6 Direct Memory Access (DMA)

DMA technique is used to transfer large volumes of data between I/O interfaces and the memory.

Example: Disk drive controllers, graphics cards, network cards and sound cards.

DMA can also be used

for intra-chip data transfer in multi-core processors,

for "memory to memory" copying or moving of data.

Remember simple programmed I/O and interrupt-driven I/O require the active intervention of the processor to transfer data, and any data transfer must traverse a path through the processor.
The CPU reads from I/O interface (or memory) and then writes to the memory (or I/O interface).
In DMA technique a hardware module, called **DMA Controller (DMAC)**, is used that can act as a CPU by generating addresses and initiating memory read or write cycles.

The CPU configures the DMAC and delegates the I/O operations to it.

Now the CPU can continue with its job.

All I/O operations are performed by the DMAC by taking the control of the system bus. The data will not flow through the CPU.

6.1 Overview of DMA:

When there is a need for a data transfer the I/O interface signals the DMAC.

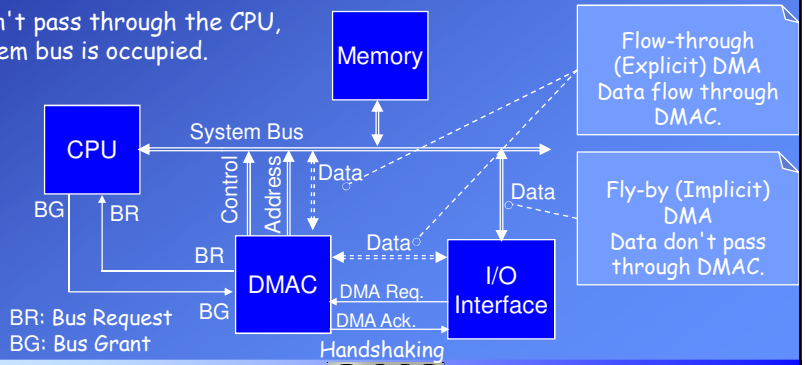
DMAC requests the system bus from the CPU (BR).

The CPU completes the current bus cycle (not the instruction), isolates itself from the system bus, and responds to the DMAC by activating BG.

Now, it is the responsibility of the DMAC to generate all bus signals and performing the transfer.

The CPU can perform its internal operations.

The data don't pass through the CPU, but the system bus is occupied.



6.2 Types of DMA Controllers:

a) **Flow-through** (Explicit) DMAC: The data, transferred between memory and I/O interface pass through the DMAC.

The DMAC first reads data into an internal register and then writes it to the destination.

b) **Fly-by (Implicit)** DMAC: The data don't pass through the DMAC.

After the DMA controller gains access to the bus, it puts the source (or destination) address and other control signals (R/W, VMA, etc.) out.

It activates the memory and the I/O interface at the same time.

So, it initiates a read and a write cycle simultaneously. The data is read from the source address, and written to the destination, in **one clock cycle**.

Therefore, fly-by technique can transfer data faster than the flow-through.

But.

This technique implies that either the source or destination does not require an address, since it is very unlikely that both would use the same.

Therefore, a fly-by DMAC can only transfer data between an I/O port and a memory address,
but not between two I/O ports or two memory locations.

6.3 DMA Transfer Modes:

a) Burst mode (Block Transfer Mode): Once the DMA controller takes the control of the system bus, it transfers all bytes of data in the data block before releasing control of the system bus back to the CPU.

The size of the block is predetermined in the initialization process by the CPU.

The CPU can be inactive for a relatively long time.

This mode is useful for loading programs or data files into memory, because the CPU needs these data to continue its work.

b) Cycle stealing: DMAC requests the bus as in the burst mode. When it is granted to access the bus by the CPU, the DMAC transfers only one word and gives the control of the bus back to the CPU.

DMAC continually issues requests, transferring one word of data per request, until it has transferred its entire block.

This technique is suitable for systems in which the CPU should not be disabled for a long time.

The CPU needs to access the memory in instruction fetch, operand fetch cycles and if necessary for operand write operations.

In decode and execution cycles the CPU can operate without accessing the memory in parallel with the DMAC.

DMA Transfer Modes: (cont'd)

b) Cycle stealing: (cont'd)

This mode interleaves instruction execution by the CPU and data transfer by the DMAC.

The data block is transferred slower than in burst mode.

c) Transparent mode (Hidden DMA):

The DMAC (or an additional hardware unit) monitors the CPU and the bus is used by the DMAC only when the processor is not using it.

When the DMAC recognizes that the processor is executing an instruction which leaves sufficient empty clock cycles to perform a word transfer, it accesses the bus during this time.

The processor is not slowed down.

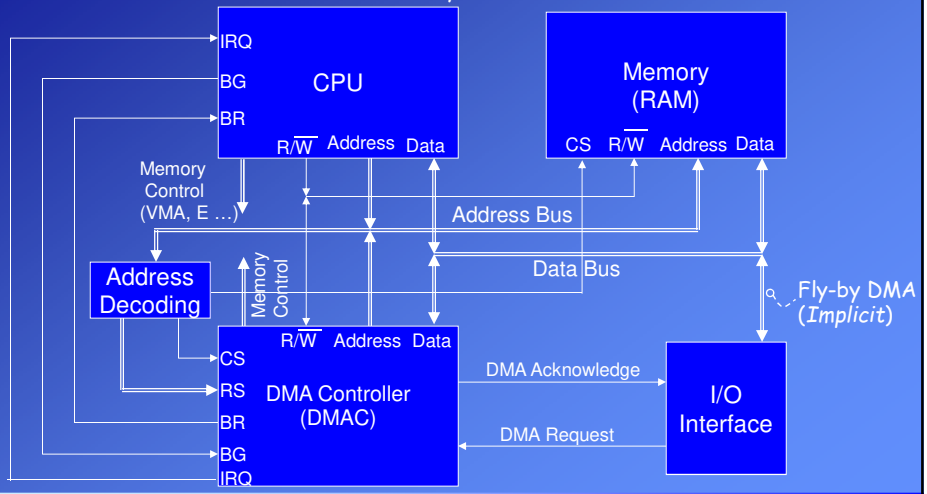
Transfer of the data block can take longer than other modes.

Disadvantage is that the hardware needed to determine when the CPU is not using the system.

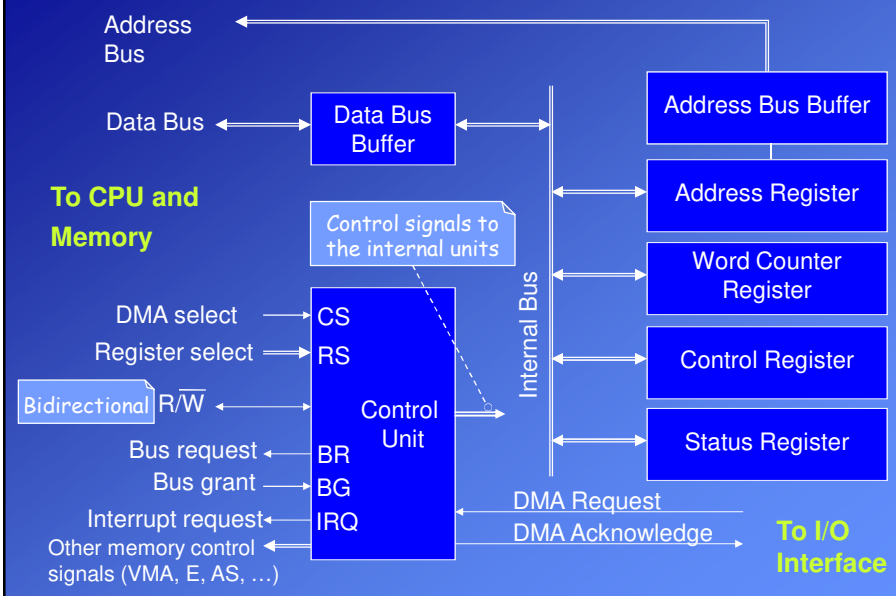
Burst and cycle stealing modes are two common transfer modes of the DMA.

6.4 CPU - DMAC Interconnection:

Before the data transfer DMAC is configured (initialized) by the CPU (necessary information is written to registers of the DMAC).
In this step DMAC act as a memory unit or an I/O interface.
After DMAC has taken the control of the system bus it acts as a CPU.



Internal structure of a DMAC:



6.5 Steps of data transfer in DMA technique

1. The CPU programs (configures) the DMA controller (DMAC).
   I/O address, memory address, read/write, amount of data, transfer mode etc. are written to the registers of the DMAC.
   In this step DMAC acts as a memory or I/O unit that is addressable by the CPU. Now, the R/W' pin of the DMAC is an input pin.
2. I/O interface sends request to DMAC by asserting DMA Request (ready to send or receive).
3. DMAC requests the bus from the CPU by asserting the BR.
4. CPU completes the current bus cycle (not the instruction),
   isolates itself from the system bus (moves to 3rd state – high impedance -),
   and grants the bus to the DMAC by asserting the BG pin.
   The CPU **cannot mask** (disable) the request of the DMAC unlike an interrupt request.

6.5 Steps of data transfer in DMA technique (cont'd)

5. Now the new bus master is the DMAC. It is the responsibility of DMAC to provide all necessary signals to address the memory as the CPU does.

   In this step the R/W pin of the DMAC is an output.

   DMAC puts the address, R/W and other necessary signals on the system bus.

   a) Fly-by (implicit) DMA:

   DMAC signals the I/O interface with DMA Acknowledge.

   The I/O interface reads the data from or puts the data on the data bus.

   b) Flow-through (explicit) DMA:

   DMAC signals the I/O interface with DMA Acknowledge.

   DMAC reads the data from the I/O interface and writes it to the memory

   or

   DMAC reads the data from the memory; signals the I/O interface with DMA Acknowledge and writes the data to the I/O interface.

6.5 Steps of data transfer in DMA technique (cont'd)

6. If the I/O interface persists in its transfer request (DMA Request), DMAC keeps the BR active.

   Burst mode: BR remains active until the whole block is completed.

   Cycle stealing: After transferring one word the DMAC deactivates the BR allowing the CPU to use the system bus and then signals the request again.

   While the DMAC is transferring the data the CPU can perform its internal operations, which don't need the access to the system bus, such as instruction decoding, operations on register (Compare to polling and interrupt-driven I/O).

7. If there is not a new request from the I/O interface (DMA Request is not active) or the entire block is transferred (the word counter of the DMAC is zero), DMAC isolates itself from the system bus, i.e. its control lines move to 3rd state (high impedance).

   DMAC deactivates BR. The CPU gets the control of the system bus back.

6.5 Steps of data transfer in DMA technique (cont'd)

8. After finishing the transfer of a block DMAC can send an interrupt request to the CPU to inform it that the job has been completed (if it is configured in this way).

   The CPU can read the internal registers of the DMAC to get information about the previous transfer (how many words have been transferred, are there any errors?).

   Here interrupts are not involved in requesting the bus and data transfer.

Example 1: I/O using DMA Technique

Problem:

Instruction cycle of a CPU has the following 5 states (cycles) with the given durations:

1. Instruction fetch: 60 ns, 2. Instruction Decode: 20 ns, 3. Operand fetch: 60 ns, 4. Execution: 30 ns, 5. Interrupt: 200 ns.

Assume that the CPU accesses the memory in the instruction fetch and operand fetch cycles but not in the decode and execution cycles.

In this system there is a 2-wire (BR, BG) DMAC that is configured to transfer words from the I/O interface to the memory using the **cycle-stealing** technique. The type of the DMAC is **fly-by** (Implicit). Data don't pass through DMAC.

Memory access time and I/O interface access times are both 50 ns.

Assume that we start a clock (Clock = 0) when the CPU begins to run the program.

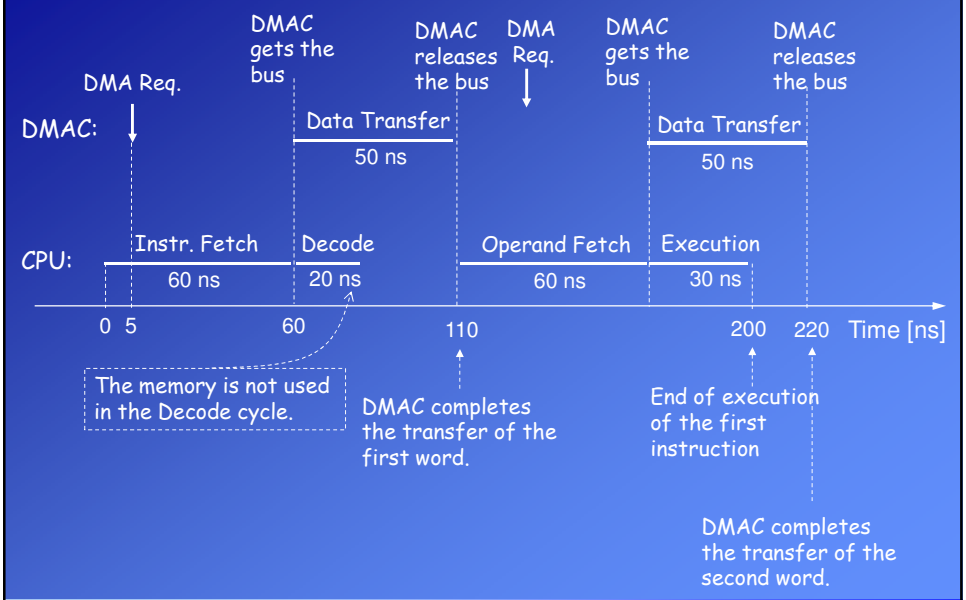
As the CPU is in the instruction fetch cycle for the first instruction (Clock = 5ns) the DMAC attempts to start the data transfer.

a. When (Clock =?) will the DMAC complete the transfer of the first word? Why?

b. When (Clock =?) will the CPU finish the first instruction? Why?

c. When (Clock =?) will the DMAC complete the transfer of all 10 words? When will the CPU complete the run of the entire program with 10 instructions?

Solution:



Solution (cont'd):

a) CPU completes the current bus cycle (Instruction fetch) and isolates itself from the system bus. The DMAC transfers the first word.

Since the type of the DMAC is **fly-by** (Implicit) data is transferred in 50 ns.

Clock= 60 + 50 = 110ns

b) Instruction decoding and execution cycles of the CPU can run in parallel with DMA transfers.

Since the DMAC uses the **cycle-stealing** technique, after the transfer of the first word, it will give the bus to CPU. After the operand fetch the CPU executes the instruction.

Clock= 60 + 50 + 60 + 30 = 200ns

c) During one instruction cycle the DMAC transfers **two words in 220 ns**.
10 words are transferred in 5 x 220 = 1100 ns.
Clock = 5 * 220 = 1100ns

During the transfer of 10 words the CPU can run 5 instructions.
After the transfer of the 10 words the CPU runs each instruction in 170 ns.
Clock= 1100 + 5*170 = 1950ns

Compare these times with the interrupt-drive I/O example in slide 5.17.

Example 2: DMA and Interrupt

Problem:

Instruction cycle of a CPU has the following 5 states (cycles) with the given durations:

1. Instruction fetch and decode: 60 ns, 2. Operand fetch: 70 ns, 3. Execution: 30 ns, 4. Result write: 60 ns, 5. Interrupt: 200 ns.

The CPU does not access memory only in the "3. Execution" cycle. CPU uses memory in the other cycles (1, 2, 4, 5).

Memory access time and I/O interface access times are both 50 ns.

In this system there is a single interrupt source (IS). The duration of its interrupt service routine (ISR) is 2500ns.

In this system there is a 2-wire DMAC that is configured to transfer words from the I/O interface to the memory using the **cycle-stealing** technique. The type of the DMAC is **fly-by** (Implicit). Data don't pass through DMAC.

Assume that we start a clock (Clock = 0) when the CPU begins to run the program.

Problem: (cont'd)

At Clock = 10ns the device IS sends an interrupt request.

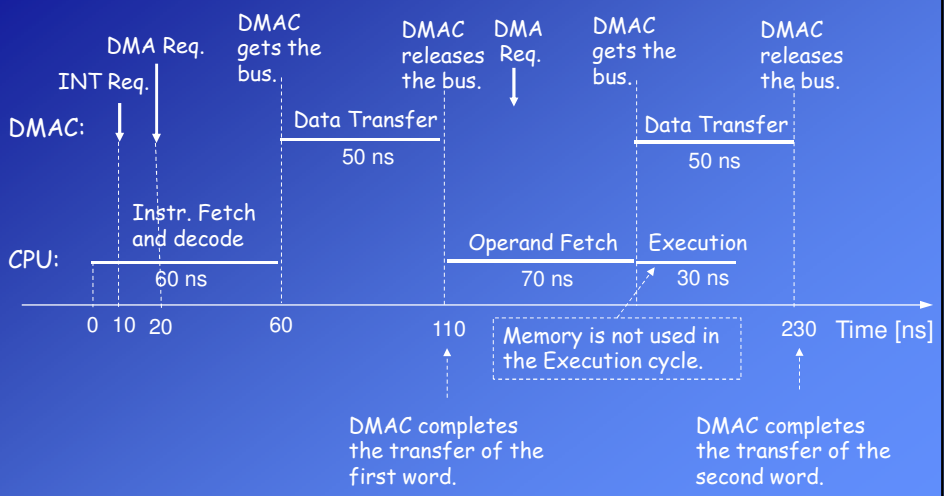
At Clock = 20ns the DMAC attempts to start the data transfer from I/O interface to memory. We assume that the I/O interface is always ready to transfer.

a. When (Clock =?) will the DMAC complete the transfer of the first, second and third words? Why?

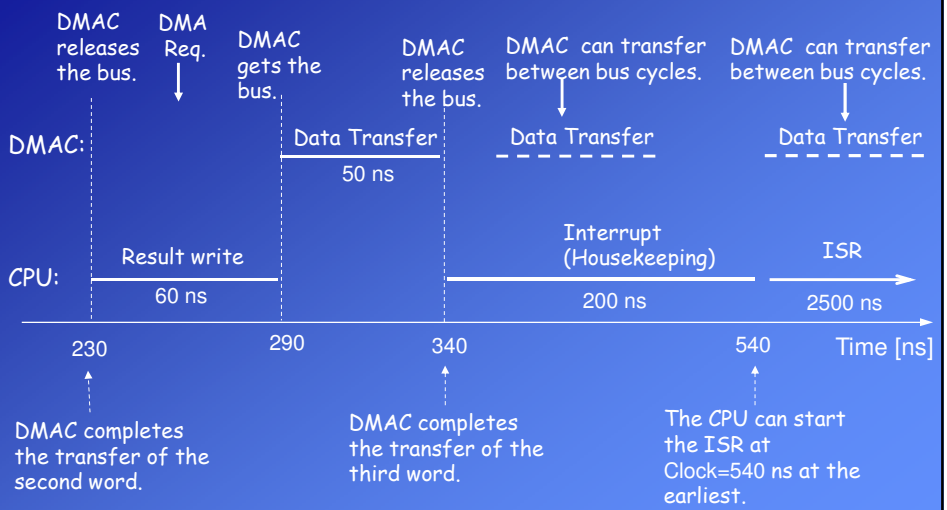
b. When (Clock =?) will the ISR of the IS start to run? Why?

c. What happens if the DMAC has still words to transfer while the ISR is running?

Solution:



Solution (cont'd):



Solution (cont'd):

Remember; interrupt requests are considered after the instruction has been completed.

If there is a request and interrupts are enabled, then the CPU enters the interrupt cycle.

In the Interrupt cycle there are many bus (memory) cycles: the vector number is read, return address and the status register are pushed into the stack.

If the DMAC is enabled it can still transfer data between these memory cycles.

The CPU can start the ISR at Clock = 540 ns at the earliest.

The ISR is a program that consists of instructions.

Therefore, the DMAC can continue to transfer during the ISR.

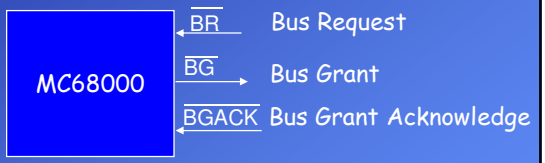
6.6 3-Wire DMA

The DMAC explained in previous section uses two control lines (BR, BG) for bus arbitration.

There are also 3-wire DMACs that are suitable for systems with many DMACs.

Example MC68000:

MC68000 uses 3 control lines for bus arbitrations.



The operation of a DMAC in a 68000-based system:

1. DMAC asserts a bus mastership request by activating BR' (active low).
2. 68000 asserts BG' as soon as possible.

   It does not mean that the processor has finished its current bus cycle.

   It only means that the processor is ready to leave the bus at the end of the current bus cycle and the next bus master can be determined.

   There can be many DMACs in the system. Which one will be the next bus master?

   An additional unit (bus arbiter) will determine the next bus master.

The operation of a DMAC in a 68000-based system: (cont'd)

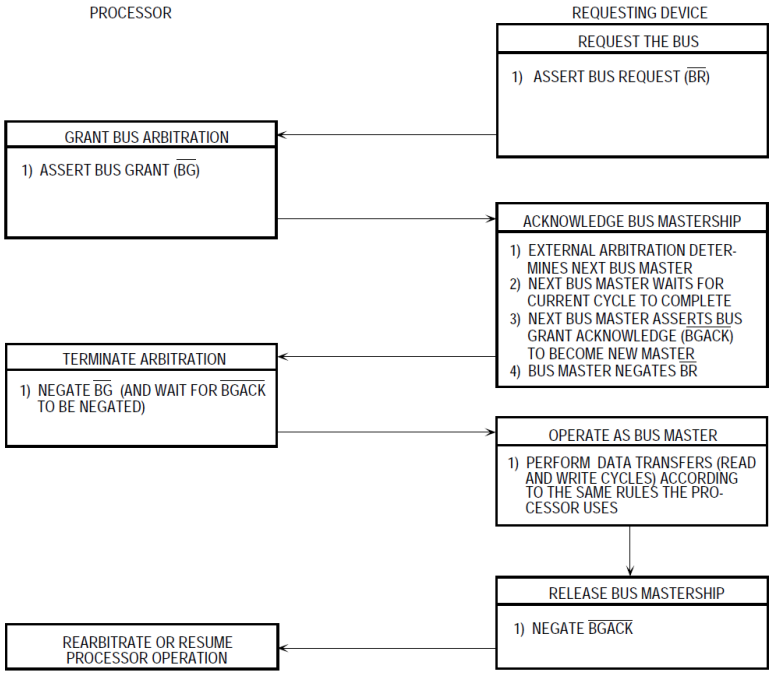
3. If there are more than one DMACs in the system the bus arbiter circuitry determines the next bus muster.

   Upon receiving BG, the requesting device waits until the current bus cycle is completed (AS, DTACK and BGACK must be negated).

   The negation of AS indicates that the previous bus master has completed its cycle. (No device is allowed to assume bus mastership while AS is asserted.)
   The negation of DTACK indicates that that neither memory nor peripherals are using the bus.
   The negation of BGACK indicates that the previous master has released the bus.
4. The new bus master asserts BGACK and keeps it active until the bus cycle(s) is complete.

   The BR of the granted DMAC is negated after BGACK asserted. Hence another DMAC in the system can issue its request.
5. When the data transfer is complete the DMAC relinquishes control of the bus by negating BGACK.

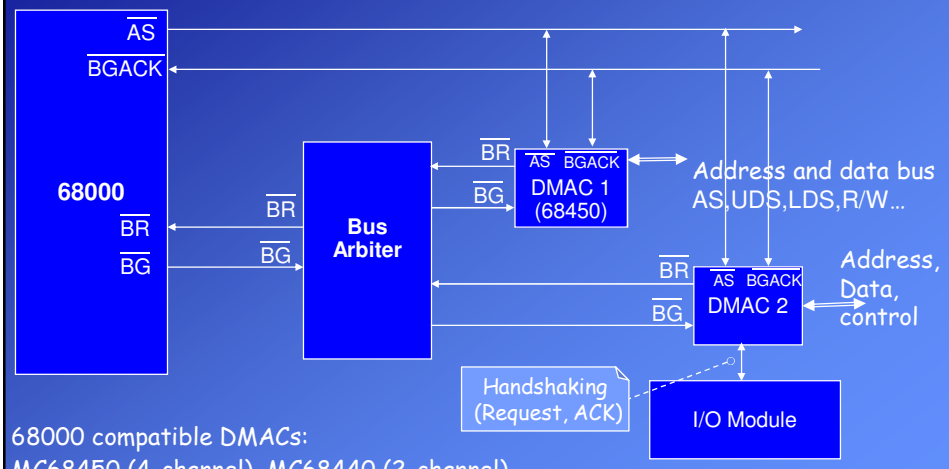
   The processor cannot access the bus while BGACK is asserted.



Bus Arbiter

In DMA operations the processor is at a lower bus priority level than the DMACs.

In systems consisting of a processor and many DMACs, a bus arbiter is necessary to determine the next bus master, if many devices issue a request simultaneously.



68000 compatible DMACs:
MC68450 (4-channel), MC68440 (2-channel)

Function of the system:

If the DMAC uses the 3-wire scheme (with BGACK) (example MC68450)

1. DMAC issues the request to the bus arbiter via BR.
2. Bus arbiter sends the request to 68000 (BR).
3. Bus arbiter gets the grant signal (BG) from 68000 and sends this signal to the requesting DMAC with the highest priority.
4. After receiving the bus grant BG, DMAC observes the AS and BGACK signals and waits until the previous bus cycle is complete and the previous bus master releases the bus.

   AS indicates the state of the current bus cycle and BGACK indicates whether the previous master has released the bus or not.
5. If the bus is free DMAC asserts BGACK to inform the processor and other DMACs about its mastership.

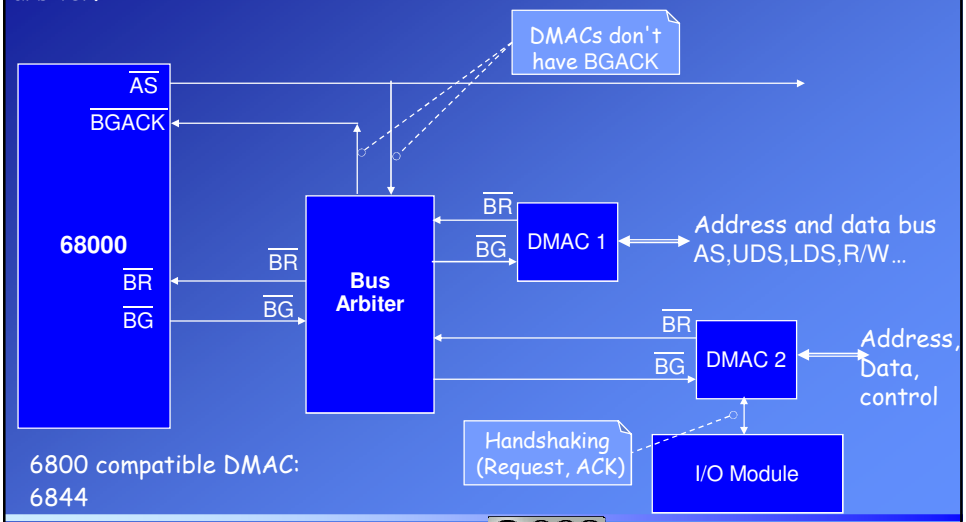
   After being the new bus master the DMAC inactivates its BR output to allow other DMACs sending their requests.

   It keeps the BGACK active until the transfer is complete.

   It is responsibility of the bus master DMAC to supply all address and control signals (AS, UDS, LDS, VMA, R/W …).
6. When all DMACs complete their transfers the BGACK input of the 68000 is negated. Now, 68000 can use the system bus again.

Connecting 2-wire (BR, BG) DMACs to 68000:

If it is necessary to connect 2-wire (BR, BG) DMACs without the BGACK to the 68000 than the operations about BGACK signal must be performed by the bus arbiter.



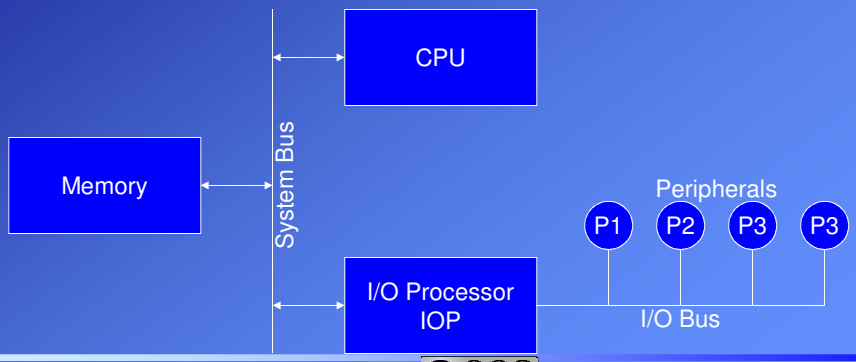
Function of the system:

If the DMAC uses the 2-wire scheme (without BGACK)

1. DMAC issues the request to the bus arbiter via BR.
2. Bus arbiter sends the request to 68000 (BR).
3. After receiving the bus grant BG from 68000 the bus arbiter waits the current bus cycle to be complete by observing AS line.
   It also waits the previous bus master DMAC (if any) to release the bus (BR).
   In this system the bus arbiter supplies BGACK signal itself, therefore it does not need to check this line.
4. Bus arbiter sends the bus grant signal (BG) to the requesting DMAC with the highest priority.
5. Bus arbiter asserts the BGACK signal to inform the 68000 that a DMAC is using the bus.
6. A two-wire DMAC keeps its request (BR) output active as long as its transfer continues.
   In this case the bus arbiter also keeps the BGACK signal active.
7. When then DMAC completes its transfer it negates its request (BR) output.
   If there is not any other pending request from another DMAC the bus arbiter negates the BGACK signal so that the 68000 can use the bus again.

6.7 The I/O Processor

The I/O processor (IOP) is a combination of a CPU, a DMAC and I/O interfaces.
Some IOPs also include a local memory.
The IOPs have a specialized instruction set tailored for I/O.
The CPU directs the IOP to execute an I/O program in memory.
The I/O processor fetches and executes these instructions without CPU intervention (DMA method).
They can perform format conversion, encryption/decryption, error correction.



6.8 Indivisible Bus Cycle

Semaphore operations are performed using **Test-And-Set (TAS)** instruction that operates in a single indivisible bus cycle.

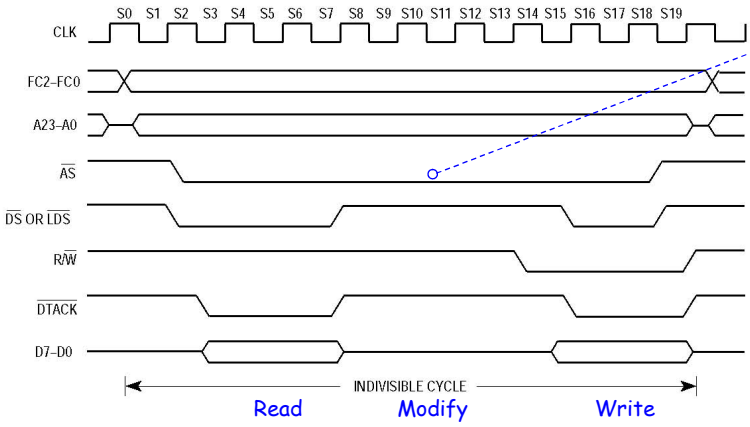
In a single instruction and in a single bus cycle

the memory is read,
the data is tested (compared to zero),
modified and
written back to the memory.

The TAS instruction and the read-modify-write cycle used by this instruction are indivisible in 2 ways:

- Three operations (read/test, modify, write) are performed in a single instruction. Therefore this operations cannot be interrupted.
- AS (address strobe) remains asserted throughout the entire cycle. Therefore DMACs or other processors cannot divide this cycle to access the memory.

Example : Read-modify-write cycle in MC6800-based systems
Timing diagram of Read-modify-write cycle:
(used by TAS "Test and set" instruction)



Read-Modify-Write Cycle Timing Diagram

AS remains asserted throughout the entire cycle. DMACs or other processors cannot divide this cycle.

TAS **Test and set an operand** (MC6800)
Format: TAS <ea>
Operation: [CCR] ← tested([operand <ea>]); [destination: <ea>(7)] ← 1
Tests the operand: According the value of data, Z and N flags are modified.
The most significant bit (7) of the operand is set to 1 (made negative).
These operations are indivisible (single instruction, single bus cycle).
It is used for semaphore operations.

Example1: Critical section access without TAS instruction (dangerous!)

```
TEST      TST.B    FLAG      (Divisible)
          BMI      TEST      If negative (MSB=1)
CRITICAL  OR.B     #$80,FLAG Semaphore  is set
          ......             Critical section
          ......
END       CLR.B    FLAG      Semaphore is cleared (unlock)
```

As these two instructions are divisible this program may run incorrectly.

Example2: Critical section access with the TAS instruction (correct)

```
TEST      TAS      FLAG      Tests the semaphore and sets it if necessary
          BMI      TEST
CRITICAL  ......             Critical section
          ......
END       CLR.B    FLAG      Semaphore is cleared (unlock)
```