5 Direct Memory Access (DMA)

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

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

DMA can also be used for

- intra-chip data transfer in multi-core processors
- "memory to memory" copying or moving of data

Reminder: 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 the programmed I/O and interrupt-driven I/O techniques.

The DMA technique uses a hardware module, called the **DMA Controller (DMAC)**. The DMAC, acting like a CPU, can generate addresses and initiate memory read or write cycles.

- The CPU programs the DMAC and delegates the I/O operations to it.
- The CPU then continues with other work.
- The DMAC performs all I/O operations by taking control of the system bus. The data does not go through the CPU.

5.1 Overview of DMA:

When data transfer is necessary, the I/O interface signals the DMAC.

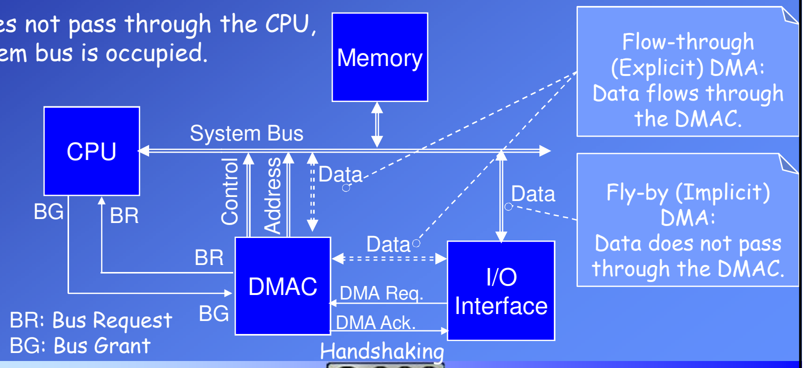
The DMAC requests the system bus from the CPU (BR).

The CPU completes the current bus cycle (not the instruction), isolates itself from the system bus, and responds to the DMAC by asserting 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 does not pass through the CPU, but the system bus is occupied.



5.2 Types of DMA Controllers:

a) Flow-through (Explicit) DMAC: The data, transferred between memory and the I/O interface passes 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 does not pass through the DMAC.

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

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

However, this technique implies that either the source or destination does not require an address because the DMAC can only put one address on the bus at any time.

So, memory-to-memory transfers (between two different addresses) are not possible in this mode.

Therefore, a fly-by DMAC can only transfer data between an I/O port and a memory address.

5.3 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 CPU determines the size of the block in the initialization process.

The DMAC may render the CPU inactive for a relatively long time.

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

b) Cycle stealing: The 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.

The 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 and operand fetch cycles, and if needed, for operand write operations.

In decode and execution cycles, the CPU can operate without accessing the memory, while the DMAC performs the data transfer.

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 rate of data transfer is slower than in burst mode.

c) Transparent mode (Hidden DMA):

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

When the DMAC determines 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.

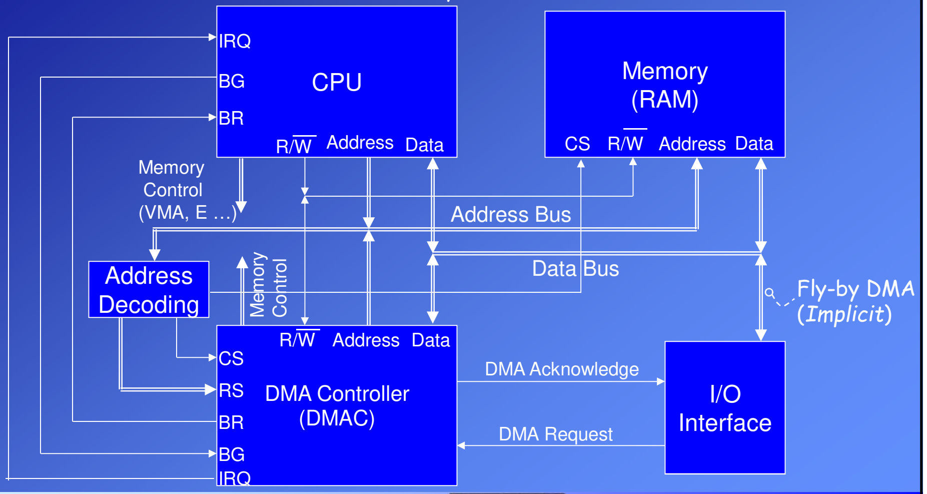
The transfer of the data block can take longer than other modes.

The disadvantage is that the hardware needs to determine when the CPU is not using the system.

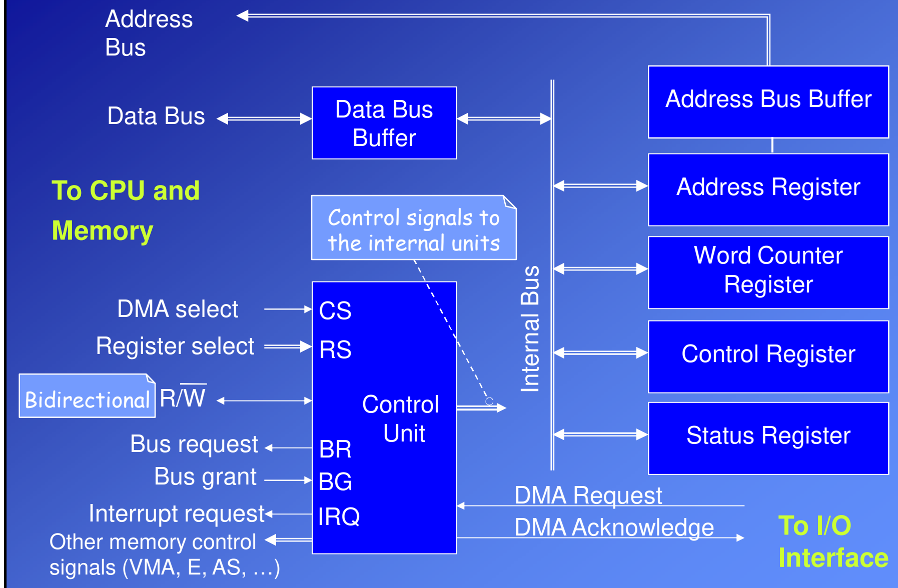
Burst mode and cycle stealing are the most commonly used transfer modes of the DMA.

5.4 CPU - DMAC Interconnection:

Before the data transfer, the DMAC is programmed (initialized) by the CPU (necessary information is written to registers of the DMAC).
In this step, the 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:



5.5 Steps of data transfer in the DMA technique

1. The CPU programs (configures) the DMA controller (DMAC).
   The I/O address, memory address, read/write, amount of data, transfer mode etc. are written to the registers of the DMAC.
   In this step, the 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. The I/O interface sends request to the DMAC by asserting DMA Request (ready to send or receive).
3. The DMAC requests the bus from the CPU by asserting the BR.
4. The CPU completes the current bus cycle (not the instruction),
   isolates itself from the system bus (goes to the 3rd state: high impedance),
   and gives control of the bus to the DMAC by asserting the BG pin.

   The CPU **cannot mask** (disable) a request from the DMAC (unlike an interrupt request).

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

5. Now, the new bus master is the DMAC. It is the responsibility of the 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.

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

   a) Fly-by (implicit) DMA:

   The DMAC sends a DMA Acknowledge to the I/O interface.

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

   b) Flow-through (explicit) DMA:

   The DMAC sends a DMA Acknowledge to the I/O interface.

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

   or

   The DMAC reads the data from memory, sends a DMA Acknowledge to the I/O interface, and writes the data to the I/O interface.

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

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

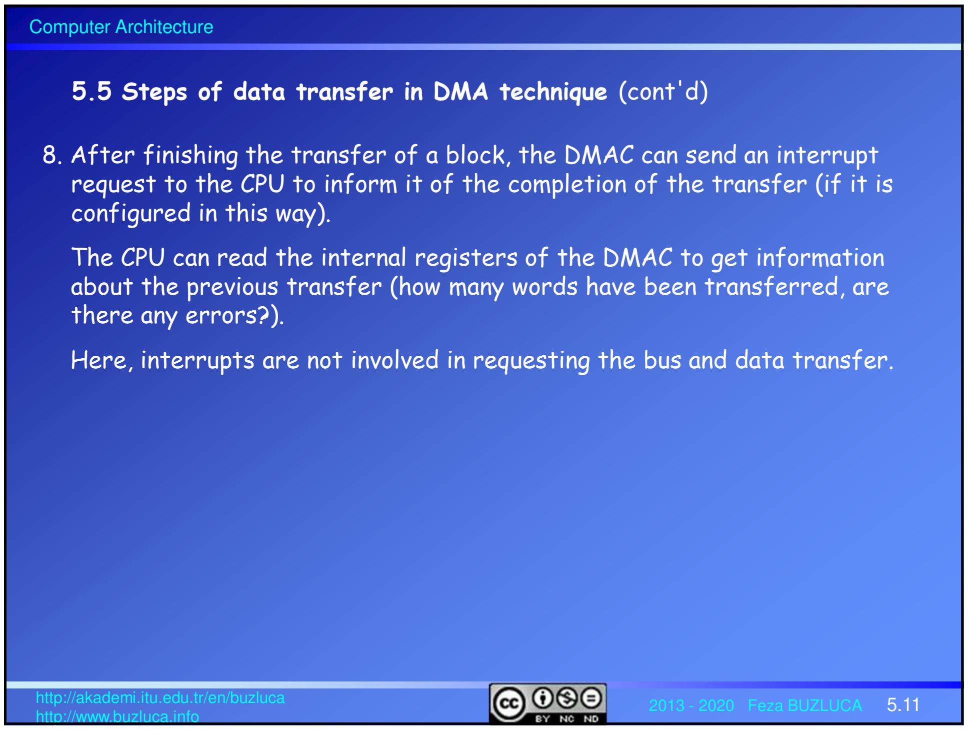
   Burst mode: The DMAC maintains BR (active) until the whole block is completed.

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

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

7. If there are no a new requests from the I/O interface (DMA Request is not active) or the entire block has been transferred (the word counter of the DMAC is zero), the DMAC isolates itself from the system bus, i.e. its control lines go to the 3rd state (high impedance).

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

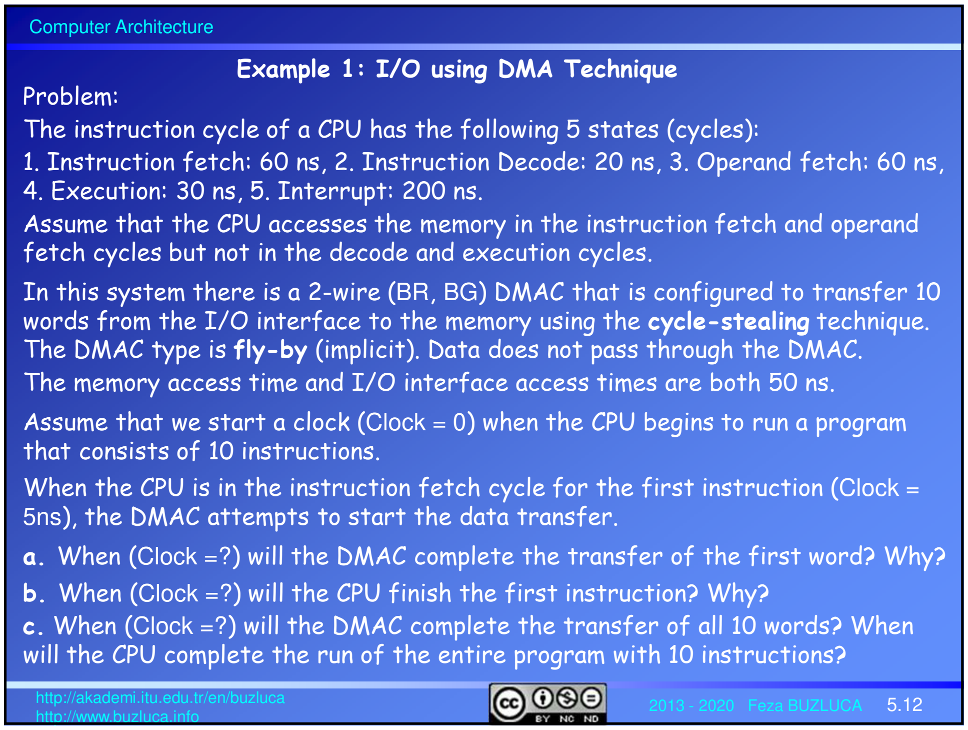


5.5 Steps of data transfer in DMA technique (cont'd)

8. After finishing the transfer of a block, the DMAC can send an interrupt request to the CPU to inform it of the completion of the transfer (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:

The instruction cycle of a CPU has the following 5 states (cycles):

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 10 words from the I/O interface to the memory using the **cycle-stealing** technique.
The DMAC type is **fly-by** (implicit). Data does not pass through the DMAC.
The 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 a program that consists of 10 instructions.

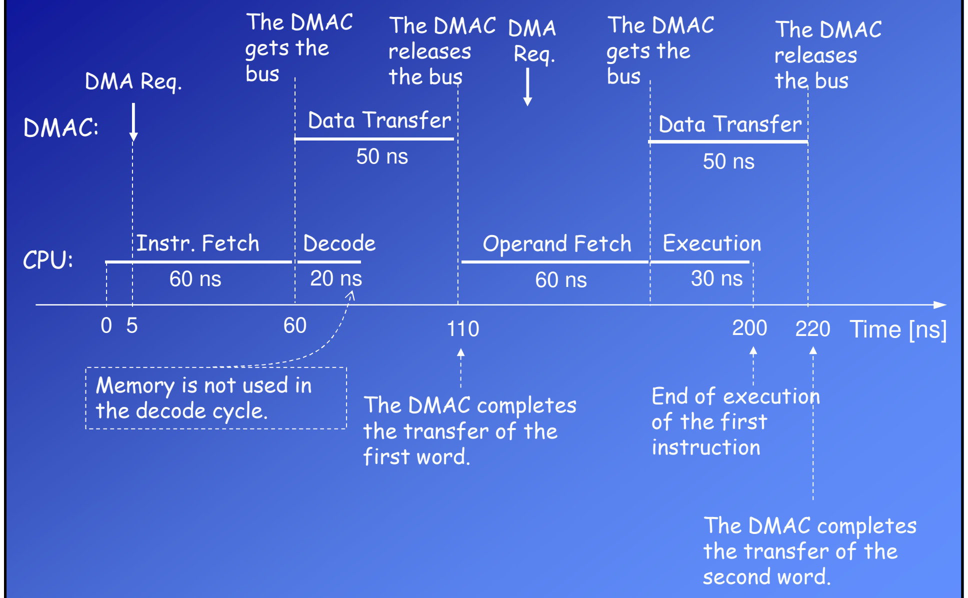
When the CPU is in the instruction fetch cycle for the first instruction (Clock = 5ns), the DMAC attempts to start the data transfer.

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:


DMA Req.
The DMAC gets the bus
The DMAC releases the bus
DMA Req.
The DMAC gets the bus
The DMAC releases the bus
DMAC:
Data Transfer
50 ns
Data Transfer
50 ns
CPU:
Instr. Fetch
60 ns
Decode
20 ns
Operand Fetch
60 ns
Execution
30 ns
0
5
60
110
200
220
Time [ns]
Memory is not used in the decode cycle.
The DMAC completes the transfer of the first word.
End of execution of the first instruction
The DMAC completes the transfer of the second word.

Solution (cont'd):

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

Since the DMAC type 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 the CPU. After the fetching the operand, 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.
The duration of an instruction cycle is 60+20+60+30= 170 ns.

Clock = 1100 + 5*170 = 1950ns

Compare these times to the interrupt-driven I/O example on slide 4.17.

Example 2: DMA and Interrupt

Problem:

The 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.

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

The memory access time and I/O interface access time 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 DMAC type is **fly-by** (implicit). Data does not pass through the 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.

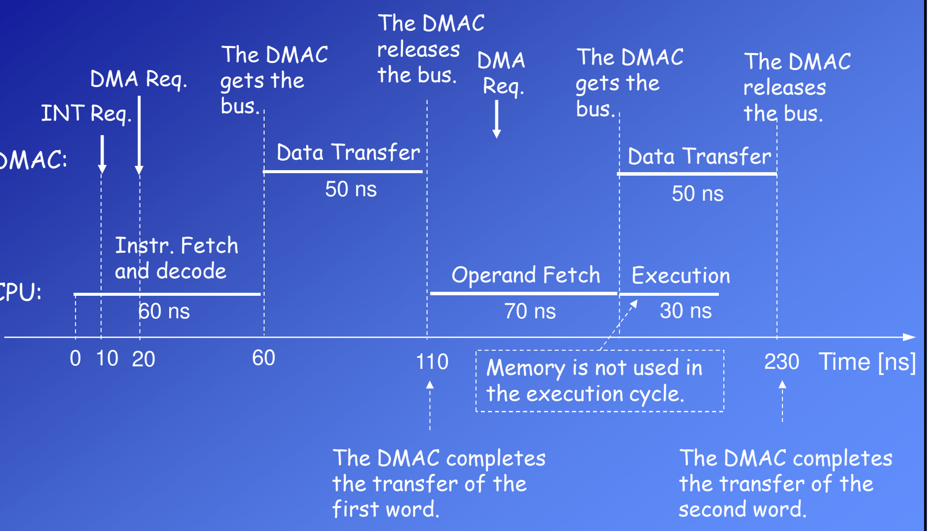
At Clock = 20ns, the DMAC attempts to start the data transfer from the I/O interface to memory. We assume that the I/O interface is always ready to transfer.

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?

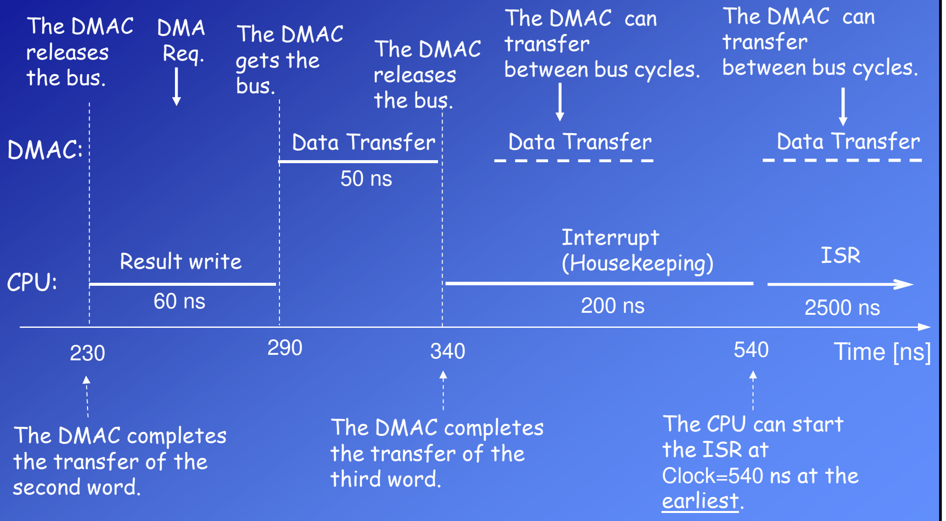
c. What happens if the DMAC has still words to transfer while the ISR is running?

Solution:



The DMAC releases the bus.
DMA Req.
The DMAC gets the bus.
DMA Req.
The DMAC gets the bus.
The DMAC releases the bus.
INT Req.
DMAC:
Data Transfer
50 ns
Data Transfer
50 ns
Instr. Fetch and decode
CPU:
60 ns
Operand Fetch
70 ns
Execution
30 ns
0
10
20
60
110
230
Time [ns]
Memory is not used in the execution cycle.
The DMAC completes the transfer of the first word.
The DMAC completes the transfer of the second word.

Solution (cont'd):



The DMAC releases the bus.
DMA Req.
The DMAC gets the bus.
The DMAC releases the bus.
The DMAC can transfer between bus cycles.
The DMAC can transfer between bus cycles.
DMAC:
Data Transfer
50 ns
Data Transfer
Data Transfer
CPU:
Result write
60 ns
Interrupt (Housekeeping)
200 ns
ISR
2500 ns
230
290
340
540
Time [ns]
The DMAC completes the transfer of the second word.
The DMAC completes the transfer of the third word.
The CPU can start the ISR at Clock=540 ns at the earliest.

Solution (cont'd):

Reminder: 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 onto 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.

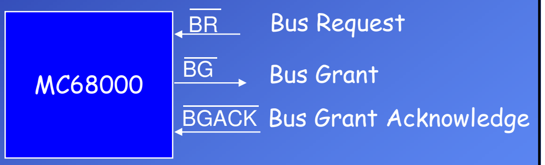
5.6 3-Wire DMA

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

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

Example MC68000:

The MC68000 uses 3 control lines for bus arbitrations.


MC68000
$\overline{BR}$ Bus Request
$\overline{BG}$ Bus Grant
$\overline{BGACK}$ Bus Grant Acknowledge

The steps of the interaction between the DMAC and the 68000:

1. The DMAC asserts a bus mastership request by asserting BR' (active low).
2. The 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 let go of the bus at the end of the current bus cycle, and the next bus master can be determined.

   There may be multiple 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 is more than one DMAC in the system, the bus arbiter circuitry determines the next bus muster.

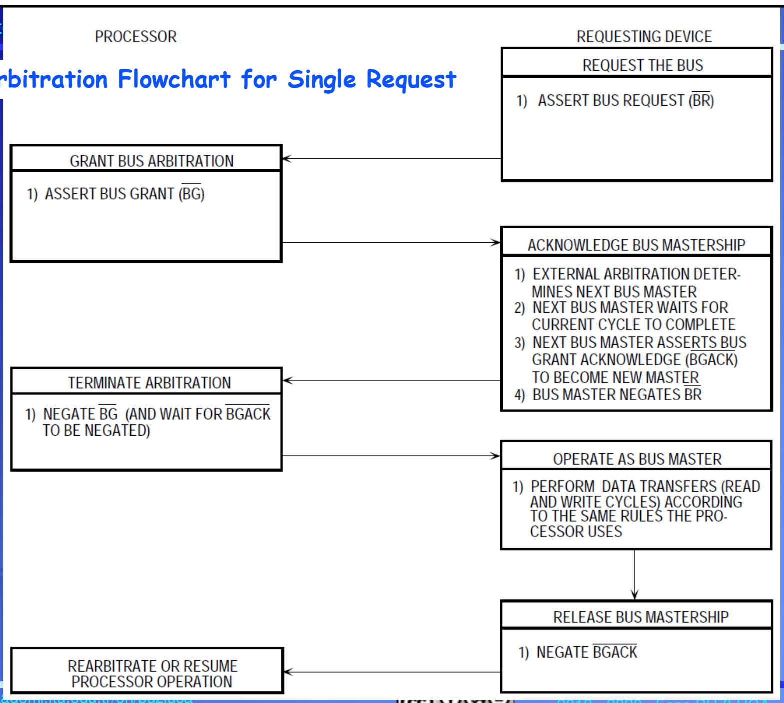
   Upon receiving BG, the requesting device waits until the current bus cycle is completed (AS, DTACK, and BGACK must be negated).

   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 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 and maintains BGACK during the entire bus cycle (or cycles) for which it is bus master.

   The BR of this DMAC is negated after BGACK is 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.

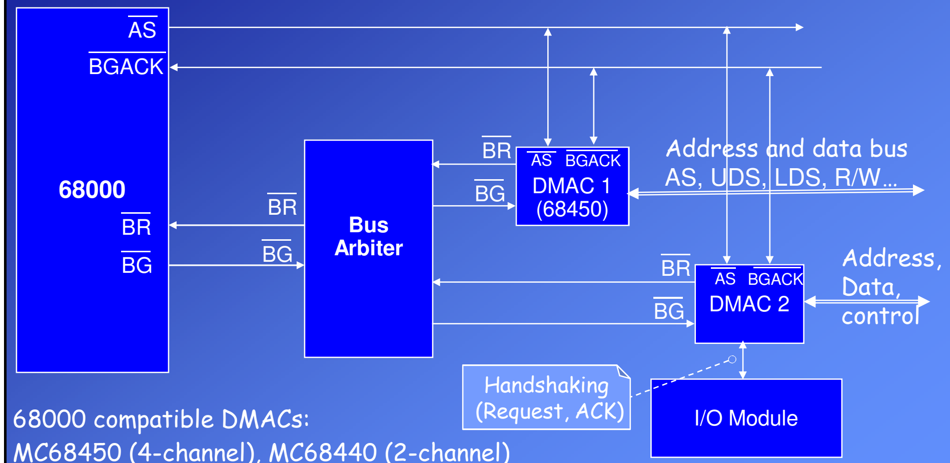
Bus Arbitration Flowchart for Single Request


PROCESSOR
REQUESTING DEVICE
REQUEST THE BUS
1) ASSERT BUS REQUEST ($\overline{BR}$)
GRANT BUS ARBITRATION
1) ASSERT BUS GRANT ($\overline{BG}$)
ACKNOWLEDGE BUS MASTERSHIP
1) EXTERNAL ARBITRATION DETERMINES NEXT BUS MASTER
2) NEXT BUS MASTER WAITS FOR CURRENT CYCLE TO COMPLETE
3) NEXT BUS MASTER ASSERTS BUS GRANT ACKNOWLEDGE ($\overline{BGACK}$) TO BECOME NEW MASTER
4) BUS MASTER NEGATES $\overline{BR}$
TERMINATE ARBITRATION
1) NEGATE $\overline{BG}$ (AND WAIT FOR $\overline{BGACK}$ TO BE NEGATED)
OPERATE AS BUS MASTER
1) PERFORM DATA TRANSFERS (READ AND WRITE CYCLES) ACCORDING TO THE SAME RULES THE PROCESSOR USES
RELEASE BUS MASTERSHIP
1) NEGATE $\overline{BGACK}$
REARBITRATE OR RESUME PROCESSOR OPERATION

Bus Arbiter

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

In systems consisting of a processor and multiple DMACs, a bus arbiter is needed to determine the next bus master if multiple devices issue requests at the same time.



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

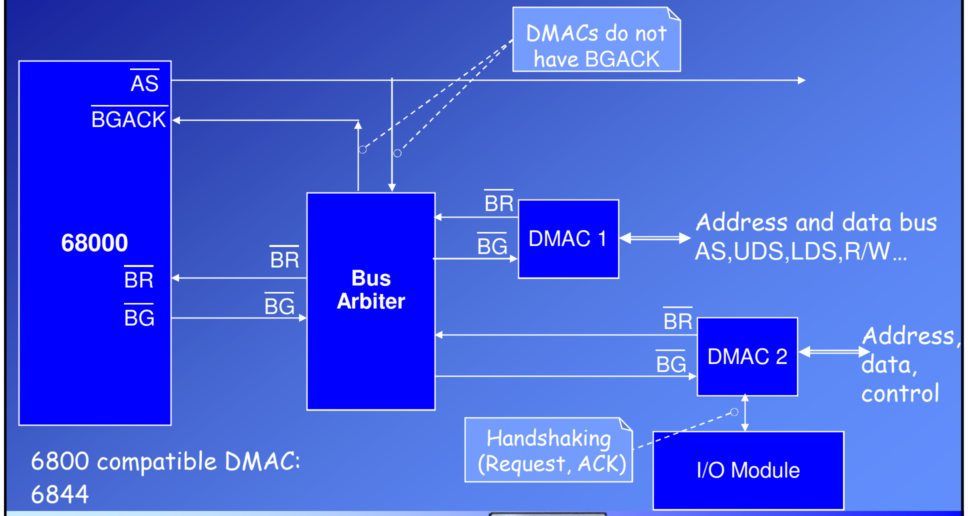
Function of the system:

If the DMAC uses the 3-wire scheme (with BGACK) (e.g., MC68450)

1. The DMAC issues the request to the bus arbiter via BR.
2. The bus arbiter sends the request to the 68000 (BR).
3. The bus arbiter receives the grant signal (BG) from the 68000 and sends this signal to the requesting DMAC with the highest priority.
4. After receiving the bus grant BG, the DMAC monitors the AS and BGACK signals to determine when it may assume mastership of the bus.
   AS and BGACK must be negated to indicate that the previous cycle is complete and the previous bus master has released the bus.
5. When this condition is met, the DMAC asserts BGACK to inform the processor and other DMACs that it has taken control of the bus.
   As the new bus master, the DMAC deasserts its BR output to allow the external arbiter to begin arbitration for the next bus master.
   It maintains BGACK until the transfer is complete.
   As the bus master, it is responsibility of this 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, the 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 BGACK to the 68000, then operations involving the BGACK signal must be performed by the bus arbiter.



6800 compatible DMAC: 6844

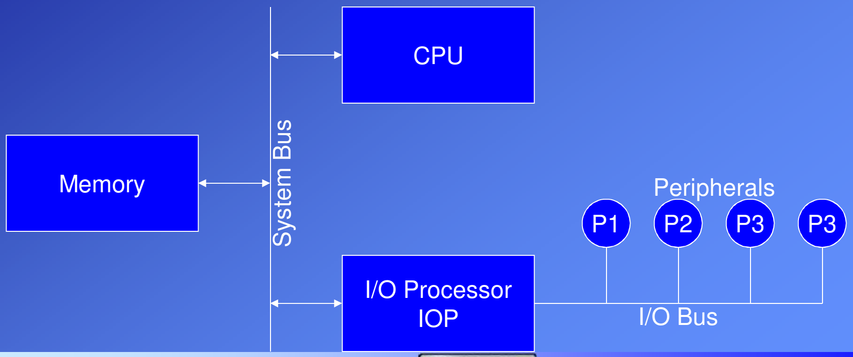
Function of the system:

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

1. The DMAC issues the request to the bus arbiter via BR.
2. The bus arbiter sends the request to the 68000 (BR).
3. After receiving the bus grant BG from the 68000, the <u>bus arbiter</u> waits for the current bus cycle to complete by monitoring the AS line.
   It also waits for the previous bus master DMAC (if any) to release the bus (BR).
   In this system, the bus arbiter itself supplies the BGACK signal; therefore, it does not need to monitor the BGACK line.
4. The bus arbiter sends the bus grant signal (BG) to the requesting DMAC with the highest priority.
5. The 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 maintains the BGACK signal.
7. When the DMAC completes its transfer, it negates its request (BR) output.
   If there is no other pending request from another DMAC, the bus arbiter negates the BGACK signal so that the 68000 can use the bus again.

5.7 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).
IOPs can also perform format conversion, encryption/decryption, error correction.



5.8 Indivisible Bus Cycle

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

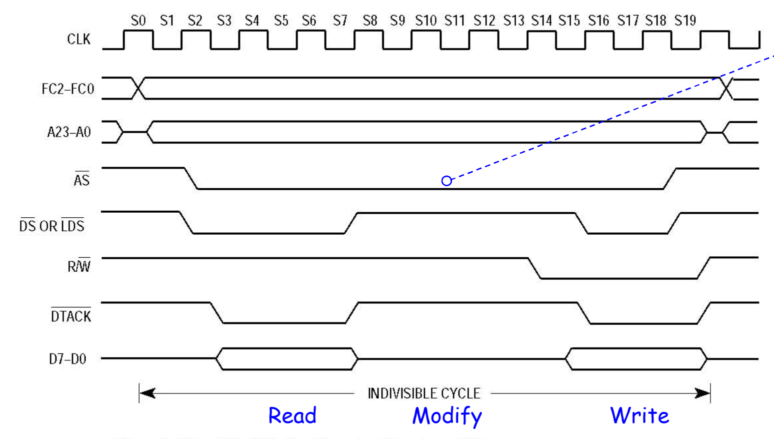
In a <u>single instruction</u> and in a <u>single bus cycle</u>

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 two ways:

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

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


S0 S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11 S12 S13 S14 S15 S16 S17 S18 S19
CLK
FC2–FC0
A23–A0
$\overline{AS}$
$\overline{DS}$ OR $\overline{LDS}$
R/$\overline{W}$
$\overline{DTACK}$
D7–D0
INDIVISIBLE CYCLE
Read
Modify
Write

Read-Modify-Write Cycle Timing Diagram

AS remains asserted throughout the entire cycle. DMACs or other processors cannot interrupt 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 using 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)
```