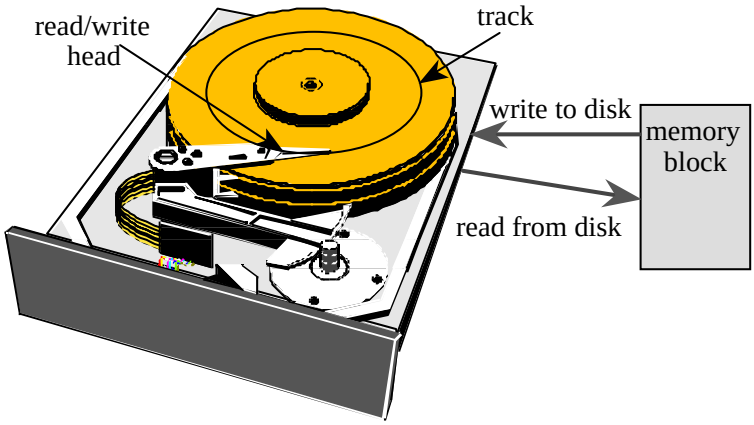


Figure 13.3. Data to/from the read/write head of a hard drive has a high bandwidth.

Checkpoint 13.2: *What happens if we are reading data off a hard drive do not satisfy the latency requirement? In other words, the read data is ready, but we do not capture it in time.*

2.2. High Speed Data Acquisition

Examples of high speed data acquisition are CD-quality sound recording (16-bit, 2 channel, 44 kHz), real time digital image recording and digital scopes (8-bit 200 MHz). Sound recording actually has two high speed data channels: one for recording into memory, and a second for storing the memory data on hard disk or CD. A spectrum analyzer combines the high speed data acquisition of a digital scope, with the Discrete Fourier Transform (DFT) to visualize the collected data in the frequency domain. In the context of this chapter, we will define a high speed data acquisition as one that samples faster than a software-controlled interface would allow.

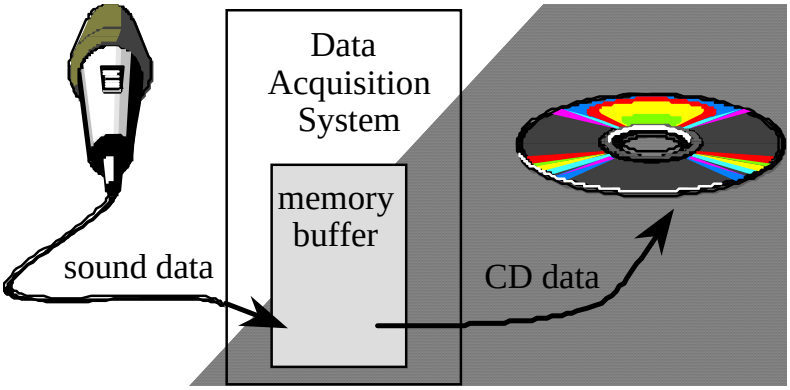


Figure 13.4. Sound data from the microphone is stored in memory, processed, then saved on CD.

Checkpoint 13.3: *What happens to the sound recording if data is missed?*

2.3. Video Displays

Real time generation of TV or video images requires an enormous data bandwidth. Consider the information bandwidth required to maintain an image on a graphics display. A VGA image is 256 colors (8-bit), 480 rows, 640 columns and is refreshed at about 60Hz. Calculating the bandwidth in bytes/sec, we get 1*480*640*60, which is 18,432,000 bytes/sec. Luckily, we don't have to communicate each pixel for each image, but rather can just transmit the changes from the previous image. In order to achieve the necessary bandwidth, video interface hardware will use a combination of DMA and dual port memories.

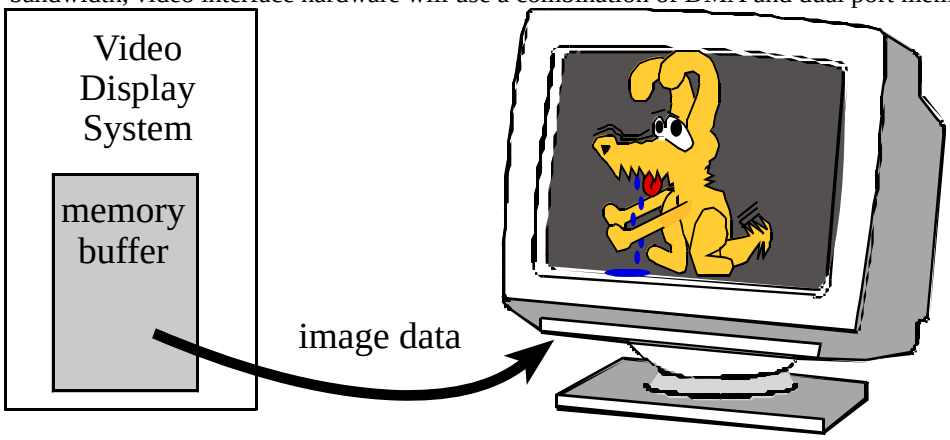


Figure 13.5. Image data from memory is displayed on the graphics screen.

2.4. High Speed Signal Generation

Examples of high speed signal generation are CD-quality sound playback (16-bit, 2 channel, 44 kHz), and real time waveform generation. Sound playback also has two high speed data channels: one for loading sound data into memory from CD, and a second for playing the memory data out to the speakers. In the context of this chapter, we will define a high speed interface as one that samples faster than a software-controlled interface would allow.

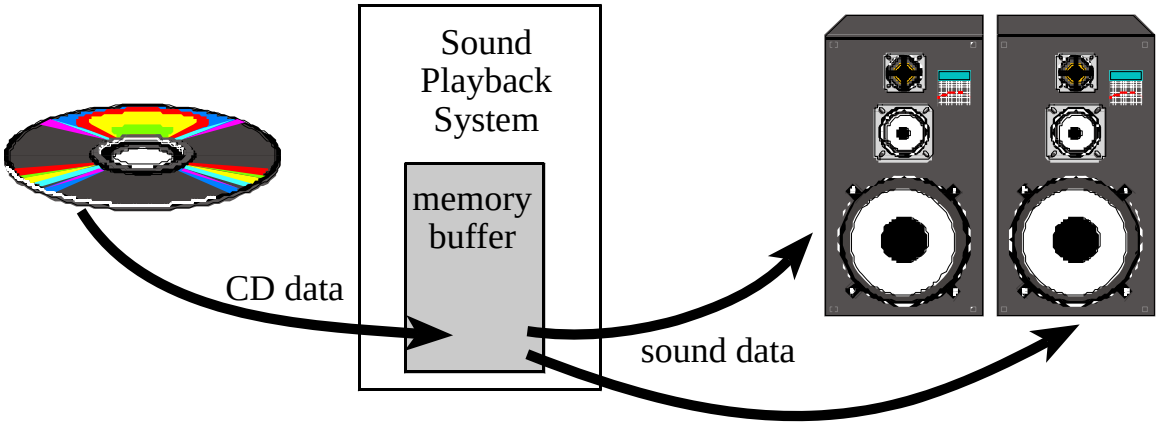


Figure 13.6. Sound data from the CD is loaded in memory, processed, then output to the speakers.

2.5. Network Communications

For many networks the communication bandwidth of the physical channel will exceed the ability of the software to accept or transmit messages. For these high speed applications, we will look for ways to decouple the software that creates outgoing messages and processes incoming messages from the hardware that is involved in the transmission and reception of individual bits. Because the network load will vary, the average bandwidth (determined by how fast the transmission software can create outgoing messages and the reception software can process incoming messages) will be slower than the peak/maximum bandwidth that is achieved by the network hardware during transmission. This mismatch allows one network to be shared among multiple potential nodes.

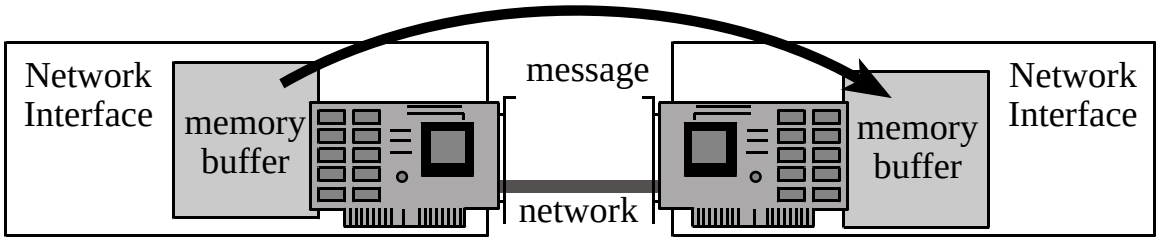


Figure 13.7. Message data is transmitted from the buffer of one computer to the buffer of another.

Checkpoint 13.4: *What happens in a communication system when packets are lost?*

3. General Approaches to High Speed Interfaces

3.1. Hardware FIFO

If the software-controlled interface can handle the average bandwidth, but fails to satisfy the latency requirements, then a hardware FIFO can be placed between the I/O device and the computer. A common application of the hardware FIFO is the serial interface. Assume in this situation, the average serial bandwidth is low enough for the software to read the data from the serial port and write it to memory. The latency requirement of a USART serial input port like the STM32F103 is the time it takes to transmit one data frame. To reduce this latency requirement (without changing the average bandwidth requirement) we can add a hardware FIFO between the receive shift register and the receive data register, as illustrated in the following figure.

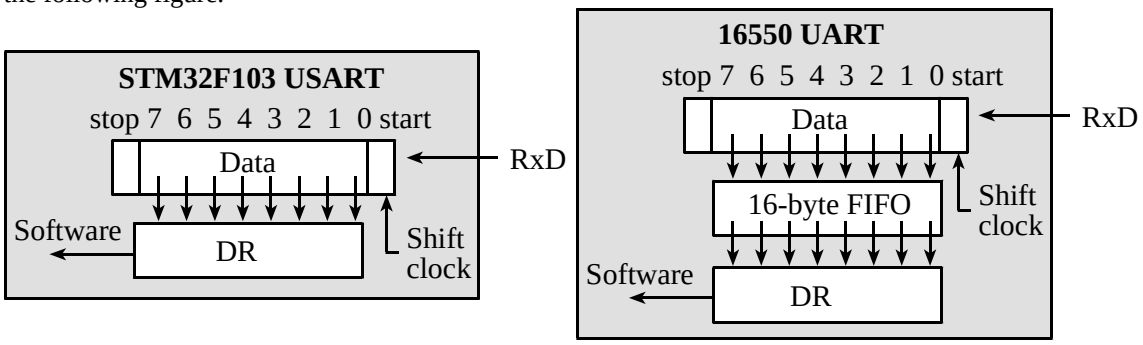


Figure 13.8. The 16550 UART employs a 16 byte FIFO to reduce the latency requirement of the interface.

Observation: *With a serial port that has a shift register, a FIFO of size n, and one data register, the latency requirement of the input interface is the time it takes to transmit n+1 data frames.*

A hardware FIFO, placed between the SCI data register and the transmit shift register, allows the software to write multiple bytes of data to the interface, and then perform other tasks while the frames are being sent.

3.2. Dual Port Memory

One approach that allows a large amount of data to be transmitted from the software to the hardware is the dual port memory. A dual port memory allows shared access to the same memory between the software and hardware. For example the software can create a graphics image in the dual port memory using standard memory write operations. At the same time the video graphics hardware can fetch information out of the same memory and display it on the computer monitor. In this way, the data need not be explicitly transmitted from the computer to the graphics display hardware. To implement a dual port memory, there must be a way to arbitrate the condition when both the software and hardware wish to access the device simultaneously. One mechanism to arbitrate simultaneous requests is to halt the processor using a MRDY signal so that the software temporarily waits while the video hardware fetches what it needs. Once the video hardware is done, the MRDY signal is released and the software resumes. Notice that except for the access conflict, both the software and graphics hardware can operate simultaneously at full speed.

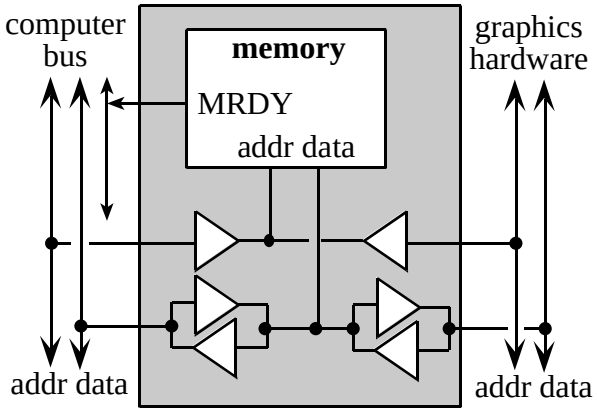


Figure 13.9. A dual port memory can be accessed by two different modules.

Checkpoint 13.5: *Explain how the bidirectional tristate buffers connected to the memory data lines in Figure 13.9 work.*

3.3. Bank-Switched Memory

Another approach similar to the dual port memory is the bank-switched memory. A bank-switched memory also allows shared access to the same memory between the software and hardware. The difference between bank-switched and dual port, is the bank-switched memory has two modes. In one mode (M=1), the computer has access to memory bank A, and the I/O hardware has access to memory bank B. In the other mode (M=0), the computer has access to memory bank B, and the I/O hardware has access to memory bank A. Because access is restricted in this way, there are no conflicts to resolve.

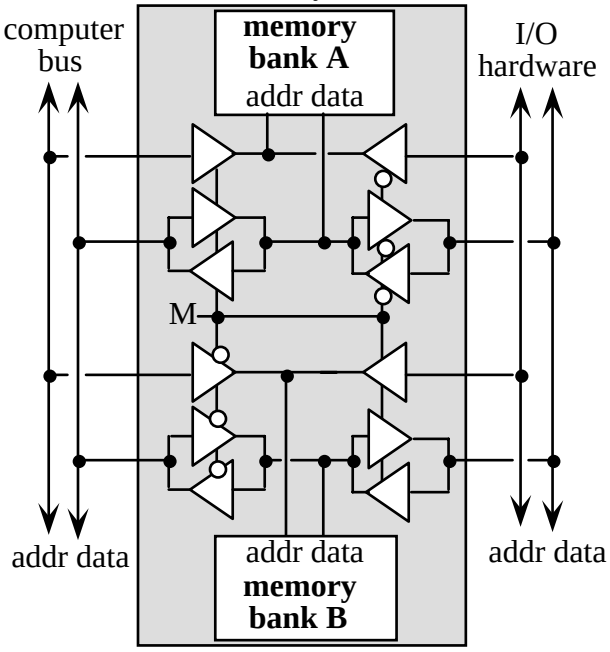


Figure 13.10. A bank-switched memory can be accessed by two different modules, one at a time.

Observation: *With a bank-switched memory, the latency requirement of the software is the time it takes the hardware to fill (or empty) one memory bank.*

Many high speed data acquisition systems employ this approach. The ADC hardware can write into one bank while the computer software processes previously collected data in the other. When the ADC sampling hardware fills a bank, the memory mode is switched, and the software and hardware swap access rights to the memory banks. In a similar way, a real time waveform generator or sound playback system can use the bank-switched approach. The software creates the data and stores it into one bank, while the

hardware reads data from the other bank that was previously filled. Again, when the hardware is finished, then the memory bank mode is switched.

Checkpoint 13.6: *How would you redesign the bank-switched memory in Figure 13.10 if the communication channel were simplex (data flows left to right only)?*

4. Fundamental Approach to DMA

With a software-controlled interface (gadfly or interrupts) if we wish to transfer data from an input device into RAM, you must first transfer it from input to the processor, then from the processor into RAM. In order to improve performance, we will transfer data directly from input to RAM or RAM to output using **D**irect **M**emory **A**ccess, DMA. Because DMA bandwidth can be as high as the bus bandwidth, we will use this method to interface high bandwidth devices like disks and networks. Similarly, because the latency of this type of interface depends only on hardware and is usually just a couple of bus cycles, we will use DMA for situations that require a very fast response. On the other hand, software-controlled interfaces have the potential to perform more complex operations than simply transferring the data to/from memory. For example, the software could perform error checking, convert from one format to another, implement compression/decompression, and detect events. These more complex I/O operations may preclude the usage of DMA.

4.1. DMA Cycles

During a DMA read cycle, the processor is halted, and data is transferred from memory to output device. During a DMA write cycle, the processor is halted and data is transferred from input device to memory. In some DMA interfaces, two DMA cycles are required to transfer the data. The first DMA cycle brings data from the source into the DMA module, and the second DMA cycle sends the data to its destination.

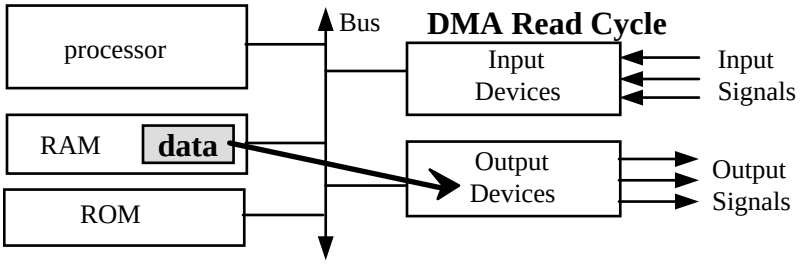


Figure 13.11. A DMA read cycle copies data from RAM or ROM to an output device.

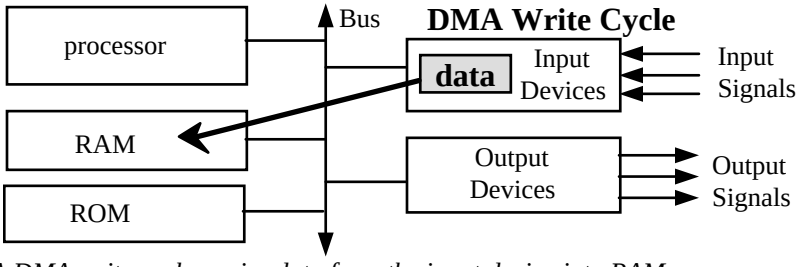


Figure 13.12. A DMA write cycle copies data from the input device into RAM.

4.2. DMA Initiation

We can classify DMA operations according to the event that initiates the transfer. A *software initiated* transfer begins with the program setting up and starting the DMA operation. Using DMA to transfer data from one memory block to another greatly speeds up the function. The efficiency of memory block transfers is very important in larger computer systems. Benchmarks on many microcontrollers show that even for small block sizes it is faster to initialize a DMA channel and perform the transfer in hardware. As the block size increases the performance advantage of DMA hardware over traditional software becomes more dramatic.

Most DMA applications involve a *hardware initiated* DMA transfer. For an input device, the DMA is triggered on new data available. For an output device, the DMA is triggered on output device idle. For periodic events, like data acquisition and signal generation, the DMA is triggered by a periodic timer. The difference with DMA is that the servicing of the I/O need will be performed by the DMA controller hardware without software having to explicitly transfer each byte.

4.3. Burst versus Cycle Steal DMA

When the desired I/O bandwidth matches the computer bus bandwidth, then the computer can be completely halted, while the block of data is transferred all at once. Once an input block is ready, a burst mode DMA is requested, the computer is halted, and the block is transferred into memory.

Figure 13.13. An input block is transferred all at once during burst mode DMA.

Figure 13.13 describes an input interface, but the same timing occurs on an output interface using burst mode DMA. For an output interface, the DMA is requested when the interface needs another block of data. During the burst mode DMA, the computer is halted, and an entire block is transferred from memory to the output device.

If the I/O bandwidth is less than the computer bus bandwidth, then the DMA hardware will steal cycles and transfer the data one DMA cycle at a time. In cycle steal mode, the software continues to run, although a little bit slower. In either case the processor is halted during the DMA cycles.

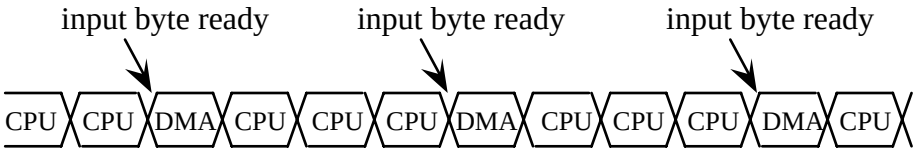


Figure 13.14. Each time an input byte is ready it is transferred to memory using cycle steal DMA.

The above figure describes an input interface, but the same timing occurs on an output interface using cycle steal mode DMA. For an output interface, the DMA is requested when the interface needs another byte of data. During the cycle steal DMA, one byte is transferred from memory to the output device.

Observation: *Some computers must finish the instruction before allowing a burst-DMA. In this situation, the latency will be higher than cycle steal DMA, which does not need to finish the current instruction.*

Since most I/O bandwidths are indeed less than the memory bandwidth, one technique to enhance speed is I/O buffering. In this approach a dedicated I/O memory buffer exists in the I/O interface hardware. This buffer is like the bank-switched memory discussed earlier. For example, on a hard disk read block operation, raw data comes off the disk and into the buffer. During this time the processor is not halted. When the buffer is full, burst DMA is used to transfer the data into the system memory. Similarly on a hard disk write block operation, the software initiates a burst DMA to transfer data from system memory into the I/O buffer. Once full, the I/O interface can write the data onto the disk.

Checkpoint 13.7: *What is the maximum latency in a cycle steal DMA system?*

4.4. Single Address versus Dual Address DMA

Some computer systems allow the transfer of data between the memory and I/O interface to occur in one bus cycle, while others need two bus cycles to complete the transfer. In a *single address DMA cycle*, the address and R/W line dictate the memory function to be performed and the I/O interface is sophisticated enough to know it should participate in the transfer.

In this single address example, the disk interface is reading bytes from a floppy disk. Cycle steal mode will be used because the floppy disk bandwidth is slower than the bus. When a new byte is available,

Request will be asserted and this will request a DMA cycle from the DMA controller. The DMA controller will temporarily suspend the processor and drive the address bus with the memory address and the R/W to write. During this cycle the DMA controller will respond to the floppy interface by asserting the **Ack**. The floppy uses the **Ack** (ignoring the address bus and R/W) to know when to drive its data on the bus.

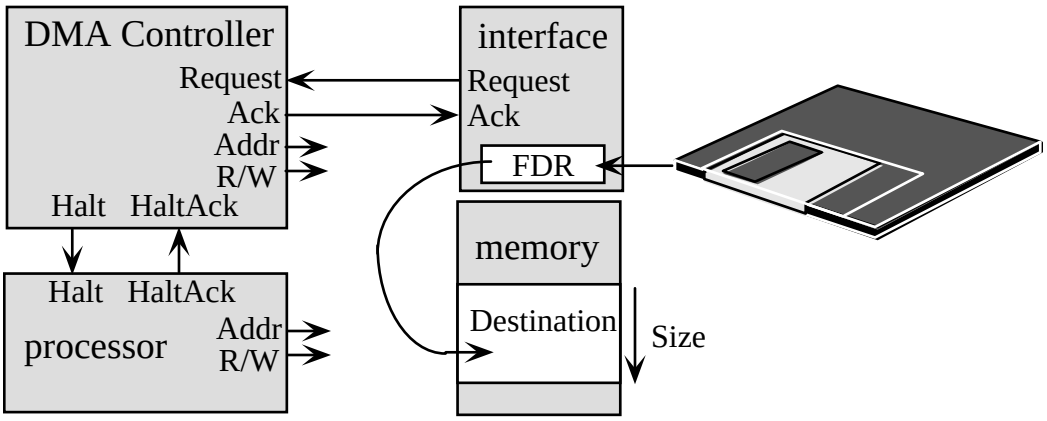


Figure 13.15. Block diagram showing the modules involved in a floppy disk read.

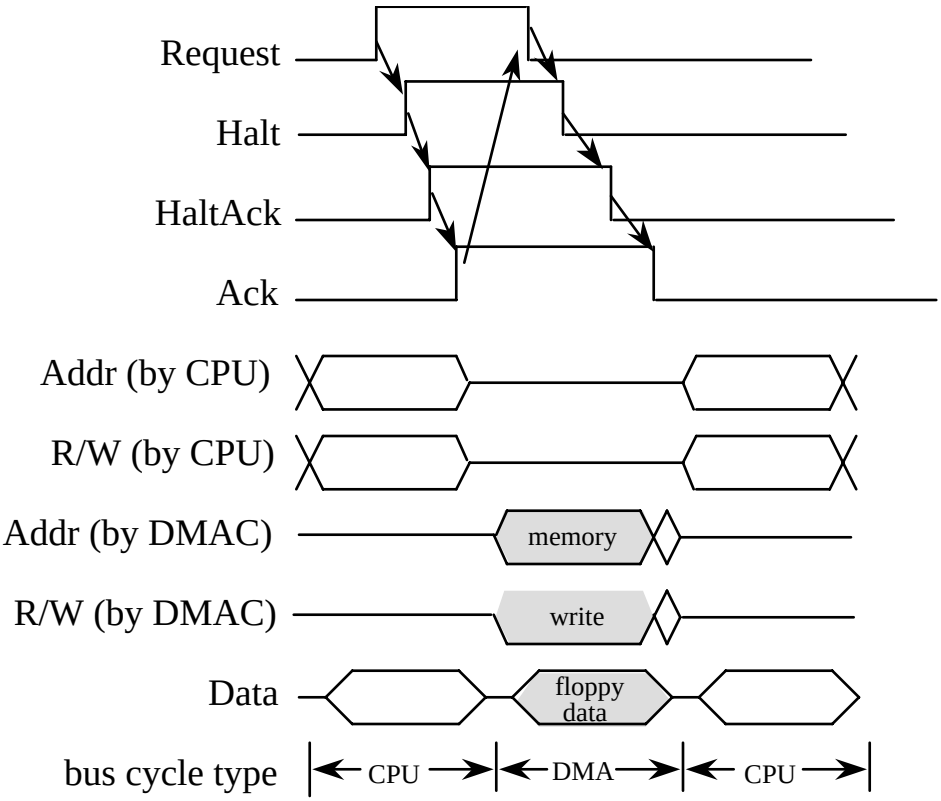


Figure 13.16. Timing diagram of a single address DMA-controlled floppy disk read.

In a *dual address DMA cycle*, two bus cycles are required to achieve the transfer. In the first cycle the data is read from the source address, and copied in the DMA controller. During the first cycle the address bus contains the source address and R/W signifies read. The information from the data bus is saved in the **Temp** register within the DMA controller. In the second cycle, the data is transferred to the destination address. During the second cycle, the address bus contains the destination address, the data bus has the **Temp** data, and R/W signifies write.

In this dual address example, the SPI interface is receiving bytes from a synchronous serial network. Cycle steal mode will be used because the SPI bandwidth is slower than the bus. When a new byte is available, **Request** will be asserted and this will request a DMA cycle from the DMA controller. The DMA controller will temporarily suspend the processor and first drive the address bus with the SPI data register address (R/W=read), then in the second cycle the DMA controller will drive the address bus with the memory address (R/W=write). The SPI knows it has been serviced, because its data register has been read.

Observation: *The single address DMA is twice as fast as dual address DMA.*

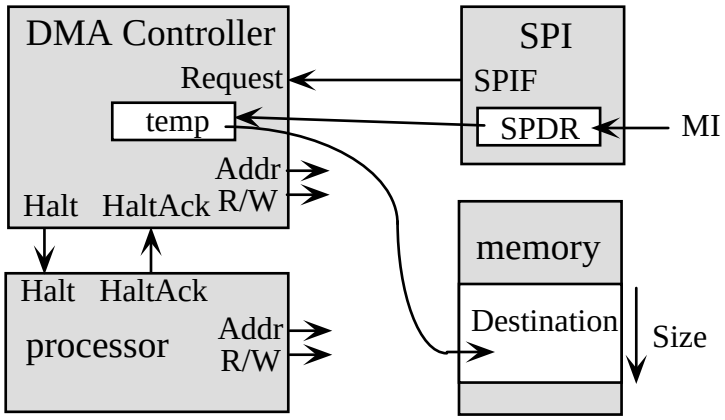


Figure 13.17. Block diagram showing the modules involved in a SPI read.

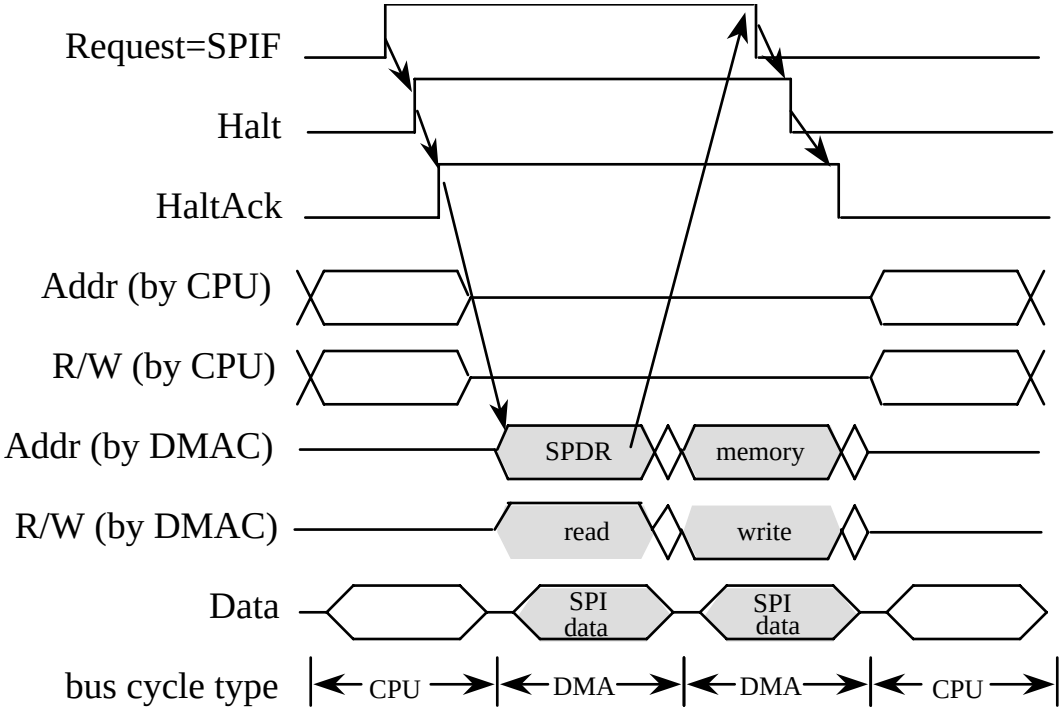


Figure 13.18. Timing diagram of a dual address DMA-controlled SPI read.

Observation: *The dual address DMA can be used with I/O devices not configured to support DMA.*

5. DMA programming approach

Although DMA programming varies considerably from one system to another, there are a few initialization steps that most require. The following table lists the mode parameters that must be set to utilize DMA. There are two categories of DMA programming: initialization and completion. During initialization, the software sets the DMA parameters, so that the DMA will begin.

parameter	possible choices
what initiates the DMA	software trigger, input device, output device, periodic timer
type	burst versus cycle steal
autoinitialization mode	single event or continuous transfer
precision	8-bit byte or 16-bit word or 32-bit
mode	single address or dual address
priority	Should software completely halt? Should interrupts be serviced during DMA?
synchronization	set gadfly flag, or interrupt on block transfer complete

Table 13.2. DMA initialization usually involves specifying these parameters.

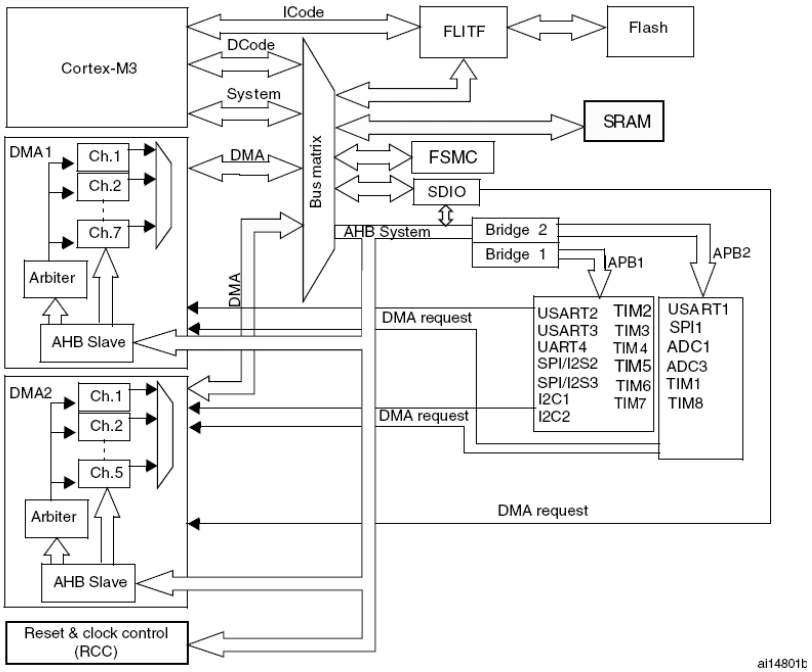
At the end of a block transfer, a done flag is set and a number of additional actions may occur. If the system is armed, an interrupt can be generated. At the end of a block transfer in a continuous transfer DMA, the parameters are autoinitialized, so that the DMA process continues indefinitely. This next table lists additional parameters we will need to initialize

parameter	definition
source address	address of the module (memory or input) that generates the data
inc/dec/static	automatically +1/-1/+0[1] the source address after each transfer
destination address	address of the module (memory or output) that accepts the data
inc/dec/static	automatically +1/-1/+0 the destination address after each transfer
block size	fixed number of bytes to be transferred

Table 13.3. More DMA initialization parameters.

6. DMA on the STM32F103

6.1. STM32F103 DMA block diagram



[1] Obviously the DMA controller will +2/-2/+0 when the precision is 16 bits

The DMA controller performs direct memory transfer by sharing the system bus with the Cortex™-M3 core. The DMA request may stop the CPU access to the system bus for some bus cycles, when the CPU and DMA are targeting the same destination (memory or peripheral). The bus matrix implements round-robin scheduling, thus ensuring at least half of the system bus bandwidth (both to memory and peripheral) for the CPU.

Interrupts on
- Transfer complete (TCIF/TCIE)
- Half-transfer flag (HTIF/HTIE) to implement double buffering
- Transfer error (TEIF/TEIE)

Separate RAM, Flash buses so DMA does not slow down instruction fetches

Table 58. Summary of DMA1 requests for each channel

Peripherals	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5	Channel 6	Channel 7
ADC1	ADC1						
SPI/I²S		SPI1_RX	SPI1_TX	SPI2/I2S2_RX	SPI2/I2S2_TX		
USART		USART3_TX	USART3_RX	USART1_TX	USART1_RX	USART2_RX	USART2_TX
I²C				I2C2_TX	I2C2_RX	I2C1_TX	I2C1_RX
TIM1		TIM1_CH1	TIM1_CH2	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	
TIM2	TIM2_CH3	TIM2_UP			TIM2_CH1		TIM2_CH2 TIM2_CH4
TIM3		TIM3_CH3	TIM3_CH4 TIM3_UP			TIM3_CH1 TIM3_TRIG	
TIM4	TIM4_CH1			TIM4_CH2	TIM4_CH3		TIM4_UP

6.2. STM32F103 Continuous ADC sampling from Blinky

DMA configuration **`DMA1_Channel1->CCR = 0x2520`**

- Bit 14 MEM2MEM: Memory-to-memory transfers
- Bits 13:12 PL[1:0]: Four levels of priority
- Bits 11:10 MSIZE[1:0]: Memory 8/16/32-bit data widths
- Bits 9:8 PSIZE[1:0]: Peripheral 8/16/32-bit data widths
- Bit 7 MINC: Memory increment mode (increment or not)
- Bit 6 PINC: Peripheral increment mode (increment or not)
- Bit 5 CIRC: Circular mode to implement continuous ADC sampling
- Bit 4 DIR: Data transfer direction
- Bit 3 TEIE: Transfer error interrupt enable
- Bit 2 HTIE: Half transfer interrupt enable
- Bit 1 TCIE: Transfer complete interrupt enable
- Bit 0 EN: Channel enable

```
void adc_Init (void) {
//  GPIOA->CRL &= ~0x0000000F;            // set PIN1 as analog input (see stm32_Init.c)
  RCC->AHBENR |= (1<<0);                  // enable peripheral clock for DMA
  DMA1_Channel1->CMAR  = (u32)&ADC_ConvertedValue;// set channel1 memory address
  DMA1_Channel1->CPAR  = (u32)&(ADC1->DR); // set channel1 peripheral address
  DMA1_Channel1->CNDTR = 1;               // transmit 1 word
  DMA1_Channel1->CCR   = 0x00002520;      // configure DMA channel
  DMA1_Channel1->CCR  |= (1 << 0);        // DMA Channel 1 enable
  RCC->APB2ENR |= (1<<9);                 // enable peripheral clock for ADC1
  ADC1->SQR1  = 0x00000000;               // only one conversion
  ADC1->SMPR2 = 0x00000028;               // set sample time channel1 (55,5 cycles)
  ADC1->SQR3  = 0x00000001;               // set channel1 as 1st conversion
  ADC1->CR1   = 0x00000100;               // use independent mode, SCAN mode
  ADC1->CR2   = 0x000E0103;               // use data align right, continuous conversion
  // EXTSEL = SWSTART  enable ADC, DMA mode, no external Trigger
  ADC1->CR2  |=  0x00500000;              // start SW conversion
}
```