Oggi parleremo di trasferimenti DMA. Tutti i dispositivi collegati al BUS sono dotati di registri: sia il dispositivo DMA sia i dispositivi di IO sono dotati di registri. Il processore può quindi interagire coi dispositivi attraverso operazioni di lettura e scrittura in questi registri. La modalità di accesso preferenziale a tali registri è la mappatura in memoria: in sostanza il processore vede tutto il resto del sistema come se fossero celle di memoria RAM (la maggior parte sarà effettivamente RAM ma alcune celle saranno in realtà i registri dei dispositivi collegati al BUS). Ci sarà quindi un sottoinsieme di indirizzi utilizzati per mappare in memoria i registri. Il nostro Processore può interagire solo con cose raggiungibili dal BUS, non può accedere a un’unità a disco. Ma può accedere ai registri mappati in memoria che costituiscono da drive per tale registro. Questo è di base molto limitante, quindi l’idea è di copiare parte delle informazioni dell’unità a disco all’interno di un blocco di memoria RAM chiamato Buffer, ogni cui modifica comporta la modifica anche dei dati presenti nella memoria a disco. Come si trasferisce in maniera efficiente i blocchi di dati dall’unità a disco alla RAM e viceversa? In teoria non ci sarebbe bisogno di un dispositivo DMA: un modo brutale di eseguire queste operazioni è avere il processore che chiede un solo valore per volta all’unità disco, lo prende dal driver e lo copia in RAM e continua fino ad aver copiato un intero blocco. Questa tecnica sarebbe uno spreco di tempo da parte del processore. Il DMA è quindi un coprocessore, programmabile dal processore principale, a cui delegare l’operazione di trasferimento di dati. Il processore quindi scrive delle istruzioni nel DMA (relazione Master-Slave), che devono contenere almeno 5 informazioni: l’indirizzo del dispositvo IO da cui prelevare/a cui inviare i dati, il blocco dell’unità a disco su cui operare, l’indirizzo del Buffer, la dimensione del Buffer (cioè quante celle copiare, si può fare con un indicizzamento o indicando la cella finale) e indicare se è un’operazione di lettura o di scrittura. Inviate queste 5 istruzioni il processore può procedere a fare altre cose, nel frattempo il DMA assume il ruolo di Master per l’accesso al BUS ed esegue i comandi che gli sono stati dati dal processore. Il DMA da quindi il comando (di lettura) all’unità a disco, il risultato è salvato nel driver ed è spostato dal driver al buffer dal DMA e continua il ciclo fino ad aver copiato tutto il necessario. Concluso il ciclo di lettura il DMA può iniziare un ciclo di scrittura, in cui i valori salvati nel buffer vengono salvati uno a uno nell’unità a disco dopo averli spostati nel driver (i cicli avvengono con l’auto-incremento dell’indirizzo RAM a cui punta il DMA e con confronto di esso con l’indirizzo finale). Quando finisce la scrittura il DMA può inviare una interrupt al processore per segnalargli la fine dell’operazione.

Tutto ciò rende il sistema più efficiente grazie alla gerarchia di memorie cache che permette al processore di evitare di accedere alla RAM mentre la usa il DMA. Un’ulteriore complicazione nell’implementazione della cache è che ci possono essere degli indirizzi (quelli del buffer) etichettati come “da non tradurre nella cache” (ciò è fatto a livello di tabelle di traduzione) perché i dati lì presenti possono essere spostati senza l’intervento del processore.

Se si volesse effettuare una scrittura dalla RAM all’unità a disco il procedimento è analogo, si affida al DMA di spostare i valori in RAM (tramite autoincremento dell’indirizzo del buffer) al driver I/O e da lì copiarli nell’unità a disco.

Grazie alla presenza della Cache il processore può andare avanti lo stesso, anche se il BUS è occupato. Qualsiasi programma che non abbia bisogno di accesso alla RAM non copiata nella cache può essere mandato in esecuzione, quindi questo rende molto più efficiente l’esecuzione di programmi.

Il discorso funziona analogamente per i dispositivi I/O come per i dispositivi di interconnessione di rete. Anche lì c’è il registro mappato in memoria e l’interazione avverrà a sempre a cura del DMA tramite la copia o del registro nel buffer (o l’inserimento in esso di certi valori).

La cosa che non ci piace è che per ogni elemento trasferito servono due accessi al bus (uno di lettura e uno di scrittura). Quindi la complessità dell’operazione in sé rimane invariata, il DMA ha solo preso il posto della CPU. Si ha un piccolo overhead durante la programmazione del dispositivo DMA (si perdono un paio di cicli di clock).

Tutto ciò ha quindi senso solo se i Buffer sono di una dimensione sostenuta (perché l’overhead è ammortizzato solo da trasferimenti di grandi dimensioni). Per questo motivo in un sistema POSIX i dispositivi I/O sono divisi in due classi: abbiamo i dispositivi “a blocchi” e i dispositivi “a carattere”. La differenza dal punto di vista fisico è che i Block Device richiedono il trasferimento di tante celle di memoria (e quindi sono affidate al DMA), mentre i Character device sono quelli per cui non vale la pena di utilizzare il DMA e sono usati direttamente dal processore. Questa è una distinzione stabilita nel sistema operativo. Se si chiama ls -l /dov si vede quali dispositivi sono riconosciuti come a blocchi e quali come a carattere.

Il DMA non risolve il problema dell’utilizzo del bus. Vediamo un’alternativa. La struttura proposta vede il nostro processore con la sua cache, la RAM e il dispositivo di IO. La differenza è che il DMA anziché essere vicino al processore si trova all’interno del dispositivo di I/O. Fase iniziale: il processore esegue dei cicli di scrittura per scrivere nei registri di DMA (che non è un coprocessore in questo caso) e dà al DMA le istruzioni per trovare i blocchi da copiare nella memoria a disco e la posizione e la dimensione del Buffer in RAM. Alla fine il DMA manderà comunque un segnale di interrupt al processore. Quello che cambia è che il registro temporaneo si trova già vicino al DMA e quindi non è richiesto l’uso del BUS per ottenerne il valore. Quindi in questo caso per trasferire dalla memoria RAM all’unità a disco o viceversa è necessario UN solo accesso al BUS per elemento (anziché 2 come sarebbe col DMA come coprocessore).

Questa seconda versione è quella utilizzata nei dispositivi moderni e questa configurazione in cui il DMA è contenuto nel dispositivo I/O è detta BUS-MASTERING. Da un punto di vista fisico è più complicato e costoso da realizzare, perché per ogni dispositivo di BUS-MASTERING c’è un DMA in più, e quindi anche un Master in più a cui il BUS deve stare attento quando assegna la priorità. Si deve quindi anche realizzare un BUS più complicato. Tuttavia i vantaggi (ossia l’accesso al BUS in meno) danno una velocità di funzionamento maggiore (oltre che la possibilità di avere più trasferimenti DMA contemporanei) tale da rendere quasi trascurabili gli svantaggi in fase di progettazione.

A questo punto cosa rimane da ottimizzare? L’interazione tra il processore e i dispositivi DMA BUS-MASTER. Quando il processore può andare a programmare i trasferimenti DMA? Nel caso normale il processore può andare a programmarlo quando si presenta la necessità (per esempio un’applicazione sta chiedendo di aprire un file in modalità lettura e le informazioni ovviamente sono sull’unità a disco) e in questo caso c’è ben poco da fare: il processore manda le istruzioni al DMA e poi passa a fare qualcos’altro, l’unico ulteriore overhead sarà la risposta all’interruzione (nel caso del processore Amber abbiamo i registri sdoppiati per ottenere ciò). Ci sono degli altri casi in cui è possibile programmare in anticipo il DMA. L’esempio più tipico è la ricezione di informazioni dalla rete. In tal caso il sistema operativo, se è presente la NIC, può programmare la scheda di interconnesione di rete per inviare la prossima informazione ricevuta dalla rete all’interno di un buffer prestabilito: così facendo si pre-programma il trasferimento DMA e far sì che questo scarichi subito il proprio buffer interno in quello della memoria RAM (così se si riceve un secondo messaggio dalla rete, il primo è già stato salvato nel buffer in RAM e il secondo rimane memorizzato internamente alla NIC, evitando di perdere il messaggio). Un ritardo dal processore può comunque portare alla perdita di informazioni quando addirittura arriva un terzo messaggio prima che il processore non abbia risposto alla richiesta di interruzione abbastanza in fretta.

Rispondendo all’interruzione il processore può dare l’informazione salvata in RAM a un’applicazione che ne ha bisogno e riprogrammare il DMA per cambiare il Buffer RAM su cui salvare il prossimo (cioè il secondo) messaggio.

Quindi se vogliamo dei dispositivi di interconnessione di rete molto veloci (in cui possono arrivare tanti messaggi in breve tempo) occorre trovare una soluzione ancora diversa rispetto al DMA BUS-MASTERING. Questa soluzione è ovviamente più complicata. Infatti bisogna poter programmare in anticipo tanti trasferimenti senza bisogno di interventi da parte del processore, in modo da non avere problemi anche se il processore non sente l’interruzione per troppo (perché magari è mascherata) tempo mentre arrivano informazioni su informazioni dalla rete. La soluzione che è stata trovata circa una ventina di anni fa è la seguente:

Oltre ai Buffer si specifica un’area di memoria particolare che viene utilizzata in memoria condivisa da parte del processore e dal dispositivo DMA. In questa area di memoria viene trasferita una struttura dati abbastanza sofisticata visibile e manipolabile sia dal processore che dal DMA. In questa area di memoria vengono inserite DUE strutture dati ad Anello (chiamate Ring): ne servono due perché ce ne deve essere una per ogni direzione del trasferimento di dati (c’è quindi un Ring di input e uno di output). In queste strutture dati si inseriscono i puntatori ai buffer di memoria. C’è una fase di inizializzazione in cui il processore costruisce questa struttura dati.   
Supponiamo che il processore voglia una struttura in cui ci sono 3 Buffer di ricezione e uno di trasmissione, deve quindi scrivere gli indirizzi di questi 4 Buffer nella zona RAM condivisa. Così facendo si sta passando un informazione al dispositivo DMA riguardo a quali Buffer saranno utilizzati (che al momento sono vuoti). Fatto questo il processore programma il dispositivo DMA dicendogli qual è l’indirizzo della RAM in cui può trovare i due ring di input e di output. A questo punto sia il processore che il DMA partono ed entrambi possono in qualsiasi momento andare a leggere o scrivere informazioni in uno di quei due ring. Così facendo il processore non deve più programmare ogni volta il dispositivo DMA. Se il processore vuole inviare un messaggio lo scrive nel buffer del ring di uscita e cambiandone il flag di uscita da 0 a 1. La Nic a questo punto non ha bisogno di niente: ogni tot microsecondi effettua una lettura dei flag di occupazione dei ring, quando vede che uno di quelli di uscita è passato da 0 a 1 procede con il trasferimento dei dati tramite il DMA, che li sposta nel buffer interno della Nic e poi li invia. Quando tutti i dati sono stati copiati il DMA riporta a 0 il flag di occupazione del ring e invia una interrupt al processore. Se il processore vuole mandare più di un messaggio uno dietro l’altro (per esempio tre) serviranno più buffer (due in più nel nostro esempio), ma una volta che li ha a lui basta riempirli con i valori da inviare e cambia i bit di occupazione e quando la Nic parte (scandendo la struttura ad anello secondo un verso prestabilito) allora effettua l’invio dei messaggi tramite il trasferimento del DMA ed azzerando uno alla volta i bit di occupazione dopo che è finita l’operazione sul buffer corrispondente (ed aver iniziato il trasferimento del successivo).

Il processore quando ha bisogno di inviare nuovi messaggi va a vedere qual è il primo buffer vuoto e lo riempie, se scandendo il ring vede che tutti i buffer sono pieni può decidere di aggiungere un altro buffer.

In ricezione i buffer sono inizialmente vuoti col flag a 0. Quando arriva il primo messaggio la Nic prende il primo buffer vuoto, effettua il trasferimento DMA, cambia il flag da 0 a 1 e invia un’interruzione. La Nic può fare la stessa cosa quando le arriva il secondo e il terzo messaggio (seguendo lo stesso giro di scansione del Ring seguito dal processore): l’interruzione è quindi una cortesia nei confronti del processore, che così non deve continuamente accedere all’area di memoria dei buffer, come invece fa la Nic (tramite l’operazione di “polling”, che le permette di interrogare costantemente la struttura dati per vedere se c’è qualcosa da fare, fintanto che non sta eseguendo già un’operazione). Nella sfortunata situazione in cui i buffer di ricezione si riempiano il DMA NON può creare altri buffer, perché l’unico vero Master della RAM è la CPU (perché esegue il sistema operativo, che decide quali aree della RAM sono da usare come Buffer). Se il processore non aggiunge nuovi Buffer, può capitare che a un certo punto la Nic si debba fermare (in questo caso si possono comunque avere delle perdite di messaggi).

Se la versione di prima era il BUS-MASTERING, quella presentata adesso si chiama RING-BASED DMA. Ovviamente il DMA di questa cosa è più complicato di quanto è nel BUS-MASTERING, è praticamente un microprocessore (perché non ha più solo dei registri ad incremento. Tipicamente il programma da seguire da questo processore che funziona come DMA è hard-coded ed è presente fisso all’interno della Nic. Questo tipo di programmi normalmente è chiamato “Firmware”, per distinguerlo da quei programmi caricati in memoria RAM (come il sistema operativo). Anche il processore principale può avere un firmware, che viene caricato durante la fase di bootstrap.