UNIVERSITÀ DEGLI STUDI DI TRIESTE

dipartimento di ingegneria e architettura

I/O subsystem management

A.Carini – Microcontrollers

I/O subsystem

- The “I/O subsystem” is the set of circuits, external to the CPU but implemented on chip, that transfer signals from the μC or DSP to the external world and vice versa. They are also called *peripherals* or *I/O peripherals.*
- Most common peripherals:
    - Analog to digital converters (ADC) and digital to analog converters (DAC)
    - Timers and counters, PWM modulators,
    - Communication modules (serial ports, CAN bus, I2C, SPI, …)
    - Interface circuits for encoders.
    - Display interfaces,
    - Peripherals for external memory modules management.
- The I/O subsystem can be managed with different approaches:
    - The most common solution is based on *interrupts,* but we will see also *polling* (or program control) and *direct memory access controller (DMAC).*

I/O subsystem

- The different peripherals are connected to the bus with an *interface circuit*.
- The interface circuit provides the logical and (if needed) electrical adaptation between peripherals and CPU.
- The interface must have some registers for the peripheral configuration (*CREG*), for storing the exchanged data (*DREG*), for reading the peripheral status (*SREG*).
- CREG and SREG often have a few bits and form a unique register.
- The size of registers *CREG, DREG, SREG* could be different from that of the bus.
- If the size of *DREG* is larger than the bus, it can be placed in two or more memory locations.
- The CPU and peripheral interaction can be performed in two ways:
  - Isolated input/output;
  - Memory mapped input/output.

I/O subsystem

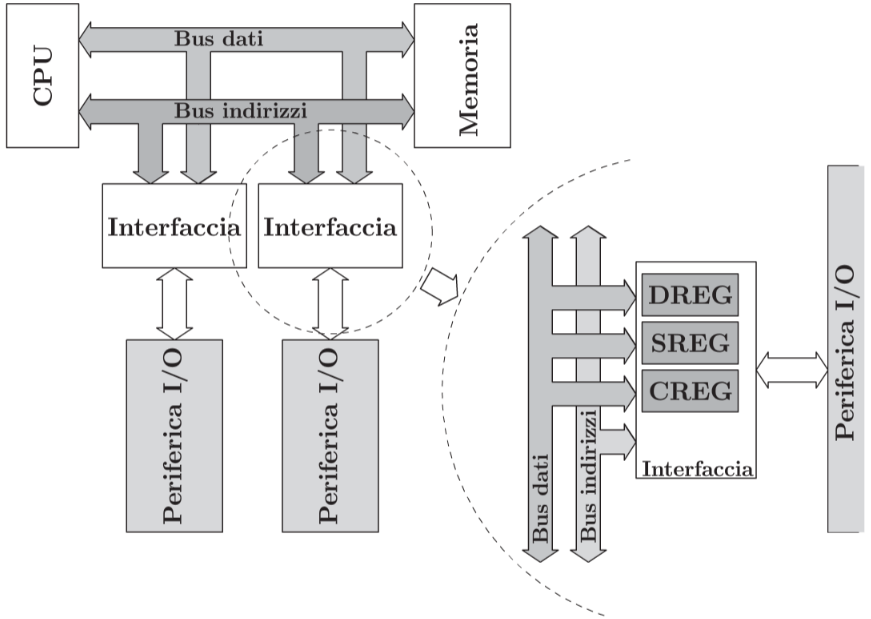


Figura 8.1: Organizzazione di un sistema di I/O. Le diverse periferiche sono connesse al bus attraverso un opportuno circuito di interfaccia.

Isolated I/O

- The isolated I/O organization exploits instructions specifically dedicated to the peripheral management.
- The set of instructions is composed by two classes: the CPU instructions and the peripherals instructions.
- The execution of a peripheral instruction operates on the bus similarly to the memory read and write instructions. But, there are dedicated control lines that select the desired peripherals.
- This I/O subsystem organization has been practically abandoned.

Memory mapped I/O

- In memory mapped I/O the registers of peripherals are treated directly as normal memory locations, without dedicated instructions or control lines.
- The technique is nowadays used in all modern µCs, DSPs, DSCs.
- Its implementation requires the peripheral to access and decode at least some of the address bus lines.
- In most recent µCs, DSPs, DSCs, a *bus* exclusively *dedicated* to the peripherals is available. This bus does not interfere with the memory access.

I/O subsystem management strategies

- The peripherals operate asynchronously and with a lower speed than CPU.
- There is the problem of synchronizing the data exchange from processor and external units.
- There are three fundamental approaches:
  - *Polling* or *program controlled I/O;*
  - *Interrupts;*
  - The use of a I/O dedicated processor, or *DMAC*.

Program controller I/O

- In program controlled I/O, the user code is designed to *periodically* poll the status register of the peripherals of interest.
- Inside an infinite cycle, the register is read and compared with a predefined configuration.
- When there is a match, the CPU jumps to a subroutine that perform the desired operation.
- Once completed the subroutine, the control return to the main program that repeats the polling.
- The approach is very inefficient: the CPU remains in a wait state for long times, without the possibility of doing other jobs.

Interrupts

- Allow the CPU to perform other jobs while the peripherals are active.
- When a peripheral needs a CPU intervention (for example, because it has produced a data), it sends the CPU a request by asserting a dedicated control line.
- The signal sent to the CPU is called *interrupt request* or *IRQ:* after receiving it, the CPU stops the execution of its program and manage the request.
- Interrupts are a particular case of *exception.* The term indicates all operation conditions of a processor that do not correspond to the normal code execution:
  - *Reset*, when a processor start execution
  - *Fault:*
    - *Hard:* processor is in an undetermined state
    - *Memory:* the code has tried to access some protected are of memory
    - *Program:* the code has performed instructions with undefined result
  - *System clock zero:* watchdog has reached zero.

Interrupts

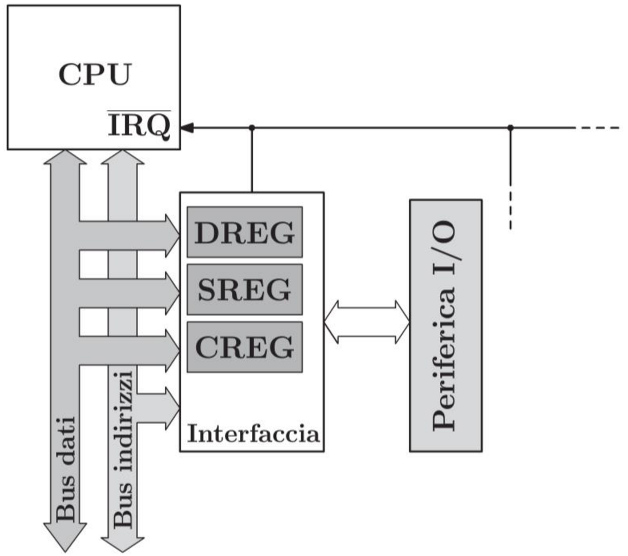


Figura 8.2: Un sistema di interruzioni richiede la presenza di una linea di controllo specifica nella CPU, che possa essere asserita dalle periferiche per richiedere l'interruzione delle attività in corso.

Interrupts

- At an interrupt request, the CPU must
    1. Complete the current instruction,
    2. Save context
    3. Activate the interrupt service routine,
    4. Recover context
    5. Continue program execution.
- Context saving requires storing at least the program counter and status register
- Some processor (e.g., ARM7) operates on an alternative set of registers.
- Others require the programmer to store in a stack the contents of the registers used by the service routine and to restore them at the routine end.

Interrupts

- Since there can be many peripherals of a µC or DSP, in the interrupt management we have to face some problems:
  1. Determine the peripheral that has requested the interrupt,
  2. Manage simultaneous IRQ by different peripherals
  3. Manage interrupts inside an interrupt service routine (*nested interrupts*)
- These problems have been solved in different ways, with:
  - Interrupt status register
  - Vectored interrupts
  - Daisy chain

Interrupt status register

- The first solution of early devices exploits one or more interrupt status register (ISR), where each bit is dedicated to a different peripheral.
- At the IRQ, the corresponding status bit is asserted in the register.
- The interrupt service routine must check one by one the bits to determine which peripheral to service.
- The solution implicitly implements a *priority* mechanism, scanning the bits in the order decided by the programmer.
- It is also possible to service nested interrupts.
- The method is anyway inefficient, since the ISR scanning can require a significant time.

Vectored interrupts

- In this case the processor has different interrupt lines (IRQ1, IRQ2, …), each dedicated to a different peripherals (of small group of peripherals) and can automatically activate the appropriate service routine.
- When the control circuits receive an interrupt request, it identifies it using an encoder with priorities.
- Immediately, it saves the program counter and updates it with the address location reserved to the interrupt source in the interrupt vector.
- The interrupt vector is a portion of the instruction memory where are placed the subroutine calls to the different interrupts.
- In this organization, both the identification of the calling peripheral and the context saving are performed automatically by the control circuit.
- It is still possible to manage nested interrupts.

Vectored interrupts

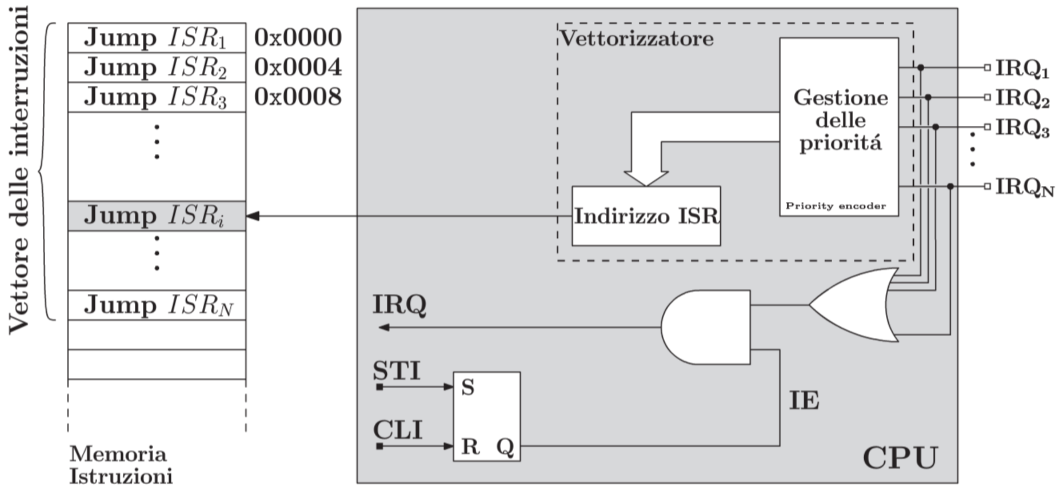


Figura 8.3: Organizzazione (semplificata) di un sistema di interruzioni vettorizzato. Si vede il circuito che abilita (quando è asserito il bit di stato STI, o *set interrupts*) e disabilita (quando viene asserito il bit di stato CLI o *clear interrupts*) la linea IRQ, attraverso il segnale IE, o *interrupt enable*. Inoltre, è indicato il circuito, che consiste in un encoder con priorità, che risponde alle linee di interruzione in ingresso. Il circuito carica automaticamente nel program counter l'indirizzo di una locazione specifica del vettore delle interruzioni.

Daisy chain

- All peripherals are connected in series to the same control line.
- This solution allows to minimize the CPU access lines and to serve a large number of peripherals.
- When the CPU receives an IRQ, the CPU sends an *acknowledgement* along the daisy chain, which propagate from peripheral to peripheral till the interrupt source.
- The peripheral that asserted the IRQ, stops the propagation of the *acknowledgement* and reply by writing on the data bus his address.
- Similarly to vectored interrupt, the program counter is saved and overwritten with the address of the interrupt service routine.
- It introduces implicitly a priority system.
- Nested interrupts are also possible.

Daisy chain

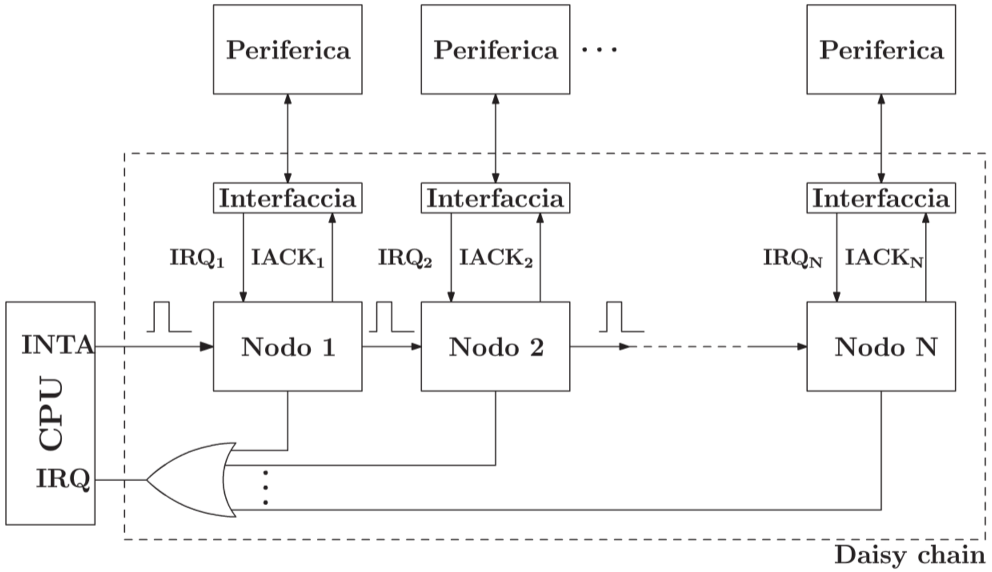


Figura 8.4: Organizzazione di un sistema di interruzioni con daisy chain.

Daisy chain

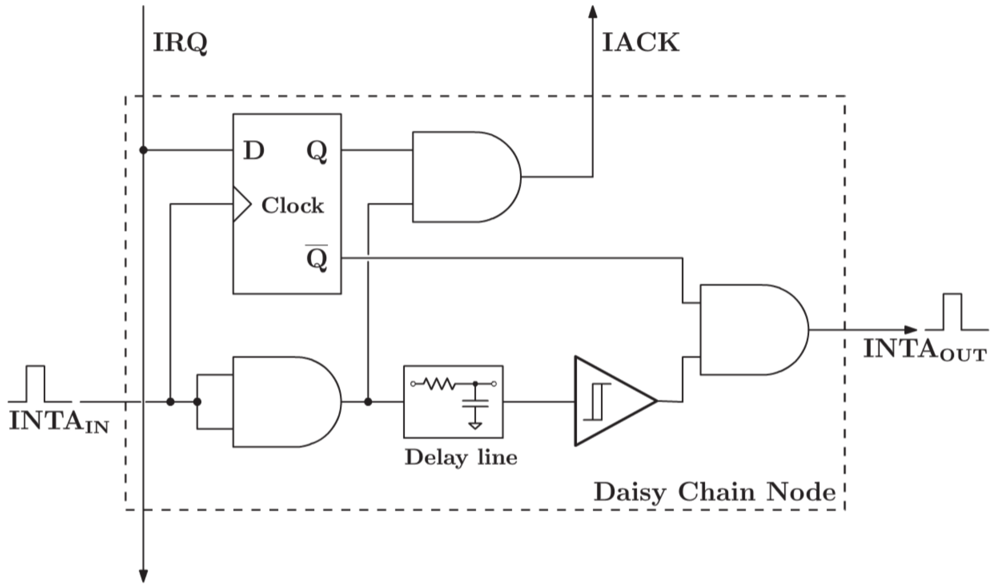


Figura 8.5: Schema elettrico semplificato di un nodo della DCL.

Interrupt masking

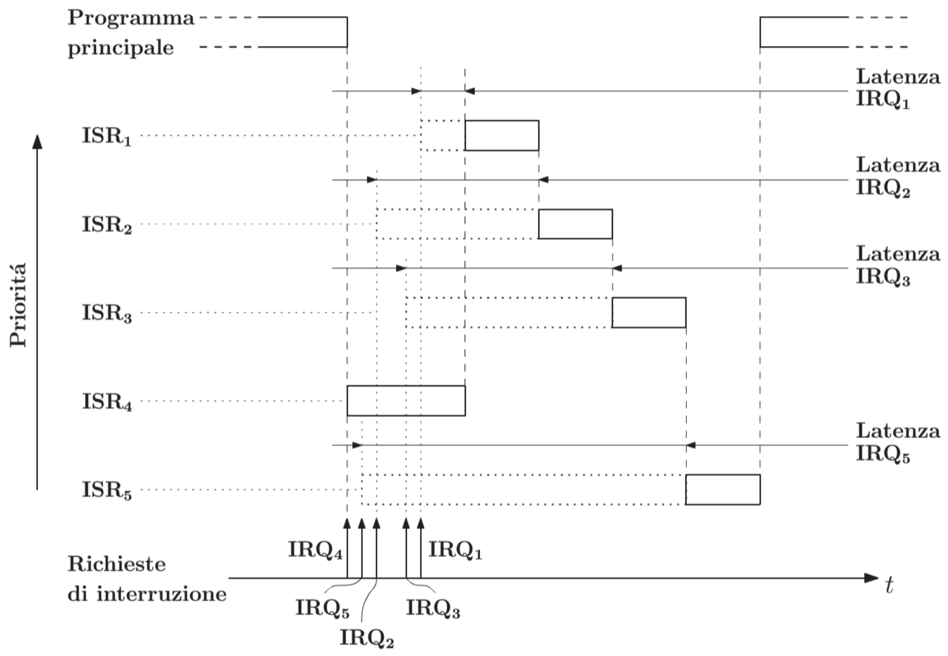
- It is possible to avoid the CPU being interrupted by peripherals by *masking* the interrupt sources.
- This is achieved keeping low the Interrupt Enable control signal, always present in µCs, DSPs, DSCs.
- Sometimes, this is obtained by using a certain user code in specific instruction.
- In other cases, by modifying one or more bits of the processor status register.
- Often it is possible to inhibit the single source.
- Anyway all processor have a *non-maskable interrupt* NMI for handling emergency conditions.

Parameters of an interrupt system

1. Maximum number of sources;
2. Priority management way;
3. Speed of context saving;
4. Interrupts masking.

- In real-time control the speed of context saving is often a critical parameter.
- Equally important is the interrupt masking in critical code regions.
- Most recent µCs, DSPs, DSCs have a vectored interrupt system. The interrupt service subroutine is called automatically, together with context saving.
- The number of interrupt sources if often very high (e.g. > 10) and is often possible to prioritize the interrupts.

Parameters of an interrupt system



(a)

Parameters of an interrupt system

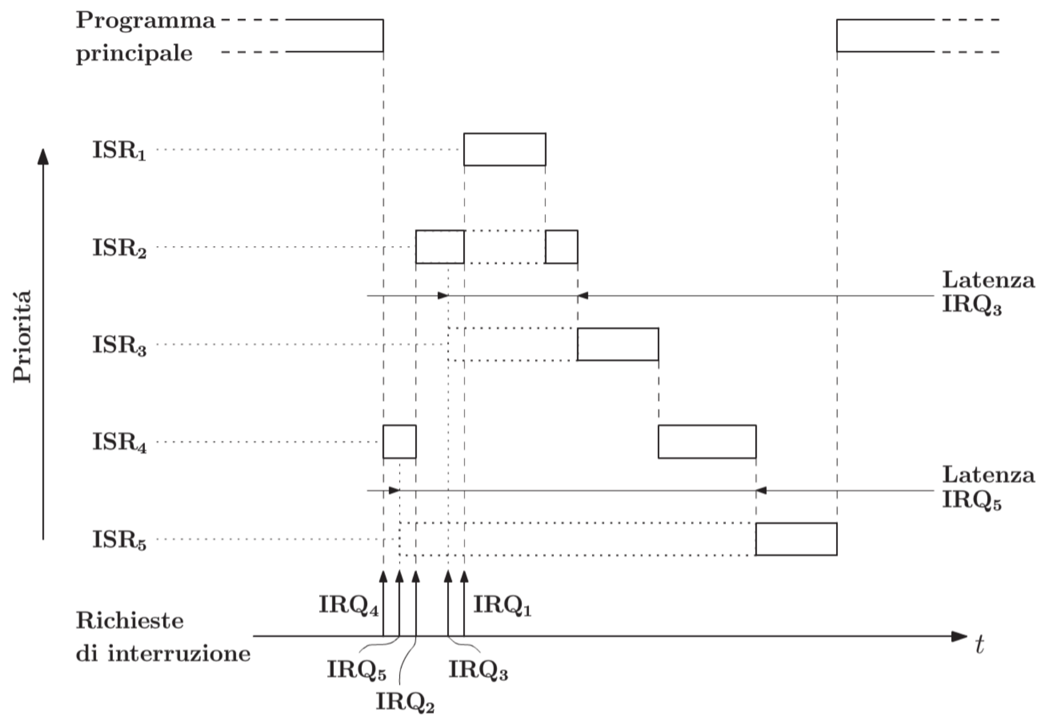


Figura 8.6: Gestione di una sequenza di richieste di interruzione: (a) in un sistema che *non* permette il nesting e (b) in uno che, invece, lo permette. In questo caso, le interruzioni di priorità più alta sono servite subito, quelle di priorità più bassa sono lasciate pendenti.

Parameters of an interrupt system

- Nesting can be advantageous.
- We can obtain similar performance without nesting only in case we have few ISRs of short execution time.
- Interrupt managing without nesting is the most frequent in µCs and DSPs.
- ARM Cortex M3 allows vectored interrupts *with nesting* (with up to 240 interrupts). Context saving in a stack is automatic.
- ARM7 manage at most two interrupt sources and does not allow nesting. Also in this case context saving is automatic.

Analysis of latency times in interrupts

- In many sensitive applications it is necessary to forecast the maximum delay before an interrupt request is serviced, i.e., its *latency time.*
- The latency time always have an intrinsic component T_{LI}, due to the necessity to save the context. Other delays can add
  1. When a particularly complex instruction is executed (in CISC arch.). Since this must be completed before servicing the ISR, we have a delay T_{IST}
  2. When interrupt requests are disabled to protect critical code regions, T_{RC}
- Thus, the total latency time is the sum of tree contributes:

$$T_{LI} + T_{RC} + T_{IST}$$

- *Critical regions* are code segments that must be executed strictly in sequence, e.g. for guaranteeing a precise temporization.

Analysis of latency times in interrupts

- In systems with several sources, interrupt requests can interfere.
- The lower priority requests will have larger latencies.
- In systems *with nesting*, the worst case latency of an interrupt with priority *n* is

$$T_{L_n} = T_{LI} + \sum_{i=1}^{n-1} T_{d_i},$$

- Where T_{d_i} is the length of ISR i: $T_{d_i} = T_{ex_i} + T_{LI} + T_{IST}$

- In systems *without nesting,*

$$T_{L_n} = T_{LI} + \underset{j>n}{Max}\left(T_{d_j}\right) + \sum_{i=1}^{n-1} T_{d_i}$$

Interrupt density

- When a µC or DSP has to manage a certain number of interrupt sources, we must guarantee that *all* can be serviced.
- A *necessary* (but non sufficient) condition for sustainability is

$$\sum_{i=1}^{N} \frac{T_{d_i}}{T_{p_i}} = \sum_{i=1}^{N} \frac{T_{ex_i} + T_{LI} + T_{IST}}{T_{p_i}} < 1$$

- Where T_{p_i} is the repetition period of *i*-th ISR.
- The condition is not sufficient, since there could be critical regions and since the combined requests could prevent to service the *i*-th ISR in T_{p_i}.
- To include also these conditions, for any interrupt we must have:

$$T_{d_i} + Max\left(T_{RC}, \underset{j>i}{Max}\left(T_{d_j}\right)\right) + \sum_{k=1}^{i-1} N_k \cdot T_{d_k} < T_{p_i}. \qquad N_k = \left\lceil \frac{T_{p_i}}{T_{p_k}} \right\rceil$$

Example

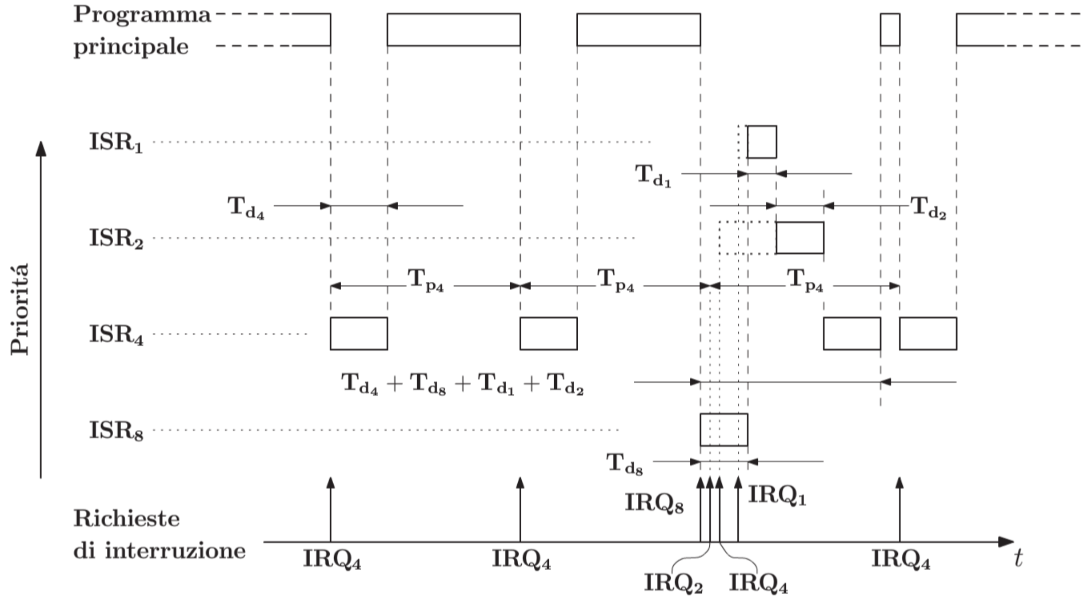


Figura 8.7: Comportamento di un sistema di interruzioni senza nesting in condizioni di elevata densità di richieste. Si considerano nulli o trascurabili T_{RC}, T_{LI} e T_{IST}.

Direct memory access (DMA)

- Is the most efficient approach for interfacing a CPU and some I/O peripherals.
- It consists of a additional control circuit, called DMA controller (DMAC) that, once programmed, can manage peripherals *without involving the CPU.*
- The DMAC exploits a portion of data memory, specified by the programmer in the configuration, for transferring in a bidirectional way the data produced or expected by the peripherals, and this at the maximum velocity without any CPU activity.
- When the memory block has been updated with new data, or all data has been transferred to the peripheral, a single interrupt request is generated.
- This is an efficient and expensive solution, but is now available in many µCs, DSPs, and DSCs.

DMAC organization

- The DMAC acts replacing the CPU in the address and data bus control.
- To control the bus, DMAC uses some gates that act as switches. When they are active, they isolate the CPU from bus for all the data transfer time.
- The operation can be performed a little at a time, with a *cycle stealing* approach, or from the beginning till the end of a block of data, with an approach called *burst.*
- The protocol used is the following:
    1. The peripheral requests to DMAC permission to transfer (Transfer Request).
    2. DMAC request CPU permission to use bus (DMA Request).
    3. CPU grant use of bus (DMA Grant): switches isolate CPU and connects DMAC and peripheral to memory.
    4. DMAC set the address on bus an authorize the peripheral to read or write data (Transfer Grant).

DMAC organization

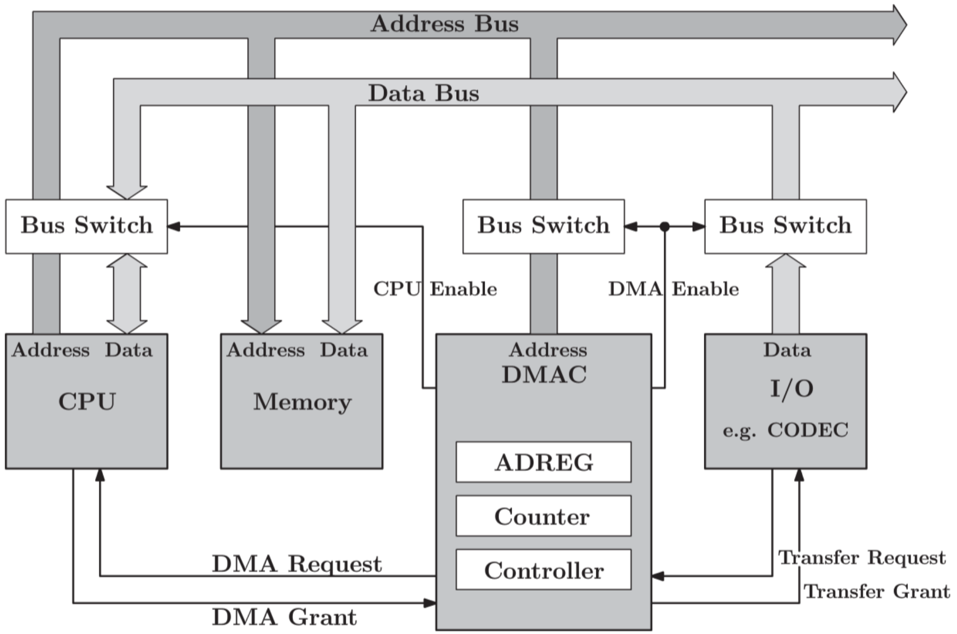


Figura 8.8: Schema di principio di una organizzazione con DMAC. La periferica servita è un CODEC, che produce un flusso continuo di dati ad elevata velocità. La figura mostra anche alcuni registri interni del DMAC, necessari per specificare l'indirizzo del blocco di memoria su cui deve operare, la dimensione del blocco stesso e i parametri di configurazione del trasferimento.

See:

- Simone Buso, «Introduzione alle applicazioni industriali di Microcontrollori e DSP» Società editrice Esculapio, 2018
    - Chapter 8