Appunti IoT Systems

Capitolo 10

Performance dei sistemi di I/O

Simone Coco

Performance dei sistemi di Input/Output

Abbiamo visto quali sono le principali tecniche per la gestione dei device di Input/Output, classificandole in:

- **Polling:** la CPU chiede continuamente se la risorsa è disponibile o meno. Questa tecnica occupa dei tempi di CPU elevati e quindi non è la migliore.
  - × Elevato CPU time poiché la CPU deve controllare il traffico e lo stato di ready dei device I/O
- **Interrupt:** la CPU si occupa di eseguire le ISR (interrupt Service Routine) per gestire il device I/O a seguito di un interrupt che viene inviato dai device quando sono ready e che interrompe la CPU stessa nell'esecuzione di un foreground program. Con un funzionamento gerarchico si può stabilire la priorità degli interrupts. Questa soluzione perde però ancora del CPU time utile per il trasporto dei dati attraverso un unico bus condiviso.
  - √ La CPU non richiede continuamente lo stato di ready ai device di I/O (migliore del polling)
  - × Si perde ancora del CPU Time per gestire il trasporto dei dati con unico Bus
- **DMA:** viene utilizzato un supporto hardware che "sostituisce" la CPU nelle funzioni di trasporto dei dati; utilizzando un'architettura Von Neumann si avrebbero dei problemi di conflitti tra CPU e DMA per l'accesso al Bus poiché vi è un unico bus centralizzato, tuttavia, supponendo di utilizzare un'**architettura Harvard,** quindi utilizzando due bus differenti (uno per il trasporto dei dati utilizzato dal DMA e l'altro per la computazione della CPU) si ottimizza al meglio la tecnica DMA; l'unico caso di conflitti nasce per la gestione delle istruzioni di **load/store** (circa il 20-25%)
  - √ La CPU non richiede continuamente lo stato di ready ai device di I/O (migliore del polling)
  - √ La CPU non si occupa del trasporto dei dati (migliore dell'Interrupt)
  - × Si potrebbero avere dei conflitti per le istruzioni di load/store (20-25% delle istruzioni totali)

Studiamo adesso più approfonditamente le performance delle tecniche che abbiamo elencato.

Performance dei sistemi di I/O: Polling

Definiamo l'efficienza del trasferimento di dati utilizzando il polling:

$$\eta_{POLLING} = \frac{T_{TRANSPORT}}{T_{POLLING} + T_{TRANSPORT}}$$

Con:

$T_{TRANSPORT}$: tempo speso dalla CPU per il trasporto di un dato "pronto"

$T_{POLLING}$: tempo speso dalla CPU per controllare che il dato sia pronto

Diremo, inoltre, che:

$$T_{POLLING} = (x + 1)T_{READY}$$

Con:

T_{READY} : tempo necessario per ogni controllo di presenza del dato (controllo sullo stato di ready)

x : numero di controlli aventi esito negativo (che "bloccano" la CPU in attesa del dato)

L'efficienza della tecnica del polling si trova come rapporto tra il tempo utile per il trasporto ed il tempo totale impiegato dato dalla somma dei tempi di trasporto e dei tempi di attesa in cui la CPU attende lo stato di ready dei device I/O.

L'efficienza di polling è tanto più piccola quanto è maggiore il tempo in cui la CPU resta in attesa del dato ($T_{POLLING}$) [Tempo che possiamo chiamare "Overhead del polling"].

Riscriveremo allora la formula come segue:

$$\eta_{POLLING} = \frac{T_{TRANSPORT}}{T_{POLLING} + T_{TRANSPORT}} = \frac{T_{TRANSPORT}}{(x+1)T_{READY} + T_{TRANSPORT}}$$

Possiamo suppore che il tempo di Ready sia, effettivamente, un frazione del tempo di trasporto dato che l'informazione, prima di essere trasportata (e spendere il tempo $T_{TRANSPORT}$) deve trovarsi nello stato di ready, (impiegando il tempo T_{READY}) fintanto che essa non sia pronta per il trasporto.

Introduciamo allora un parametro che chiameremo **peso W** che indica quanto "pesi" il tempo impiegato in attesa dello stato di ready di un dato rispetto al tempo totale di trasporto:

$$W_{POLLING} = \frac{T_{READY}}{T_{TRANSPORT}} \qquad \text{Da cui otteniamo:} \qquad T_{READY} = W_{POLLING}\, T_{TRANSPORT}$$

Con questa equazione possiamo scrivere tutto in termini di $T_{TRANSPORT}$ effettuando così le dovute semplificazioni, ottenendo:

$$\eta_{POLLING} = \frac{T_{TRANSPORT}}{(x+1)T_{READY} + T_{TRANSPORT}} \rightarrow \frac{T_{TRANSPORT}}{(x+1)\, W\, T_{TRANSPORT} + T_{TRANSPORT}}$$

$$\eta_{POLLING} = \frac{\cancel{T_{TRANSPORT}}}{\cancel{T_{TRANSPORT}}} \frac{1}{(x+1)\, W * 1 + 1} \rightarrow \boldsymbol{\eta_{POLLING} = \frac{1}{(x+1)\, W + 1}}$$

Dalla formula così scritta possiamo osservare che, sostanzialmente, l'efficienza è data da due parametri:

- *x* : indica il numero di tentativi spesi per trovare il dato nello stato di ready (x indica i tentativi andati a vuoti e 1 il tentativo andato a buon fine)
- W: tempo sprecato per effettuare il polling

L'efficienza è data dal tempo impiegato ad effettuare il polling nella ricerca del dato per tutte le volte che viene effettuata tale ricerca.

Generalizzando, supponendo di avere **S sorgenti che effettuano N trasferimenti di dati,** diremo che l'efficienza è data da:

$$\eta_{POLLING} = \frac{\sum_{i=1}^{N} T_{TRANSPORT\ i}}{\sum_{i=1}^{N} T_{POLLING\ i} + \sum_{i=1}^{N} T_{TRANSPORT\ i}}$$

Con:

$$T_{POLLING\ i} = (x_i + 1)T_{READY}$$

T_{READY} : tempo (costante) necessario per ogni controllo di presenza del dato
x_i : numero di controlli aventi esito negativo per la trasmissione i-esima

Quindi:

$$\eta_{POLLING} = \frac{\sum_{i=1}^{N} T_{TRANSPORT\ i}}{\sum_{i=1}^{N} (x_i + 1)T_{READY_i} + \sum_{i=1}^{N} T_{TRANSPORT\ i}}$$

Definendo nuovamente il **peso W_i dell'i-esimo dato:**

$$W = \frac{T_{READY}}{T_{TRANSPORT\ i}}$$

Possiamo definire la generica efficienza per S sorgenti come segue:

$$\eta_{POLLING} = \frac{\sum_{i=1}^{N} T_{TRANSPORT\ i}}{\sum_{i=1}^{N}(x_i + 1) W\, T_{TRANSPORT_i} + \sum_{i=1}^{N} T_{TRANSPORT\ i}}$$

Supponendo che il tempo di trasporto $T_{TRANSPORT}$ sia costante per tutte le trasmissioni, possiamo riscrivere la formula come segue:

$$\eta_{POLLING} = \frac{N\, T_{TRANSPORT\ i}}{W\, T_{TRANSPORT}\, [\sum_{i=1}^{N}(x_i + 1)_i] + N\, T_{TRANSPORT\ i}}$$

Semplificando si ottiene:

$$\eta_{POLLING} = \frac{\cancel{T_{TR}}}{\cancel{T_{TR}}} \frac{N}{\sum_{i=1}^{N}(x_i + 1)_i\ W * 1\ + N} \rightarrow \boldsymbol{\eta_{POLLING} = \frac{N}{W\ \sum_{i=1}^{N}(x_i + 1)_i\ + N}}$$

$$\boxed{\boldsymbol{\eta_{POLLING} = \frac{N}{W\ \sum_{i=1}^{N}(x_i + 1)_i\ + N}}}$$

[Caso Generico]

Anche in questo caso l'efficienza dipende dai parametri x e W, con la particolarità che, avendo N trasmissioni, l'efficienza dipenderà oltre che da questi parametri anche da N.

Approssimazioni con il valor medio

Per calcolare l'efficienza con la formula appena trovata per N trasmissioni da S sorgenti dovremmo conoscere tutte gli x_i, che corrispondo al numero di tentativi di lettura/scrittura dell'i-esimo dato (i tempi che sono trascorsi nell'attesa che il dato passasse allo stato di ready) [x_1 è il numero di tentativi per il dato 1, … , x_N è il numero di tentativi per il dato N-esimo].

A volte potrebbe però essere difficoltoso conoscere tutti i numeri di tentavi per gli N dati, quindi si potrebbe voler studiare **in media** l'efficienza del polling per un dato sistema.
Si definisce allora **x_m il valor medio di check status** (numero medio di tentativi per dato), avremo:

$$x_m = \frac{\sum_{i=1}^{N} x_i}{N}$$

Sfruttando il valor medio al posto della somma dei singoli tentativi, otteniamo:

$$\eta_{POLLING} = \frac{N}{W \sum_{i=1}^{N}(x_i + 1)_i + N} = \frac{N}{WN\ x_m + WN + N}$$

Semplificando si ottiene:

$$\eta_{POLLING} = \frac{\cancel{N}}{\cancel{N}} \frac{1}{W\ (x_m + 1)} \rightarrow \boldsymbol{\eta_{POLLING} = \frac{1}{W\ (x_m + 1)}}$$

$$\boxed{\boldsymbol{\eta_{POLLING} = \frac{1}{W\ (x_m + 1)}}}$$

Un altro importante dato che possiamo ottenere è il **tempo per trasferire un byte con polling** (tempo impiegato per l'I/O con tecnica polling)**:**

$$\boldsymbol{T_{IO\_POLLING} = T_{POLLING} + T_{TRANSPORT}}$$

Quindi possiamo riscrivere la formula da cui abbiamo incominciato a trarre i nostri ragionamenti così come segue:

$$\eta_{POLLING} = \frac{T_{TRANSPORT}}{T_{POLLING} + T_{TRANSPORT}} \quad \rightarrow \quad \eta_{POLLING} = \frac{T_{TRANSPORT}}{T_{IO\_POLLING}}$$

Avendo trattato il tempo per trasferire un byte con polling possiamo anche trattare il **tempo medio per trasferire un byte con polling:**

$$T_{IO\_POLLING\ m} = \frac{T_{TRANSPORT}}{\eta_{POLLING}} \rightarrow \boldsymbol{T_{IO\_POLLING\ m} = T_{TRANSPORT}[\,W(x_m + 1) + 1]}$$

Con questa formula si calcola quanto tempo di impiega mediamente a trasferire un byte con la tecnica del polling. Conoscendo questo tempo si può anche conosce il suo reciproco: **la frequenza $f_{I/O}$ e questa mi darà un limite sulla frequenza massima del segnale che si può campionare:**

$$\boldsymbol{f_{IO\_POLLING} = \frac{1}{T_{IO\_POLLING}}}$$

Performance dei sistemi di I/O: Interrupt

L'efficienza nel caso di tecnica di gestione dell' I/O basata su interrupt si ottiene dalla formula:

$$\eta_{INTERRUPT} = \frac{T_{ISR}}{T_{IO\_INTERRUPT}}$$

Con:

$$T_{IO\_INTERRUPT} = T_{OVERHEAD} + T_{ISR}$$

L'efficienza è praticamente data dal rapporto tra il tempo che la CPU impiega ad eseguire la Interrupt Service Routine per gestire il device di I/O ed il tempo totale impiegato nella gestione dell'I/O; quest'ultimo è costituito dal tempo dell'esecuzione dell'ISR ed i tempi di overhead per stoppare/riprendere l'esecuzione del foreground program.
Ricapitolando:

$$\boldsymbol{\eta_{INTERRUPT} = \frac{T_{ISR}}{T_{OVERHEAD} + T_{ISR}}}$$

Definiamo allora nuovamente il **peso W** per quanto riguarda la tecnica di gestione con Interrupt, ottenendo:

$$\boldsymbol{W_{INTERRUPT} = \frac{T_{OVERHEAD}}{T_{ISR}}}$$

Questa quantità indica quanto tempo viene speso nell'overhead rispetto al tempo per l'ISR.

Avendo definito il peso, possiamo scrivere l'efficienza come segue:

$$\eta_{INTERRUPT} = \frac{T_{ISR}}{T_{IO\_INTERRUPT}} \rightarrow \eta_{INTERRUPT} = \frac{T_{ISR}}{WT_{ISR} + T_{ISR}}$$

Da cui si ottiene:

$$\eta_{INTERRUPT} = \frac{\cancel{T_{ISR}}}{\cancel{T_{ISR}}} \frac{1}{W + 1} \rightarrow \boldsymbol{\eta_{INTERRUPT} = \frac{1}{W + 1}}$$

$$\boldsymbol{\eta_{INTERRUPT} = \frac{1}{W + 1}}$$

L'efficienza dell'interrupt dipende solamente da quanto pesa l'overhead sull'intera trasmissione.
NB: Come si può notare, a differenza dell'efficienza calcolata per la tecnica di gestione di I/O con polling, **non vi è il termine x_m** , questo perché il metodo con interrupt la CPU non procede con la continua interrogazione (polling) ma è instaurato un metodo di avvertimento della CPU: appunto, l'interrupt che permette al device di "avvertire" la CPU quando il dato è pronto.
Dunque, a parità di W l'efficienza dell'interrupt è maggiore.

A questo punto, possiamo scrivere anche il tempo totale impiegato per la gestione con interrupt:

$$\boldsymbol{T_{IO\_INTERRUPT} = \frac{T_{ISR}}{\eta_{INTERRUPT}} = (1 + W) * T_{ISR}}$$

Supponiamo di avere un sistema di I/O gestito con interrupt. Noti i valori T_{ISR} (tempo impiegato dall' interrupt service routine) e W (peso dell'overhead sull'intera trasmissione) si può calcolare il tempo totale per il trasferimento con tecnica di gestione dell'I/O con interrupt.

Si noti, inoltre, che conoscendo il tempo $T_{IO\_INTERRUPT}$ si può entrare a conoscenza anche della frequenza f_{IO} , la **massima frequenza del segnale acquisibile**, quindi il **throughput massimo che si può smaltire con il dato sistema di I/O:**

$$\boldsymbol{f_{IO\_INTERRUPT} = \frac{1}{T_{IO\_INTERRUPT}}}$$

f_{IO} da informazioni circa il massimo numero di campioni al secondo che possono essere trasferiti (numero di byte al secondo supponendo che un campione coincida con un byte).

Domande sugli Interrupt:

1. Si utilizzi l'interrupt come tecnica per la gestione dell'I/O. Quale delle seguenti opzioni è vera?
   a. Il vettore degli interrupt serve a gestire in modo più efficiente la priorità
      Falso: La priorità ed il vettore di interrupt sono ortogonali: sono due cose distinte e disgiunte, l'una non implica l'altra e viceversa.

   b. Il vettore degli interrupt viene sempre generato dalla CPU
      Falso: Il vettore di Interrupt viene inviato dai device collegati verso la CPU ed è falso il contrario.

   c. Il vettore degli interrupt serve a identificare la ISR (Interrupt Service Routine) da gestire
      Vero: il vettore di Interrupt permette di identificare la ISR che deve essere gestita a seconda dell'Interrupt.

   d. Nessuna delle precedenti

Performance dei sistemi di I/O: DMA

Vogliamo calcolare il massimo tempo di input/output ottenibile con la tecnica DMA, così facendo possiamo dedurre la frequenza massima del segnale acquisibile e quindi il numero di dispositivi di I/O che possono essere collegati al microcontrollore.

Performance ideale per DMA

Consideriamo come **caso ideale** l'utilizzo di un'architettura Harvard nel caso di **nessun conflitto con la CPU per l'accesso in memoria** (non vi sono conflitti per il load/store), quindi si avrebbe il massimo della performance ottenibile poiché CPU e DMA possono lavorare perfettamente in parallelo**.**

La CPU, seppur non entri in contrasto con il DMA per l'accesso in memoria, perde comunque del tempo per la programmazione di quest'ultimo (il settaggio che deve essere fatto all'inizio per il corretto funzionamento del DMA), chiameremo questo tempo $\mathbf{T_{DMA\_PROGRAM}}$**.**

Oltre a questo tempo, la CPU perde il tempo per eseguire l'interrupt di end of transfer EOT: $\mathbf{T_{INT\_EOT}}$

Diremo allora:

$$T_{CPU/DMA} = T_{DMA\_PROGRAM} + T_{INT\_EOT}$$

Come si può notare, i tempi trattati non dipendono da N, ovvero dal numero di byte trasmessi, a differenza di polling ed interrupt: per questi, infatti, avendo N byte sarà necessario moltiplicare per N volte i tempi impiegati, quindi all'aumentare di byte si incrementa il tempo di I/O finale.

Questo non vale per la tecnica DMA poiché la CPU programma il DMA in modo che possa avvenire il trasferimento di N byte entro il tempo massimo di $T_{CPU/DMA}$.

Già da questa osservazione si può comprendere il vantaggio che offre la tecnica DMA.

Possiamo allora chiederci: rispetto alla tecnica con interrupt, quanto è più veloce la tecnica con DMA? Per rispondere a questa domanda introduciamo il parametro **Speedup** che indica **quanto il DMA sia più performante rispetto ad un'altra tecnica**. Esempio:

Si supponga di avere:

$$T_{DMA\_PROGRAM} = T_{INT\_EOT} = \frac{1}{2} T_{IO\_INTERRUPT} = \frac{1}{2} (1 + W) * T_{ISR}$$

Stiamo supponendo che utilizzando una tecnica DMA si ha che il tempo impiegato per la programmazione del DMA che sia uguale al tempo impiegato per eseguire l'interrupt di end of transmission (EOF).

Queste due quantità sono pari a metà del tempo (totale) impiegato dalla tecnica di gestione dell'I/O basata su interrupt; ricordando quanto detto per la tecnica con interrupt, possiamo scrivere che $T_{IO\_INTERRUPT} = (1 + W) * T_{ISR}$, che abbiamo infine sostituito.

$$T_{CPU/DMA} = T_{DMA\_PROGRAM} + T_{INT\_EOT} \qquad ma: \quad T_{DMA\_PROGRAM} = T_{INT\_EOT}$$

$$Quindi: \qquad T_{CPU/DMA} = 2 * T_{INT\_EOT} = \cancel{2} * \left[\frac{1}{\cancel{2}} T_{IO\_INTERRUPT}\right] = (1 + W) * T_{ISR}$$

In questo esempio, abbiamo finora ottenuto che il tempo per trasferire un byte con la tecnica DMA è pari a quello impiegato con una tecnica con Interrupt; si noti, chiaramente, che questo vale solo nel dato esempio **.**

Quanto detto però vale solo per un singolo byte poiché, come abbiamo già detto, nel caso di trasmissione di N byte, con la tecnica basata su interrupt, si impiega N volte il tempo di trasferimento del singolo byte:

$$T_{IO\_INTERRUPT(N\_BYTE)} = N * (1 + W) * T_{ISR} = N * T_{IO\_INTERRUPT}$$

Definiamo allora lo **speedup** come il **rapporto tra il tempo impiegato nel sistema gestito con interrupt (sistema più lento) e quello gestito con DMA (sistema più veloce):**

$$\boldsymbol{Speedup_{IO(N\_BYTE)} = \frac{T_{IO\_INTERRUPT(N\_BYTE)}}{T_{CPU/DMA}}}$$

Otteniamo quindi:

$$Speedup_{IO(N\_BYTE)} = \frac{T_{IO\_INTERRUPT(N\_BYTE)}}{T_{CPU/DMA}} = \frac{N * \cancel{T_{IO\_INTERRUPT}}}{\cancel{T_{IO\_INTERRUPT}}} = N$$

Questo valore dipende dal fatto che la CPU ha programmato anticipatamente di effettuare N volte le operazioni di I/O.

Le formule adoperate finora valgono se vale la condizione ideale: quando viene utilizzata un'architettura Harvard e quando CPU e DMA non entrano mai in conflitto per l'accesso alla memoria.

Performance reale per DMA

Abbiamo trattato un caso puramente ideale poiché è molto improbabile che la CPU non debba accedere alla memoria contemporaneamente al DMA (che non nascano conflitti per l'accesso in memoria).

Nel caso reale, infatti, in un'architettura Harvard, possono avvenire dei conflitti tra CPU e DMA quando si effettua l'accesso alla memoria dati, ad esempio per istruzioni di load/store (che avvengono mediamente per circa il 20%).
Nel caso di conflitto, la performance per DMA non può essere calcolata come descritto precedentemente.

Supponiamo di avere N trasferimenti tramite DMA e una loro frazione f in cui si verifica un conflitto tra CPU e DMA (con 0 < f < 1 : valore, in percentuale, che indica la possibilità che possa avvenire un conflitto).
Quando avviene il conflitto **la CPU attende che il DMA completi il trasferimento**, quindi rispetto al caso ideale, nel caso reale il tempo di I/O totale aumenta: al tempo per programmare il DMA ($T_{DMA\_PROGRAM}$) e al tempo per l'esecuzione dell'interrupt end of transfer (T_{INT_EOT}) si aggiunge il tempo in cui la CPU resta in attesa del completamento del trasferimento da parte del DMA.
Si supponga che il DMA impieghi per trasferire un byte dalla sorgente alla destinazione un tempo pari a $\mathbf{T_{TRANSFER\_DMA}}$, dunque la CPU dovrà attendere un tempo pari a $T_{TRANSFER\_DMA}$ quando avviene un conflitto.

A questo punto allora definiamo il tempo di I/O totale con la tecnica DMA (reale) come segue:

$$\boldsymbol{T_{CPU/DMA} = T_{DMA\_PROGRAM} + T_{INT\_EOT} + f * N * T_{TRANSFER\_DMA}}$$

Ovvero come la somma di:
$T_{DMA\_PROGRAM}$: tempo per programmare il DMA,
T_{INT_EOT} : tempo per l'esecuzione dell' interrupt EOT
$T_{TRANSFER\_DMA}$: tempo che la CPU trascorre attendendo la terminazione del trasferimento da parte del DMA, quest'ultimo viene moltiplicato per gli N conflitti che avvengono e la durata temporale di ognuno: f

Per mettere a confronto la tecnica DMA (reale) con le tecniche di polling e di interrupt, possiamo esprimere diversamente la formula tratta:

Il tempo per trasferire i dati il DMA è minore del tempo che impiegherebbe la CPU alla trasferimento dello stesso dato nella tecnica con interrupt, poiché la CPU, oltre al trasferimento dei dati, perderebbe dell'altro tempo per l'esecuzione delle istruzioni:

$$T_{TRANSFER\_DMA} < T_{IO\_INTERRUPT}$$

A questo punto possiamo allora riprendere la definizione di **speedup** che ci da informazione su **quanto più veloce va la tecnica DMA rispetto alla tecnica con interrupt:**

$$Speedup_{DMA/CPU(Byte)} = \frac{T_{IO\_INTERRUPT}}{T_{TRANSFER\_DMA}} = s$$

In questo modo possiamo scrivere $T_{IO\_INTERRUPT}$ in funzione di s e di $T_{TRANSFER\_DMA}$:

$$T_{TRANSFER\_DMA} = \frac{1}{s} T_{IO\_INTERRUPT}$$

Possiamo quindi sostituire nella formula che abbiamo prima trattato, in modo da mettere in relazione il tempo impiegato per trasferire i dati con DMA ed il tempo per trasferirli con Interrupt:

$$T_{CPU/DMA} = T_{DMA\_PROGRAM} + T_{INT\_EOT} + f * N * T_{TRANSFER_{DMA}} \Rightarrow$$

$$\boldsymbol{T_{CPU/DMA} = T_{DMA\_PROGRAM} + T_{INT\_EOT} + \left[\frac{1}{s} f * N * T_{IO_{INTERRUPT}}\right]}$$

Riproponendo allora la condizione precedente:

$$T_{DMA\_PROGRAM} = T_{INT\_EOT} = \frac{1}{2} T_{IO\_INTERRUPT} = \frac{1}{2} (1 + W) * T_{ISR}$$

Avremo che:

$$T_{CPU/DMA} = T_{IO\_INTERRUPT} * \left(1 + \frac{1}{s} * f * N\right) = T_{ISR} * (1 + W) * \left(1 + \frac{1}{s} * f * N\right)$$

In quanto: $T_{IO\_INTERRUPT} = (1 + W) * T_{ISR}$

Con questo esempio abbiamo rapportato i tempi impiegati per la gestione di I/O con tecnica interrupt e quelli con tecnica DMA.

A questo punto, è possibile calcolare lo **Speedup in caso di conflitti** (nel caso reale):

$$Speedup_{IO(N\_BYTE)} = \frac{T_{IO\_INTERRUPT(N\_BYTE)}}{T_{CPU/DMA}} = \frac{N * \cancel{T_{IO\_INTERRUPT}}}{\cancel{T_{IO\_INTERRUPT}} * \left(1 + \frac{1}{s} * f * N\right)}$$

Quindi:

$$\boldsymbol{Speedup_{IO(N\_BYTE)} = \frac{N}{\left(1 + \frac{1}{s} * f * N\right)}}$$

Nel caso ideale lo Speedup è pari ad N, nel caso reale, in cui vi sono conflitti, il valore si discosta da N, esso infatti ha un valore minore di N poiché viene posto al denominatore una quantità che sarà maggiore di 1 [solo quando (1/s * f * N) vale zero si ha il denominatore unitario, tuttavia, sotto questa condizione si cadrebbe nuovamente nel caso ideale in cui non si hanno conflitti].

Aumentando i conflitti, chiaramente, diminuisce lo Speedup perché, anche se i trasferimenti li effettua il DMA, quando si hanno dei conflitti per l'accesso al bus dati, la CPU resta comunque in attesa quindi si spreca del CPU time: maggiore è f e minore è lo speedup.
S invece è lo speedup misurato tra i tempi totali del DMA e quelli dell'interrupt: mette a confronto quanto il DMA è più veloce rispetto all'Interrupt. Quanto più grande è questo valore, tanto più piccolo è il rapporto (1/s * f * N).

Abbiamo finora trattato performance in ambito temporale, calcolando i tempi di CPU trascorsi durante le operazioni di gestione dell'I/O, osservando che per polling ed interrupt la CPU veniva impiegata anche per il trasporto dei dati mentre per DMA ciò non avviene ma, in caso di conflitto, **la CPU deve comunque attendere la terminazione delle operazioni svolte dal DMA**.
A differenza di quanto fatto con Polling ed Interrupt, nel caso di DMA non abbiamo ancora parlato di tempo di I/O, ovvero del tempo totale impiegato per la trasmissione dei dati per poi trarre informazioni circa il calcolo del throughput;
abbiamo parlato del tempo $T_{CPU/DMA}$ che è il tempo che la CPU attende affinché il DMA termini la propria esecuzione, liberando il bus, ma questo tempo non è sufficiente per il calcolo del throughput poiché non da informazioni circa le operazioni di I/O ma solo del tempo impiegato al DMA per trasportare i dati.

Domande sul DMA:

1. Si utilizzi il DMA come sistema di gestione dell'I/O. Quale delle seguenti opzioni è vera?
   a. Solo nel caso di utilizzo dell'architettura Harvard si ottiene un minore tempo di trasferimento rispetto alla gestione con Interrupt.
   Falso: L'utilizzo dell'architettura Harvard è vantaggioso poiché si hanno due bus diversi: uno per le istruzioni ed uno per i dati, a differenza dell'architettura di Von Neumann; in questo modo DMA e CPU lavorano per la maggior parte dei casi separatamente (solo nel caso di istruzioni di load/store questo non vale).Tuttavia, anche se non si utilizzasse l'architettura Harvard, con il DMA si otterrebbero dei risultati migliori dell'interrupt.

   b. Solo ne caso di utilizzo dell'architettura Harvard con zero conflitti risulta:
   $T_{IO\_DMA(NByte)} < T_{IO\_INTERRUPT}$
   Falso: all'aumentare di N aumenta anche $T_{IO\_DMA(NByte)}$ quindi non è sempre vero che questo è maggiore del tempo impiegato dall'interrupt per un byte $T_{IO\_INTERRUPT}$

   c. $T_{IO\_DMA(NByte)} < T_{IO\_INTERRUPT(NByte)}$
   Vero: confrontando il sistema di I/O con gestione DMA e con gestione con Interrupt risulta sempre vero che il tempo impiegato dal DMA per gestire N byte è sempre minore del tempo impiegato dall'Interrupt per gestire lo stesso numero N di byte.

   d. Nessuna delle precedenti

2. Considerando una gestione del sistema di I/O per trasferire N Byte con DMA, architettura Harvard, zero conflitti, quale delle seguenti opzioni è vera?
   a. $T_{IO\_DMA(NByte)}$ non dipende dal numero di Byte (N) da trasferire.
   Falso: Il tempo totale di input/output dipende dal numero di byte che devono essere trasmessi, sia quando il DMA attende la CPU che nel caso in cui avviene il contrario:
   $T_{IO\_DMA} = (s + N)/N * T_{TRANSFER\_DMA}$

   b. $T_{IO\_DMA(NByte)}$ dipende solo dal parametro $s = T_{IO\_INTERRUPT}/T_{TRANSFER\_DMA}$
   Falso: Il tempo totale di input/output dipende sia dal parametro speedup (S) che dal numero di byte che devono essere trasmessi :
   $T_{IO\_DMA} = (s + N)/N * T_{TRANSFER\_DMA}$

   c. Il thoughput di I/O, in byte/sec, f_{IO_DMA} è direttamente proporzionale al $T_{TRANSFER\_DMA}$
   Falso: Non può essere direttamente proporzionale, in quanto l'uno è il reciproco dell'altro: $f_{IO\_DMA} = 1/T_{IO\_DMA}$ in cui il tempo $T_{TRANSFER\_DMA}$ è compreso nel Tempo T_{IO_DMA}

   d. Nessuna delle precedenti
   Vero: in questo caso, nessuna delle risposte precedenti è vera.

Calcolo del Throughput

Affrontando le tre tecniche siamo riusciti a dedurre quale sia l'efficienza nei tre casi differenti e nel caso di tecnica di gestione dell'I/O basata su Polling ed Interrupt abbiamo anche calcolato il tempo di I/O, ovvero il tempo che trascorre nell'intera gestione dell'I/O.
In quest'ultimo caso è stato facile calcolare il throughput in quanto si è trovato come inverso del tempo totale di I/O. Diversa è invece la situazione per tecnica DMA in quanto non si è ancora trovato il tempo totale di I/O ma solo il tempo in cui la CPU resta ad attende il DMA (il $T_{CPU/DMA}$).

Parliamo nello specifico del calcolo del throughput nei tre casi; definiamo:

- **N: Numero di Byte trasferiti dal sistema di I/O**
- **T_{IO} : Tempo medio speso per il trasferimento di un byte con il sistema I/O considerato**

Allora facendo il reciproco del tempo medio per il trasferimento di un byte si può trovare il **throughput massimo** che il sistema di I/O è in grado di smaltire:

$$f_{IO} = \frac{1}{T_{IO}}$$

Questo valore limita la massima frequenza acquisibile da una sorgente analogica, in quanto, per il teorema del campionamento, si deve campionare ad una frequenza f_c che sia maggiore o uguale al doppio di una certa frequenza f_{MAX} , quest'ultima è f_{IO}.

$$f_C \geq 2\, f_{MAX} \quad => \quad T_C \leq \frac{1}{2\, f_{MAX}}$$

Di conseguenza, si può osservare che il tempo di I/O deve essere minore o uguale al tempo di I/O, altrimenti di avrebbe la perdita di informazioni.

$$T_{IO} \leq T_C$$

Affrontiamo adesso nello specifico il calcolo del throughput nelle tre tecniche di gestione dell'I/O.

Calcolo del throughput: tecnica Polling

Definiamo Il tempo $T_{IO\_Polling}$ come tempo medio speso dal sistema di I/O per trasmettere un byte ed è tutto a carico della CPU.
Otteniamo che, nel caso del polling, il **throughput** vale:

$$f_{IO\_Polling} = \frac{1}{T_{IO\_Polling}}$$

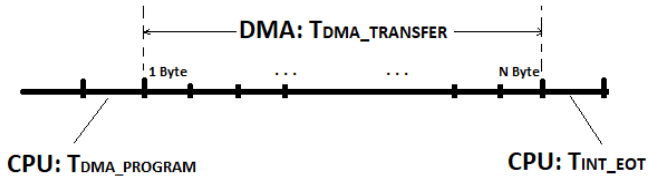
Possiamo stabilire il valore del $T_{IO\_Polling}$ mediante i due valori caratteristici della tecnica Polling: W (peso) e x_M (valor medio: numero medio di tentativi)

Calcolo del throughput: tecnica Interrupt

Definiamo Il tempo $T_{IO\_Interrupt}$ come tempo medio speso tutto dalla CPU per trasmettere un byte.
Otteniamo che, nel caso dell'interrupt, il **throughput** vale:

$$f_{IO\_Interrupt} = \frac{1}{T_{IO\_Interrupt}}$$

Calcolo del throughput: tecnica DMA [Ideale]



Utilizzando un'architettura Harvard, nel caso ideale della tecnica DMA (in cui non vi sono conflitti tra DMA e CPU), definiamo il tempo totale impiegato dal sistema di I/O per trasferire N byte come sommatoria del tempo speso dalla CPU per programmare il DMA ($T_{DMA\_PROGRAM}$), del tempo che la CPU spende attendendo il trasferimento dei dati da parte del DMA ($T_{TRANSFER\_DMA}$) e del tempo che la CPU impiega per eseguire l'interrupt di EOT (End Of Transfer: T_{INT_EOT}).

Il tempo speso per l'I/O include i tre tempi che abbiamo citato.

Per esprimere più facilmente in forma chiusa il tempo totale ci serviamo dello Speedup che avevamo già introdotto precedentemente:

$$s = Speedup_{DMA/CPU(N\_BYTE)} = \frac{T_{IO\_INTERRUPT}}{T_{TRANSFER\_DMA}}$$

Assumiamo nuovamente che, nel trasferimento di un byte, il tempo per programmare il DMA sia pari al tempo di gestione dell'ISR dell'interrupt di EOT (quindi supponendo che i due CPU Time siano uguali), e che essi siano pari alla metà del tempo che la CPU spenderebbe per il trasferimento di un byte con tecnica interrupt:

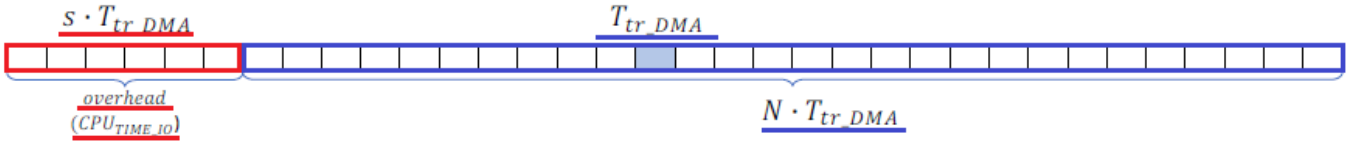
$$T_{DMA\_PROGRAM} = T_{INT\_EOT} = \frac{1}{2} T_{IO\_INTERRUPT}$$

Sotto queste opportune condizioni possiamo mettere a confronto il tempo di overhead per un sistema utilizzante tecnica di gestione con interrupt ed il tempo di trasporto impiegato dal DMA per la tecnica di gestione con DMA:

$$T_{IO\_INTERRUPT} = s * T_{DMA\_PROGRAM}$$

Il tempo impiegato dal DMA per seguire gli N trasferimenti sarà pari ad N volte il tempo per trasferire un singolo byte:

$$T_{TRANSPORT\_DMA\ (N\_BYTE)} = N * T_{TRANSFER\_DMA}$$



Se non vi sono conflitti DMA e CPU lavorano in parallelo, si ha allora che il tempo totale della gestione dell'I/O con DMA è pari a:

$$T_{IO\_DMA(N\_BYTE)} = T_{IO\_INT.} + N * T_{TRAN\_DMA} = s * T_{TRAN\_DMA} + N * T_{TRAN\_DMA}$$

Da cui si ottiene il Tempo di I/O per N Byte come:

$$\boldsymbol{T_{IO\_DMA(N\_BYTE)} = (s + N)\, T_{TRAN\_DMA}}$$

Possiamo allora ricavare il **tempo medio di gestione dell'I/O per singolo byte trasmesso con DMA**, che è pari a:

$$T_{IO\_DMA} = \frac{(s+N)}{N} * T_{TRANSFER\_DMA}$$

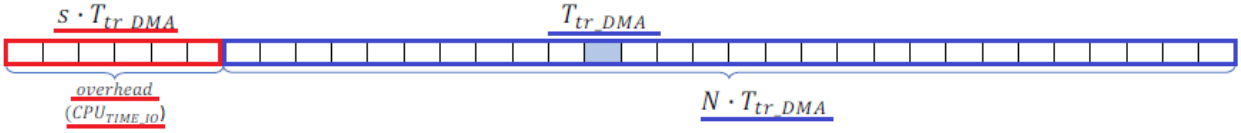
Il tempo speso nel caso ideale non dipende solo da N ma anche dall'overhead che comprende i tempi persi dalla CPU per la programmazione del DMA e per l'esecuzione dell'interrupt service routine (ISR) a seguito dell'interrupt di end of transfer (EOF).

Possiamo allora calcolare il **Throughput per la tecnica di gestione dell'I/O con DMA [nel caso ideale]** trovando il reciproco del tempo medio appena trovato:

$$f_{IO\_DMA} = \frac{1}{T_{IO\_DMA}} = \frac{N}{(s+N)} * \frac{1}{T_{TRANFER\_DMA}}$$

Il fattore $\frac{N}{s+N}$ influisce sulle performance del sistema di I/O, poiché dato un sistema avente una certa velocità nel trasferire ogni byte pari a $T_{TRANSFER\_DMA}$ avremo che le performance del sistema, ovvero il throughput del sistema, non sarà solo dato da questa velocità ma dal rapporto di quest'ultima con il fattore citato. Possiamo allora affermare che:

- $Se\ N \gg s \quad Allora \quad \frac{N}{s+N} \cong 1$
  Se N (numero di byte da trasferire) è molto più grande di s (Speedup) allora il fattore è pari a 1, di conseguenza, affermeremo che **l'Overhead della Cpu pesa poco**:



- $Se\ N = s \quad Allora \quad \frac{N}{s+N} = \frac{1}{2}$
  Altrimenti se N = s si può matematicamente dimostrare che il fattore è pari a ½ , quindi di conseguenza **il throughput si dimezza**

Quindi una volta che la CPU programma il DMA, è bene che questo venga programmato per trasferire un numero elevato di byte (esempio: 1000 byte) così da massimizzare l'efficienza e quindi il throughput; qualora si programmasse il DMA per pochi bit ciò non accadrebbe e si avrebbe un throughput peggiore poiché l'overhead della CPU pesa più del trasporto dei dati.

Calcolo del throughput: tecnica DMA [Reale] - CPU che attende il DMA

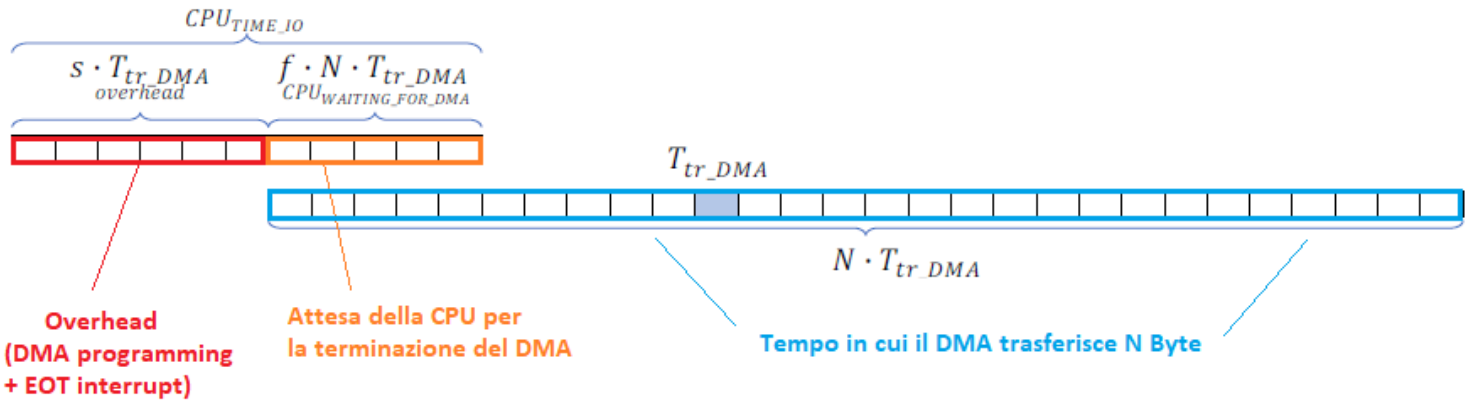
Utilizzando un'architettura Harvard, nel caso reale della tecnica DMA, può accadere che CPU e DMA entrino in conflitto per accedere al bus dati (ad esempio quando avvengono delle operazioni di load/store e il DMA deve accedere al bus dati).

Consideriamo allora il caso reale di tecnica DMA e studiamo il throughput di conseguenza: consideriamo l'utilizzo di un'architettura Harvard e una frazione di f conflitti.

Assumiamo, come prima, che:

$$T_{DMA\_PROGRAM} = T_{INT\_EOT} = \frac{1}{2} T_{IO\_INTERRUPT}$$

L'Overhead totale della CPU, in questo caso, è dato dall'overhead trovato precedentemente sommato al tempo impiegato dalla CPU ad attendere che il DMA termini la propria esecuzione quando si verifica un conflitto di accesso in memoria.
Il tempo che attende la CPU per la terminazione del DMA viene chiamato



Il tempo impiegato per il trasferimento dei byte è identico al caso ideale (quando non vi sono conflitti), questo lo si può notare anche dal grafico stesso: nel caso reale si ha il tempo di attesa che la CPU spreca quando avviene un conflitto, tuttavia, quando ciò avviene il DMA è già in esecuzione in modo parallelo. Dal grafico di evince che i tempi di attesa della CPU si "sovrappongono" al tempo in cui il DMA trasferisce N Byte, quindi, se si vede la durata totale della gestione dell'I/O, si ha lo stesso tempo impiegato nel caso ideale.
Si ha nuovamente:

$$T_{IO\_DMA(N\_BYTE)} = T_{IO\_INT.} + N * T_{TRAN\_DMA} = s * T_{TRAN\_DMA} + N * T_{TRAN\_DMA}$$

Da cui si ottiene il Tempo di I/O per N Byte come:

$$\boldsymbol{T_{IO\_DMA(N\_BYTE)} = (s + N)\, T_{TRANSFER\_DMA}}$$

Possiamo allora ricavare il **tempo medio di gestione dell'I/O per singolo byte trasmesso con DMA**, che è pari a:

$$\boldsymbol{T_{IO\_DMA} = \frac{(s + N)}{N} * T_{TRANSFER\_DMA}}$$

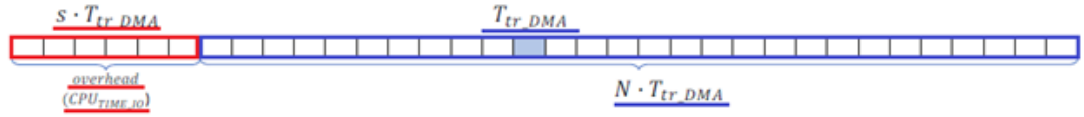
Esattamente come nel caso ideale.

Tuttavia, anche se il tempo totale di gestione dell'I/O è rimasto invariato, i tempi di CPU sono peggiorati rispetto al caso ideale, poiché all'overhead dato dai tempi di programmazione del DMA e di gestione dell'interrupt di EOT mediante ISR (tempi presenti anche nel caso ideale), si somma anche il tempo di attesa che la CPU spreca quando il DMA accede al bus dati.

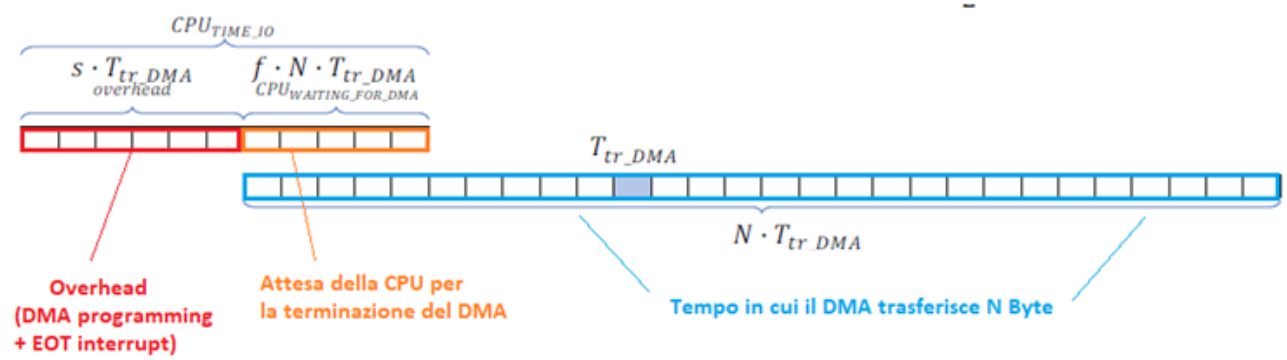
Dal punto di vista del throughput (e dunque del tempo totale di gestione dell'I/O, in quanto quest'ultimo è l'inverso del throughput) non ci si rende conto di ciò poiché il tempo che la CPU spende durante l'attesa si sovrappone ai tempi che impiega il DMA nel trasferimento dei dati.

Dal punto di vista della gestione dell' I/O non si ha alcuna perdita di performance, dal punto di vista di sistema le performance diminuiscono poiché la CPU spreca più tempo.

DMA ideale [0 conflitti]:



DMA reale [f conflitti]:



Come si può notare, nel caso di DMA reale, il tempo di CPU è dato da:

$$T_{CPU\_DMA\_REALE} = f * N * T_{TRANSFER\_DMA}$$

Quindi il tempo di CPU stavolta dipende dal numero N di byte che vengono trasferiti e dal numero f di conflitti che si verificano.

[Nel caso reale del DMA il tempo CPU speso è equivalente al caso ideale: FALSO
Nel caso reale del DMA il tempo CPU aumenta all'aumentare di f o di N: VERO **]**

Avendo trovato il tempo medio complessivo di gestione dell'I/O per il trasferimento di un byte, possiamo calcolare il **Throughput per la tecnica di gestione dell'I/O con DMA [nel caso reale]** trovando il reciproco del tempo medio appena trovato:

$$f_{IO\_DMA} = \frac{1}{T_{IO\_DMA}} = \frac{N}{(s+N)} * \frac{1}{T_{TRANSFER\_DMA}}$$

Calcolo del throughput: tecnica DMA [Reale] - DMA che attende la CPU

Escludendo che quando vi sia un conflitto possano accedere al bus entrambi (poiché altrimenti vi sarebbero danni permanenti), si deve scegliere se "dare precedenza" alla CPU o al DMA, noi abbiamo supposto finora di dar priorità a quest'ultimo.

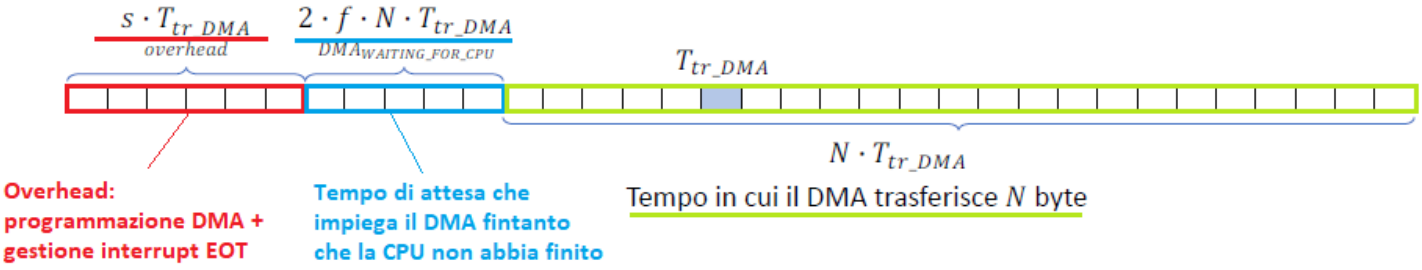
Supponiamo adesso di voler provvedere nel modo contrario: **quando avviene un conflitto il DMA arresta la propria esecuzione e da priorità alla CPU nell'eseguire le operazioni sul bus dati.** Calcoliamo allora adesso il tempo totale di I/O necessario per il trasferimento dei byte.

Per sapere quanto tempo il DMA resti in attesa della CPU, dobbiamo stimare il tempo che la CPU spende per eseguire un'operazione sui dati. Quindi supponiamo di avere un tempo di load/store pari a due volte il tempo impiegato dal DMA per il trasporto dei dati; quindi il DMA è più veloce nel trasferimento dei byte dato che impiega metà del tempo impiegato dalla CPU e questo non è irrealistico poiché la CPU deve gestire l'intera esecuzione di un processo (fetch, load, …) mentre il DMA deve solo occuparsi del trasporto ed è quindi "alleggerito" rispetto alla CPU. Avremo:

$$T_{CPU\_LOAD/STORE} = 2\ T_{TRANSFER\_DMA}$$

E continuiamo a supporre che valga la condizione:

$$T_{DMA\_PROGRAM} = 2\ T_{INT\_EOT} = \frac{1}{2} T_{IO\_INTERRUPT}$$



Quando il DMA riprende l'esecuzione non avrà ancora trasferito tutti gli N byte, li trasferirà dopo il tempo atteso.
In questo scenario, dal punto di vista della CPU vi è un'ottimizzazione poiché prevalgono i tempi per l'esecuzione; dal punto di vista del DMA i tempi aumentano quindi aumenta il tempo di gestione dell'I/O. Quindi **scegliendo questa tecnica i tempi di I/O peggiorano**. Otteniamo infatti:

$$T_{IO\_DMA(NByte)} = T_{IO\_INTERRUPT} + f * N * T_{LOAD/STORE} + N * T_{TRANS\_DMA} =$$
$$= T_{IO\_INTERRUPT} + 2 * (f * N * T_{TRANS\_DMA}) + N * T_{TRANS\_DMA} =$$
$$= s * T_{TRANS\_DMA} + 2 * (f * N * T_{TRANS\_DMA}) + N * T_{TRANS\_DMA} =$$
$$= [s + 2 * (f * N) + N] * T_{TRANS\_DMA}$$

Tempo di I/O per trasferire N byte: $T_{IO\_DMA(NByte)} = [s + 2 * (f * N) + N] * T_{TRAN\_DMA}$

Tempo di I/O per trasferire un byte:

$$T_{IO\_DMA} = \frac{[s + 2 * (f * N) + N]}{N} * T_{TRANSFER\_DMA}$$

Avendo ottenuto il tempo di I/O per trasferire un singolo byte possiamo ricavare il **throughput per tecnica di gestione con DMA (con il DMA che attende la CPU):**

$$f_{IO\_DMA} = \frac{1}{T_{IO\_DMA}} = \frac{N}{[s + 2 * (f * N) + N]} * \frac{1}{T_{TRANSFER\_DMA}}$$

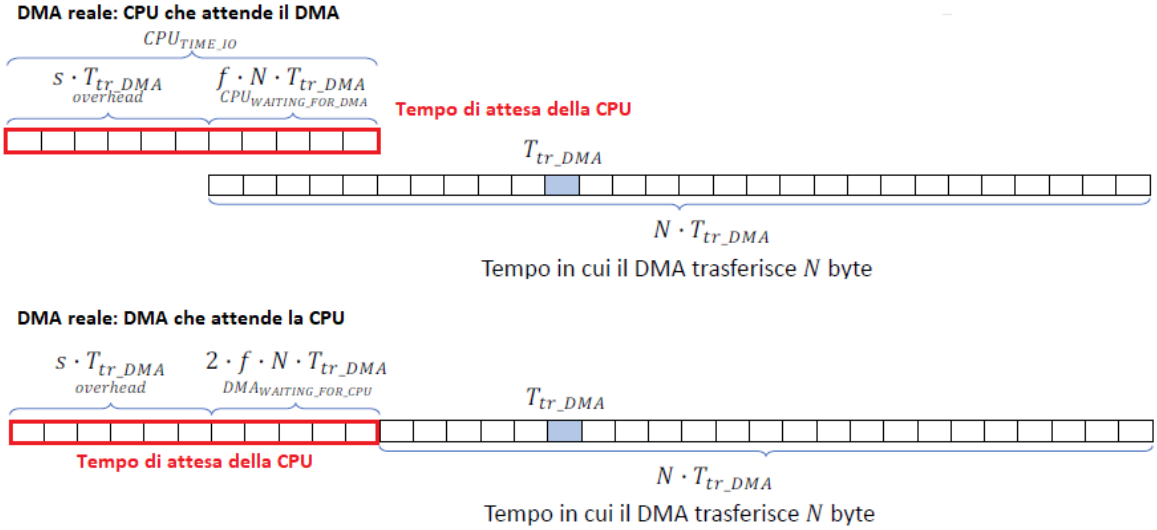
In tutti i tre casi: **DMA ideale, DMA reale con CPU che attende il DMA, DMA reale con DMA che attende la CPU** abbiamo trovato il throughput in funzione degli stessi parametri:

- **$T_{TRANSFER\_DMA}$**: Tempo impiegato dal DMA per trasferire un Byte da sorgente a destinatario
- **f**: percentuale di conflitti (se vi sono i conflitti)
- **s**: speedup (quante volte il DMA è più veloce della CPU)
- **N**: numero di byte

È quindi importante avere questi quattro dati per riuscire a calcolare il throughput nei tre casi.

[Qual è la miglior performance del DMA?
- Nel caso di conflitti quella del DMA reale in cui la CPU attende il DMA
- Nel caso di 0 conflitti non cambia nulla tra quella reale con la CPU attende il DMA e quella ideale]

Il tempo CPU speso nei casi reali è uguale: sia che la CPU attenda il DMA, sia che il DMA attenda la CPU;



Quindi <u>dal punto di vista del **throughput:**</u>

- Tra DMA ideale e DMA reale con il DMA che attente la CPU non cambia nulla
- Tra DMA ideale e DMA reale con la CPU che attende la DMA si ha un calo di prestazioni.

Per sapere quante sorgenti analogiche può supportare un sistema si deve tenere in considerazione il throughput: per DMA reale con l'attesa da parte del DMA non varia nulla rispetto al caso reale, mentre ciò non vale per DMA reale con l'attesa da parte della CPU, in cui si ha un calo del throughput. In questo caso, per avere nuovamente lo stesso throughput, è necessario diminuire il numero di dispositivi utilizzati.

Quindi <u>dal punto di vista del **CPU time:**</u> indipendentemente dalla scelta di chi far attendere tra DMA e CPU, <u>avendo quindi N conflitti</u>, si ha un calo di performance tra caso ideale e caso reale (nel caso reale si impiega un maggior CPU time).

Acquisizione sorgenti analogiche (sistema IoT)

Assumiamo che ciascun valore campionato del segnale analogico (da acquisire) venga convertito dall'A/D in un byte [Un byte è sufficiente per rappresentare un campione]. Se $\boldsymbol{TIO/byte}$ rappresenta il tempo speso dal sistema di I/O per acquisire un byte, si può calcolare la max frequenza del segnale analogico acquisibile rispettando il teorema del campionamento:

$$\boldsymbol{f_{MAX[SEGNALE]} \leq \frac{1}{2T_{IO/Byte}} = \frac{1}{2T_C} = \frac{1}{2} * f_C} \qquad \boldsymbol{Dove\ T_c = T_{IO/Byte}}$$

Questo vale per qualsiasi tipo di sistema I/O, qualunque sia la tecnica di gestione utilizzata.

In base alla tecnica di gestione dell'I/O utilizzata, avremo differenti tempi di I/O totali: $T_{IO/Byte}$ studiamo quindi i vari casi affrontati finora:

1. **Polling:**

   Nel caso del polling avremo:

   $$T_{IO\_POLLING} = \frac{T_{TRANSFER}}{\eta_{POLLING}} = T_{TRANSFER}[w(x_m + 1) + 1]$$

   [NB: $T_{IO\_POLLING/Byte}$ e $T_{IO\_POLLING}$ si riferiscono alla stessa cosa, in precedenza lo abbiamo chiamato $T_{IO\_POLLING}$ e quindi continueremo a chiamarlo così, ciò varrà anche per le altre tecniche]

   $T_{IO\_POLLING}$ è il tempo impiegato utilizzando la tecnica polling per trasmettere un byte. In cui:
   - W: overhead impiegato dal polling quando non è presente il dato
   - X_m: valor medio del numero di tentativi effettuati per ottenere il dato

   Avremo allora:

   $$\boldsymbol{f_{MAX[SEGNALE]} \leq \frac{1}{2T_{IO/Byte}} \quad => \quad f_{MAX[SEGNALE]} \leq \frac{1}{2T_{TRANSFER}[w(x_m + 1) + 1]}}$$

2. **Interrupt:**

   Nel caso l'interrupt avremo:

   $$T_{IO\_INTERRUPT} = \frac{T_{ISR}}{\eta_{INTERRUPT}} = T_{ISR}(1 + w)$$

   $T_{IO\_INTERURPT}$ è il tempo impiegato utilizzando la tecnica polling per trasmettere un byte.
   In questo caso l'efficienza è data da: $\eta = \frac{1}{1+w}$ in cui:
   - W: overhead impiegato dall'interrupt per fermare/riprendere l'esecuzione del foreground program

   Possiamo inoltre osservare:
   -T_{ISR} : tempo impiegato per eseguire l'Interrupt Service Routine, che si occuperà di trasmettere il byte

   Avremo allora:

   $$\boldsymbol{f_{MAX[SEGNALE]} \leq \frac{1}{2T_{IO/Byte}} \quad => \quad f_{MAX[SEGNALE]} \leq \frac{1}{2T_{ISR}(1 + w)}}$$

3. DMA con architettura Harvard:

Caso a: zero conflitti

Nel caso ideale in cui non si hanno conflitti di accesso al bus dati tra CPU e DMA (quest'ultimo vi entra solo quando deve eseguire operazioni di load/store) avremo un tempo di I/O pari a :

$$T_{IO\_DMA(Zero\_Conflitti)} = \frac{s+N}{N} T_{TRANSPORT\_DMA}$$

$T_{IO\_DMA(Zero\_Conflitti)}$ è il tempo impiegato utilizzando la tecnica DMA per trasmettere un byte quando, idealmente, non si hanno conflitti tra CPU e DMA per l'accesso sul bus dati.
Avremo che:

- s: rapporto tra il tempo impiegato con la gestione con interrupt ed il tempo impiegato per il trasporto dei dati con DMA; indica il miglioramento delle performance tra le due tecniche

$$s = Speedup_{DMA/CPU(Byte)} = \frac{T_{IO\_INTERRUPT}}{T_{TRANSPORT\_DMA}}$$

- N: byte trasferiti mediante interrupt

Avremo allora:

$$\boldsymbol{f_{MAX[SEGNALE]} \leq \frac{1}{2T_{IO/Byte}} \quad => \quad f_{MAX[SEGNALE]} \leq \frac{1}{2} \frac{N}{s+N} \frac{1}{T_{TRANSPORT\_DMA}}}$$

Caso b: f conflitti

Nel caso reale possono avvenire un numero f di conflitti di accesso al bus dati tra CPU e DMA (quest'ultimo vi entra solo quando deve eseguire operazioni di load/store), possiamo gestire i conflitti in due modi: facendo attendere al DMA la terminazione della CPU, così che quando quest'ultima termina di utilizzare il Bus Dati il DMA potrà accedervi, oppure facendo attendere alla CPU la terminazione del DMA, in questo modo quando il DMA finisce di utilizzare il Bus Dati, la CPU potrà accedervi.

Caso b1: f conflitti con DMA che attende la CPU

In questo caso avremo un tempo totale di I/O dato da:

$$T_{IO\_DMA(f\_Conflitti)} = \frac{s+(2*f*N)+N}{N} T_{TRANSPORT\_DMA}$$

Questo è il caso peggiore per quanto riguarda il tempo di Input/Output (come si può vedere, rispetto al caso ideale - e quindi al caso in cui la CPU attende il DMA - avremo un termine che si va a sommare al numeratore: [2*f*N]) ma è vantaggioso per quanto riguarda i tempi di CPU poiché essa è privilegiata in caso di conflitti.

Avremo allora:

$$\boldsymbol{f_{MAX[SEGNALE]} \leq \frac{1}{2T_{IO/Byte}} \quad => \quad f_{MAX[SEGNALE]} \leq \frac{1}{2} \frac{N}{s+(2*f*N)+N} \frac{1}{T_{TRANSPORT\_DMA}}}$$

Caso b2: f conflitti con CPU che attende il DMA

In questo caso ci ritroveremo le stesse quantità trovate per il caso ideale:

$$T_{IO\_DMA(Zero\_Conflitti)} = \frac{s+N}{N} T_{TRANSPORT\_DMA}$$

Questo è il caso migliore per quanto riguarda il tempo di Input/Output (esso è infatti identico al caso ideale in cui vi sono zero conflitti poiché si ha un elevato grado di parallelismo che ottimizza i tempi di I/O) ma è svantaggioso per quanto riguarda i tempi di CPU poiché essa deve fermarsi ogni volta che si ha un conflitto con il DMA.

Avremo allora:

$$\boldsymbol{f_{MAX[SEGNALE]} \leq \frac{1}{2T_{IO/Byte}} \quad => \quad f_{MAX[SEGNALE]} \leq \frac{1}{2} \frac{N}{s+N} \frac{1}{T_{TRANSPORT\_DMA}}}$$

Esempio: Massima frequenza acquisibile a seconda della tecnica di gestione di I/O

Da un sistema di IoT in cui si voglia valutare, in base alla tecnica di gestione dell'I/O, la massima frequenza di segnale analogico acquisibile nei casi:

a) Polling, con parametri: $T_{TRANSPORT}$ = 100µs; x_m = 50; w = 0.2
b) Interrupt, con parametri $T_{ISR} = T_{TRANSPORT}$ = 100µs; w = 0.2

[NB: w è nei due casi differente: nel caso a si riferisce all'overhead speso dalla CPU nell'interrogazione dei device durante il polling; nel caso b) l'overhead riguarda il blocco e la ripresa del foreground program per l'esecuzione dell'ISR da parte della CPU]

SOLUZIONE

a) Polling:

$$f_{MAX[SEGNALE]} \leq \frac{1}{2T_{IO/Byte}} \quad => \quad f_{MAX[SEGNALE]} \leq \frac{1}{2T_{TRANSFER}[w(x_m+1)+1]}$$

$$f_{MAX[SEGNALE]} \leq \frac{1}{2T_{TR.}[w(x_m+1)+1]} = \frac{1}{2*100*10^{-6}[0.2(50+1)+1]} = 0.44 KByte/s$$

[In questo caso, un segnale che avente una frequenza maggiore di 0.44 Kbyte/s non è acquisibile dal sistema. Se si aumenta x_m significa che vi sono più tentativi che non sono andati a buon fine, infatti peggiorano le performance.]

b) Interrupt:

$$f_{MAX[SEGNALE]} \leq \frac{1}{2T_{IO/Byte}} \quad => \quad f_{MAX[SEGNALE]} \leq \frac{1}{2T_{ISR}(1+w)}$$

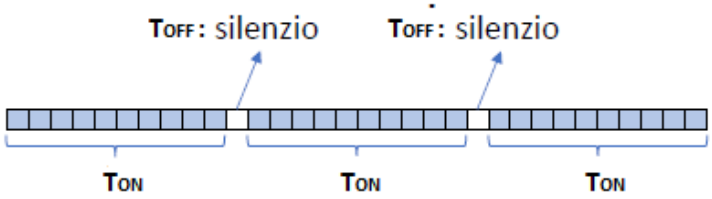
$$f_{MAX[SEGNALE]} \leq \frac{1}{2T_{ISR}(1+w)} = \frac{1}{2*100*10^{-6}(1+0.2)} = 4.1\ KByte/s$$

[Con parametri numericamente identici (ma concettualmente molto differenti tra loro) abbiamo una frequenza che è 10 volte maggiore: possiamo campionare un segnale che ha 10 volte la frequenza del segnale acquisito con la tecnica polling]

Esercizio: Acquisizione di un burst dati con diverse tecniche di gestione di I/O

In un sistema IoT occorre gestire l'acquisizione di un burst di dati caratterizzato da un tempo di interarrivo variabile in cui valore minimo è 100µs.
Ogni burst è di 1MByte e dura un secondo con frequenza di picco di 1MByte/s.

Un burst dati è un segnale in cui si hanno informazioni nel periodo di T_{ON} fino ad un tempo di silenzio T_{OFF}. Ogni volta che ci sono dati si è nel periodo di T_{ON}, quando non vi è informazione ci si ritrova nel periodo di T_{OFF}. Normalmente, il tempo di T_{ON} e T_{OFF} non sono costanti.

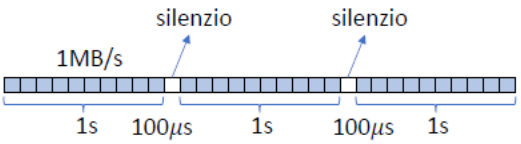


In questo esempio, il tempo di T_{ON} dura 1 secondo, entro questo tempo arriva 1MByte di dati. Il tempo di interarrivo è variabile ma ha valore minimo di 100 microsecondi, questo significa che avremo un tempo di silenzio T_{OFF} che può variare ma che avrà valore minimo 100 microsecondi; lo assumeremo pari a 100 microsecondi così da metterci nelle condizioni peggiori.

- **Caso 1: Gestione tramite interrupt:**
  Calcolo del Tempo di I/O richiesto:
  $$1MByte/s \Rightarrow T_{IO\_INTERRUPT} = \frac{1}{1MByte} = 10^{-6}\ s/byte = 1\mu s/byte$$
  Assumendo che T_{ISR} = 0.8µs (tempo che, chiaramente, è - e deve essere - minore del tempo totale di I/O), calcolare il valore di w_{MAX} per rispettare i requisiti di velocità richiesti al sistema di I/O.

- **Caso 2: Gestione tramite DMA:**
  a) **Zero conflitti => f = 0**
  b) **Con conflitti => f ǂ 0**
  Se $T_{DMA\_PROGRAM} = T_{INT\_EOT} = K$, calcolare il valore di K_{MAX} e di $T_{TRANSFER\_DMA\_MAX}$ nei due casi (a e b)

SOLUZIONE



Caso 1: Gestione tramite interrupt
Il tempo di IO interrupt si trova come:

$$T_{IO\_INTERRUPT} = (1 + w_{MAX})T_{ISR}$$

Questo, come abbiamo detto, deve essere minore o uguale a 1 microsecondo, avremo quindi:

$$T_{IO\_INTERRUPT} = (1 + w_{MAX})T_{ISR} \leq 1\mu s$$

Assumiamo che $T_{IO\_INTERRUPT}$ sia pari a 1 microsecondo e otteniamo:

$$T_{IO\_INTER.} = (1 + w_{MAX})T_{ISR} = 1\mu s \ \rightarrow (1 + w_{MAX})0.8\mu s = 1\mu s \ \rightarrow w_{MAX} = 0.25$$

Caso 2: Gestione tramite DMA

a) Zero conflitti => f = 0

In questo caso, il tempo speso per trasferire gli N byte (1mln di byte in 1 secondo nel nostro caso) deve essere dato da N per il tempo di trasferimento del singolo byte ed il risultato deve rientrare in 1 secondo [Devo riuscire entro un secondo ad inviare tutti gli N byte (1mln)]

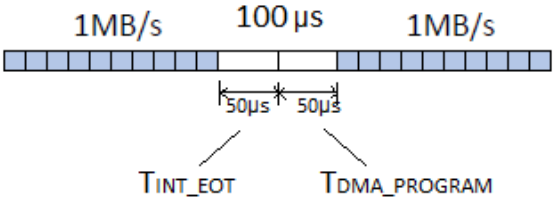
$$NT_{TRANSPORT\_DMA} \leq 1s \;\rightarrow\; 10^6 * T_{TR\_DMA} \leq 1s \;\rightarrow\; T_{TR\_DMA\_MAX} = 1\mu s/byte$$

Una volta che abbiamo trovato il tempo affinché un byte venga correttamente "smaltito", la CPU può programmare il DMA affinché questo vada allo stesso ritmo, avremo allora un tempo $T_{DMA\_PROGRAM}$; un altro tempo speso si ha per la gestione dell'ISR di EOT che avverte la fine della trasmissione. Questi tempi sono dei tempi spesi aggiuntivi, sarebbe giusto far in modo che non venga perso del tempo utile per i dati per queste operazioni, ma vengano spesi, ad esempio, i periodi "di silenzio". Imponiamo allora che questi tempi occupino massimo 100µs (la massima durata del tempo di silenzio).

$$T_{DMA\_PROGRAM} = T_{INT\_EOT} = \frac{1}{2} * T_{IO\_INTERRUPT} = k$$

Come abbiamo detto, la somma del tempo per programmare il DMA e quello per gestire l'Interrupt di EOT è uguale e la loro somma deve valere 100µs; quindi:

$$T_{DMA\_PROGRAM} + T_{INT\_EOT} = 2k = 100\mu s$$



Da cui si ottiene: $k = 50\mu s$

Quando si è nel periodo di attività T_{ON} si deve essere in grado di smaltire il traffico dei byte; quando si è nel periodo di inattività, la CPU comunica la fine della trasmissione attraverso l'interrupt EOT e programma il DMA per il prossimo burst dati

b) Con conflitti => f ∤ 0

Supponiamo adesso un caso reale in cui vi è una percentuale di conflitti, assumeremo che:
f = 0.25

Caso b1: CPU attende il DMA In questo caso vi è la stessa situazione del caso ideale, basti rivedere quanto detto nel punto a.

Caso b2: DMA la CPU

Assumiamo che $T_{LOAD/STORE} = 2T_{TRANSFER\_DMA}$ [Questo vale da consegna, non è sempre valido]
Vogliamo allora che il tempo di overhead più il tempo per trasferire N Byte rientri in 1 s:

$$2 * f * N * T_{DMA\_TR} + N * T_{INT\_EOT} \leq 1s \;\rightarrow\; (2 * f + 1)N * T_{DMA\_TR} \leq 1s$$

Avendo i dati a disposizione possiamo risolvere la disequazione e trovare $T_{DMA\_TRANSFER}$:

$$T_{DMA\_TR} \leq \frac{1}{(2 * f + 1)N} \;\rightarrow\; T_{DMA_{TR_{MAX}}} \leq \frac{1}{(2 * 0.5 + 1)10^6} = 0.66\mu s/byte$$

Senza conflitti di ha un tempo di 100µs/byte con i conflitti si arriva a 0.66 questo significa che il DMA per smaltire il throughput deve essere più veloce (dato che deve anche gestire i conflitti) [Il professore ci dà o il burst senza alcuna frequenza massima: dovremo allora smaltire il traffico burst oppure ci da un sistema avente un segnale con una frequenza massima.]