12 Direct Memory Access (DMA)

Versão 1.0

Durante o trabalho com microcontroladores, o leitor já deve ter notado que é comum a necessidade de se fazer transferências de dados entre memória e dispositivos de entrada e saída (I/O). Por exemplo, a necessidade de se enviar ou receber um pacote de dados pelas portas UART, I^2C ou SPI. Outro bom exemplo é a transferência do resultado de conversões do A/D para a memória.

Em todos os exemplos acima citados se faz necessária a intervenção da CPU que desempenha o papel de atravessador, lendo os dados de um determinado local e escrevendo-os em um outro local. Este é um problema recorrente. Vejamos o caso em que há uma certa quantidade de dados de um dispositivo de I/O e se necessita escrevê-los na memória. Precisaremos de um contador para a quantidade de dados e um ponteiro para indicar onde escrever na memória. O programa de recepção realizaria então as seguintes operações:

Enquanto contador for diferente de zero, repetir

1. Ler dado no dispositivo de I/O;
2. Escrever dado na memória usando o ponteiro;
3. Incrementar o ponteiro de escrita;
4. Decrementar o contador.

No caso deste exemplo, ocupamos a CPU para realizar inúmeras vezes uma tarefa muito simples: repetir um certo número de vezes um ciclo de leitura e escrita. Ainda mais, se temos leitura e escrita do mesmo dado, por que não transferir esse dado diretamente entre os envolvidos, então, sem a necessidade de passar pela CPU? Essa é a essência do DMA: permitir que o dado trafegue diretamente entre a memória e o dispositivo de I/O. Em palavras ainda mais simples, permitir ao I/O acesso direto à memória.

Antes de avançar, vamos sumarizar a seguir as três formas básicas de se transferir dados entre I/O e memória:

1. **Polling**: A CPU monitora constantemente o dispositivo de I/O e faz a transferência quando necessário. Neste caso a CPU é completamente consumida monitorando do dispositivo de I/O.
2. **Interrupção**: Usando a tabela de vetores de interrupção, o programador indica qual função deve ser chamada toda vez que o dispositivo de I/O precisar transferir um dado. Agora a CPU está livre para realizar outras tarefas, que são interrompidas quando surge a necessidade de uma transferência. É claro que se gasta tempo de CPU para executar a função indicada para atender à interrupção.
3. **DMA**: Agora o programador configura um circuito especial denominado Controlador de DMA. Toda vez que for necessário realizar a transferência de um dado, esse controlador assume o barramento e realiza a transferência. Agora a CPU está livre para realizar outras tarefas e apenas perde alguns ciclos de relógio a cada transferência realizada.

Com a intenção de esclarecer uma dúvida muito comum, vamos fazer duas perguntas:

1) Das soluções acima citadas, qual é a mais rápida para se transferir dados entre a CPU e um dispositivo de I/O? A resposta óbvia é o DMA.

2) Das soluções acima citadas, qual é a mais lenta para se transferir dados entre a CPU e um dispositivo de I/O? A resposta, agora não tão óbvia, é a interrupção. Isso acontece porque o mecanismo para se aceitar uma interrupção consome tempo. O leitor deve se lembrar de que são 6 ciclos para começar a executar a primeira instrução da função que atende à interrupção e, depois, mais 5 ciclos para o retorno.

Tendo uma ideia clara destes conceitos, vamos dar início ao estudo do recurso de DMA. Como no MSP430 todos os dispositivos de I/O estão mapeados em memória, as transferências por DMA sempre acontecem entre dois endereços de memória.

Dúvida sobre nomenclatura: Como será que fica melhor?

- Uma operação de DMA é composta por várias transferências, ou seja, a operação acontece até o contador chegar a zero.
- Uma transferência por DMA é composta por várias operações, ou seja, a transferência acontece até o contador chegar a zero.
- Fazer distinção entre ciclo de DMA e transferência por DMA. Ciclo é para um dado e Transferência para todos os dados?

12.0. Quero Usar o DMA e não Pretendo Ler Todo este Capítulo

Por Acesso Direto à Memória (DMA) se entende a possibilidade da troca de dados entre a memória e um dispositivo de I/O sem a intermediação da CPU. São duas as possibilidades:

- Ler do dispositivo de I/O e escrever na memória ou
- Ler da memória e escrever no dispositivo de I/O.

Sabemos que a CPU usa o barramento (endereços, dados e controle) para acessar memória e dispositivos de I/O. Esse acesso da CPU acontece através de ciclos de barramento que são de leitura ou escrita. Por exemplo, um ciclo para ler um dado da memória, um outro ciclo para escrever um dado num dispositivo de I/O etc. O que é especial numa transferência por DMA é que num mesmo ciclo de barramento temos uma leitura e uma escrita. Para que isto aconteça se faz necessário um dispositivo extra denominado de Controlador de DMA. Durante as transferências, o controlador de DMA assume o controle do barramento. Isto significa que a CPU fica "parada" enquanto acontecem as transferências por DMA.

Como o MSP430 mapeia em memória todos os dispositivos de I/O, então no caso desta arquitetura, as operações de DMA só envolvem memória:

- Num mesmo ciclo de barramento, ler um dado de um determinado endereço da memória e escrevê-lo em um outro endereço da memória.

Para programar uma operação de DMA, o usuário precisa responder às seguintes perguntas.

1) Qual canal de DMA será usado?
2) Qual o endereço de origem dos dados (DMAxSA) e como ele será alterado (DMASRCINCR)?
3) Qual o endereço de destino dos dados (DMAxDA) como ele será alterado (DMADSTINCR)?
4) Quantas transferências serão realizadas (DMAxSZ)?
5) Qual o tamanho (8/16 bits) do dado que será lido (DMASRCBYTE)?
6) Qual o tamanho (8/16 bits) do dado que será escrito (DMADSTBYTE)?
7) Qual evento será responsável por disparar as transferências (DMAxTSEL)?
8) Qual modo a ser empregado (simples, bloco, rajada) (DMADT)?
9) O que deve acontecer quando o contador (DMAxSZ) chegar a zero?
10) Haverá repetição?

A arquitetura MSP430 prevê até 8 canais de DMA. Na versão F5529 temos disponível apenas 3 canais de DMA. É importante que o leitor tenha claro os registradores usados por cada canal. Um canal **x** de DMA faz uso dos seguintes registradores:

- DMA**x**CTL → controle do canal **x**;
- DMA**x**SA → endereço de origem dos dados;
- DMA**x**DA → endereço de destino dos dados;
- DMA**x**SZ → quantidade de transferências;
- DMACTL**n** → seleção da origem do disparo (n = 0, 1, ... 4, note que **n** ≠ **x**);

--- Acho que este tópico ainda pode ser aumentado ----

Depois deste breve resumo, é fortemente recomendado que o leitor estude o restante deste capítulo.

12.1. Fundamentos do DMA

Neste tópico vamos trabalhar o conceito de DMA. Ao invés partirmos para o estudo específico do MSP430, faremos uma abordagem mais genérica pois assim a conceituação fará mais sentido. Adiante apresentaremos os tópicos específicos para o MSP430.

Para a presente explicação vamos usar a arquitetura de um típico computador, onde se tem a CPU, a memória e os dispositivos de I/O. Nas figuras a seguir, a memória será indicada como "Memo" e o dispositivo de I/O simplesmente pela sigla "I/O". Para facilitar a compreensão, vamos imaginar o caso em que se precisa receber 1.000 dados do dispositivo de I/O e escrevê-los na memória a partir do endereço 0x2400. Dentro do que já vimos, precisaremos de:

- contador = 1000
- ponteiro = 0x2400

No caso de uso da técnica de polling, ilustrado na Figura 12.1, a CPU ficaria constantemente consultando do dispositivo de I/O para saber se tem dado disponível. Quando a resposta é positiva, a CPU acessa o I/O para ler o dado e depois, usando o ponteiro, escreve esse dado na memória. Em seguida ele incrementa o ponteiro de escrita e decrementa o contador para saber se já recebeu a quantidade especificada. As linhas vermelhas ilustram o percurso realizado pelo dado em questão. Um ponto importante é percepção de que a CPU foi um atravessador no caminho que o dado percorreu, desde o I/O até a memória. Se, por um lado, a CPU fica totalmente consumida no processo de consultar o dispositivo de I/O, pelo outro, ela tem como responder, quase que imediatamente, à disponibilidade de um dado.

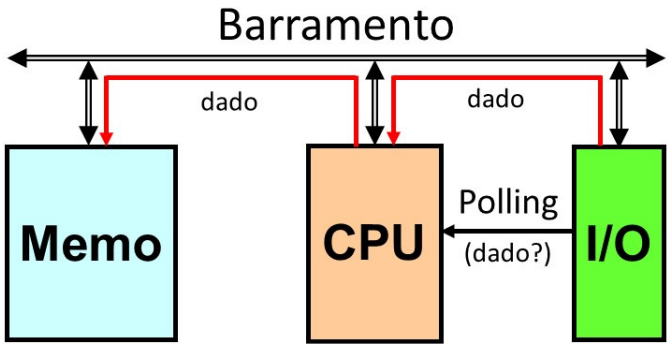


Figura 12.1. Diagrama ilustrativo do polling sendo usado para transferir dados entre I/O e memória. A linhas em vermelho ilustram o percurso realizado por um dado qualquer.

Vamos agora analisar o caso de uso da técnica de interrupção, ilustrado na Figura 12.2. Agora CPU está livre para realizar outras tarefas. Quando o dispositivo de I/O tem um dado disponível, ele envia um pedido de interrupção para a CPU. Na figura, este pedido está representado por "IRQ" (*Interrupt Request*). Como já estudamos no capítulo de interrupção, a CPU finaliza a instrução atual e usando a tabela de vetores de interrupção chama a função especificada pelo usuário. Ao executar esta função, a CPU acessa o I/O para ler o dado e depois, usando o ponteiro, escreve esse dado na memória. Em seguida ela incrementa o ponteiro de escrita e decrementa o contador para saber se já recebeu a quantidade especificada e retorna às suas atividades. As linhas vermelhas ilustram o percurso realizado pelo dado em questão. Vemos que a CPU ainda é um atravessador no caminho que o dado percorreu, desde o I/O até a memória. Se, por um lado, a CPU ficou completamente livre para executar outras tarefas, pelo outro, ela demora mais para responder quando um dado está disponível. Isso é imposto pela latência da interrupção.

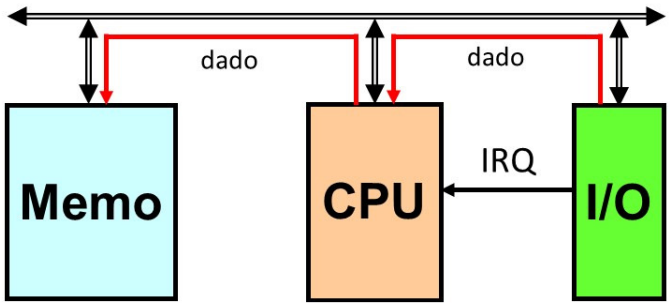


Figura 12.2. Diagrama ilustrativo da interrupção sendo usado para transferir dados entre I/O e memória. A linhas em vermelho ilustram o percurso realizado por um dado qualquer e IRQ (Interrupt Request) representa o pedido de interrupção.

Finalmente vamos agora analisar o caso de uso da técnica de DMA, ilustrado na Figura 12.3. Note que agora aparece um novo participante que é o Controlador de DMA, representado pela sigla CRTL DMA e que foi previamente configurado pelo programador com o ponteiro de escrita e a quantidade de dados a ser transferida. Novamente, a CPU está livre para realizar outras tarefas. Quando o dispositivo de I/O tem um dado disponível, ele envia um pedido de DMA (DREQ) para o Controlador de DMA. Este, por sua vez, envia um pedido de *Hold* para a CPU. Em resposta ao pedido de *Hold*, a CPU para suas operações e libera todo o barramento para o Controlador de DMA. Não está mostrado na figura, mas a partir deste ponto, o controlador sinaliza ao I/O para colocar o dado no barramento, ao mesmo tempo em que comanda a memória para escrever o dado no endereço previamente especificado. O importante agora é notar que o dado saiu do I/O e foi diretamente para a memória, sem qualquer atravessador. Após a transferência, o Controlador de DMA, incrementa seu ponteiro de escrita e decrementa o contador de transferências. Assim, o controlador fica pronto para um novo ciclo de DMA. As linhas vermelhas ilustram o percurso realizado pelo dado em questão. Vemos que a CPU ficou

completamente livre para executar outras tarefas e que não é mais um atravessador no caminho percorrido pelo dado. O preço a ser pago é que durante a transferência por DMA a CPU ficou parada, pois entregou o controle do barramento para o DMA. No caso do MSP430, cada transferência por DMA consome dois períodos de relógio (MCLK). Um tempo muito menor que o consumido pelas outras duas soluções.

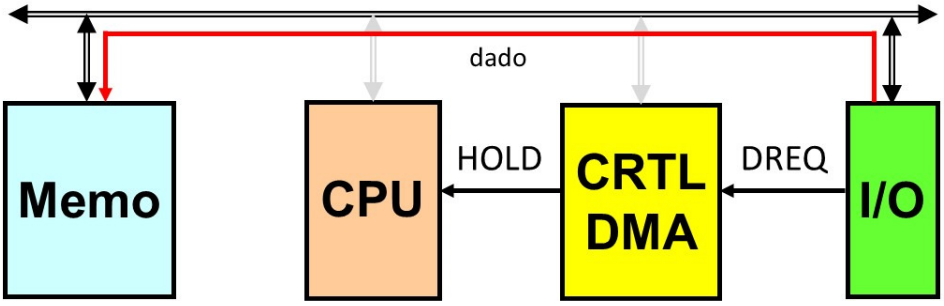


Figura 12.3. Diagrama ilustrativo do DMA sendo usado para transferir dados entre I/O e memória. A linha em vermelho ilustra o percurso realizado por um dado qualquer e DREQ (DMA Request) representa o pedido de DMA.

12.1.1. Detalhamento de uma Transferência por DMA

Os leitores que já estão satisfeitos com as explicações acima podem "pular" este tópico. Aqui vamos fazer um estudo mais detalhado e conceitual de uma transferência por DMA. Para que fique genérica, vamos adotar a arquitetura tradicional onde se tem dispositivos de I/O separados da memória, como é o caso do PC.

Toda transferência por DMA é iniciada pelo dispositivo de I/O. O dispositivo de I/O pode solicitar DMA para enviar um dado para a memória ou para receber um dado da memória. Note então, que temos algo especial: uma leitura e uma escrita simultâneas. A identificação da transferência se faz segundo a óptica da memória. Temos:

- DMA de escrita: ler de um dispositivo de I/O e escrever na memória e
- DMA de leitura: ler da memória e escrever num dispositivo de I/O.

A CPU só pode realizar, exclusivamente, o ciclo de leitura ou o ciclo de escrita, nunca os dois ao mesmo tempo. Ela não tem recursos (e nem é desejado que tenha) para esta simultaneidade. Daí a necessidade de um dispositivo extra, para assumir o controle do barramento e realizar o ciclo especial envolvendo leitura e escrita. Já que se vai adicionar este dispositivo extra, vamos capacitá-lo para incrementar (ou decrementar) o ponteiro de endereços e para decrementar o contador de transferências. Assim, seria possível programar esse controlador para realizar uma certa quantidade de transferências e, por exemplo, interromper a CPU quanto terminar.

Vamos agora para uma explicação bem mais detalhada das etapas para uma transferência por DMA. Vamos trabalhar a situação em que o dispositivo de I/O precisa transferir um dado para a memória, que está ilustrado na Figura 12.4. Note os círculos numerados que serão usados nesta explicação.

Antes, vamos detalhar os participantes. Temos, evidentemente a CPU. A CPU é o mestre do barramento, ou seja, controla as linhas de endereços e os sinais para ler da memória (MRD), escrever na memória (MWR), ler um dispositivo de I/O (IOR) e escrever num dispositivo de I/O (IOW). Sob o comando da CPU, este barramento realiza, em instantes diferentes, ciclos de leitura ou de escrita. O leitor pode identificar facilmente a memória (MEMO) e o dispositivo de I/O. Para não saturar a figura, desenhamos apenas um dispositivo de I/O, mas eles são vários. Finalmente, temos o controlador de DMA.

Vamos então à explicação passo a passo de um ciclo de DMA.

1) O programador prepara o controlador de DMA. No caso deste exemplo, configurando-o para ler do I/O e escrever na memória. Ele indica que o endereço 8000 é o primeiro a ser usado e que se deseja realizar 512 transferências. Esta era uma típica transferência para a leitura de um setor de 512 bytes do disquete flexível. Não está mostrado, mas o programador pode ainda pedir para o DMA interromper a CPU quando terminar as 512 transferências programadas.

2) O dispositivo de I/O, quando tem um dado pronto, envia um pedido de DMA (DREQ) para o controlador de DMA, que analisa se pode ou não atender a este pedido.

3) O controlador de DMA aceita o pedido e envia uma solicitação de HOLD para a CPU. O HOLD é uma solicitação para assumir o controle do barramento, ou seja, ele está pedindo para a CPU liberar todo o barramento.

4) A CPU provavelmente vai terminar o ciclo de barramento que está realizando e aceitar o pedido de HOLD. Aceitar o pedido de HOLD significa que a CPU coloca em alta impedância (Hi-Z) todas suas conexões com o barramento. Em outras palavras, o barramento fica livre para ser controlado por outro dispositivo. Após se desconectar do barramento, a CPU envia a confirmação (HLDA, abreviação de *Hold Acknowledge*) para o controlador de DMA, que então passa a ser o mestre do barramento.

5) Agora que o controlador de DMA é o mestre do barramento e ele usa o sinal DACK (DMA *Acknowledge*) para confirmar ao dispositivo de I/O que seu pedido de DMA vai acontecer.

6) O controlador de DMA coloca no endereço 0x8000, previamente programado, nas linhas de endereço do barramento.

7) O controlador de DMA prepara a memória para a escrita ativando a linha MWR.

8) O controlador de DMA ativa a linha IOR para fazer a leitura do dispositivo de I/O. O disposto que pediu DMA, ao receber os sinais DACK e IOR, tem a obrigação de disponibilizar seu dado nas linhas de dado do barramento.

9) Este é um instante importante. A memória está no modo de escrita (MWR ativado) no endereço 8000 e o dispositivo de I/O está no modo de leitura (IOR ativado). Então o dispositivo de I/O disponibiliza seu dado nas linhas de dado do barramento que vão diretamente até a memória. Notem, então, que estamos lendo do I/O e escrevendo na memória simultaneamente. Esta é a essência do DMA.

A finalização do ciclo de DMA é intuitivo. Faremos sua descrição por passos, porém preferimos não colocar mais números na figura.

10) Após a escrita no endereço 8000, o controlador de DMA incrementa seu ponteiro de endereços, que, provavelmente vai para 8001 e decrementa seu contador de transferências, que vai para 511.

11) O controlador remove o sinal DACK e as habilitações (IOR e MWR) de leitura e escrita que foram ativadas. Ainda mais, remove o endereço das linhas de endereço. Com isso, a memória sai do modo de escrita e o I/O do modo de leitura.

12) O dispositivo de I/O, retira o pedido DREQ quando percebe que o sinal DACK foi removido.

13) O controlador de DMA se desconecta do barramento e remove o pedido de HOLD. Com isso a CPU sabe que pode assumir o controle do barramento.

14) A CPU remove o sinal HLDA e assume novamente o controle do barramento e continua com suas atividades até que uma nova transferência por DMA seja solicitada.

Vimos então todos os passos de um ciclo de DMA de escrita em memória. Mutatis mutandis, se aplica para um ciclo de DMA para leitura de memória e escrita em I/O.

Como já dissemos, o controlador de DMA pode ser configurado para interromper a CPU quando seu contador de transferências chegar a zero. Existe mais um recurso. É comum a necessidade de se repetir operações de DMA. Por isso, os controladores têm o recurso de auto inicialização. Isto significa que após finalizar o total das transferências programadas, os valores iniciais são recuperados e tudo pode ser repetido.

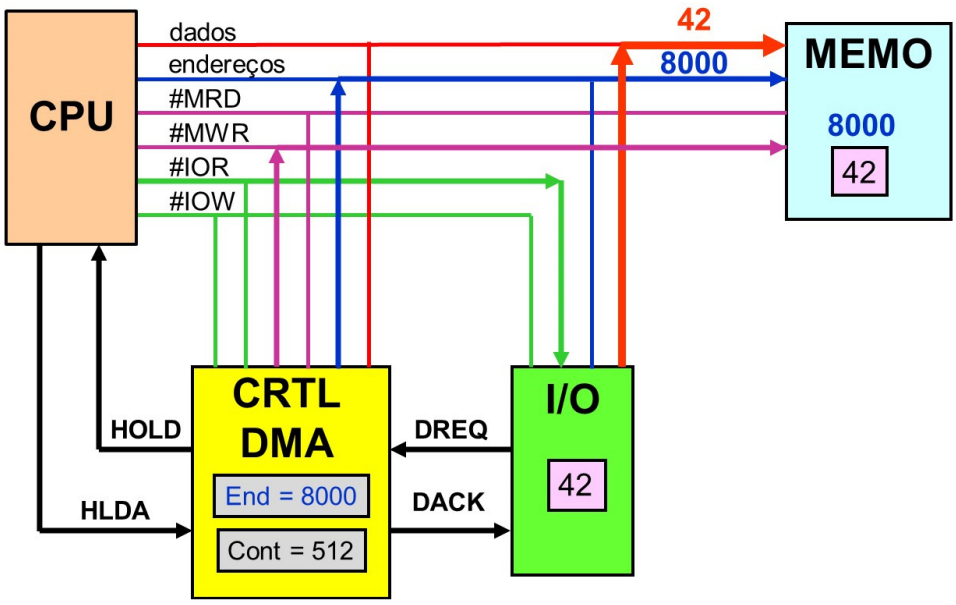


Figura 12.4. Diagrama detalhando os dispositivos envolvidos na transferência por DMA de um dispositivo de I/O para a memória.

Existe ainda o caso de DMA de memória para memória, ou seja, ler de uma posição da memória e escrever em outra posição de memória. Agora, é evidente que não é possível, ao mesmo tempo, habilitar a memória para leitura e escrita. Para este caso, o controlador tem um registrador para, provisoriamente, guardar o dado em trânsito. O DMA faz um ciclo para ler a memória e recebe o dado neste registrador provisório, em seguida, faz um ciclo de escrita em memória para escrever esse dado. São necessários, então, dois ponteiros: um de origem (fonte) e um de destino.

No caso do MSP430, todos os dispositivos de I/O estão mapeados em memória, por isso, as operações de DMA são sempre de memória para memória. O manual não fornece detalhes se existe a possibilidade de se habilitar simultaneamente leitura de um I/O e escrita na memória. Precisa ser investigado.

12.2. O Recurso de DMA do MSP430

A arquitetura MSP430 possui um controlador de DMA, o que permite a transferência de dados de um endereço para ou outro, sem a intervenção da CPU. Daí resultam duas grandes vantagens. A primeira é que se conseguem realizar transferências de dados com uma taxa elevada. A segundo é a economia de energia, pois essas transferências podem acontecer enquanto a CPU está em um modo de baixo consumo.

Listamos a seguir as principais características do Controlador de DMA do MSP430

- Até 8 canais independentes (na versão F5529 temos apenas 3 canais);
- Possibilidade de configurar a prioridade entre os canais;
- Apenas 2 ciclos de MCLK para cada transferência;
- Possibilidade de transferir bytes, words ou uma combinação deles;
- Transfere blocos de até 65.536 bytes ou words;
- Disparos configuráveis;
- Possibilidade de trabalhar com disparos por flanco ou por nível;
- Quatro modos de endereçamento e
- Transferência pode acontecer de forma simples, bloco ou rajada.

12.3. Diagrama de Blocos do DMA do MSP430

Vamos agora começar a estudar as particularidades do controlador de DMA do MSP430. Como já foi visto, todos os dispositivos de I/O do MSP430 estão mapeados em memória e, por isso, todo ciclo de DMA será entre duas posições de memória. A arquitetura MSP430 aceita até 8 canais, sendo que cada versão traz uma quantidade diferente de canais.

A Figura 12.5 apresenta o diagrama de blocos deste controlador de DMA, que traz algumas simplificações para beneficiar a clareza. Iniciamos pelo lado direito, onde se pode ver a memória, sobre a qual acontecem todas as operações de DMA. Note que esta memória é disputada entre todos os canais de DMA.

Cada canal de DMA possui registradores para sua configuração, que estão representados acima do retângulo amarelo e mais três registadores que destacamos abaixo.

- **Registrador do Endereço Fonte:** Este registrador contém o endereço de onde vai ser lido o dado a ser transferido. A cada transferência realizada, este registrador pode ser incrementado, decrementado ou permanecer estático.
- **Registrador do Endereço Destino:** Este registrador contém o endereço no qual vai ser escrito o dado a ser transferido. A cada transferência realizada, este registrador pode ser incrementado, decrementado ou permanecer estático.
- **Registrador Tamanho:** Este registrador contém a quantidade de vezes que o ciclo de DMA será executado. A cada transferência realizada, este registrador é decrementado.

Cada canal de DMA pode ter até 32 diferentes fontes de disparo. São 32 dispositivos que podem pedir para realizar DMA. Por disparo (*trigger* em inglês) se entende a solicitação para ser realizar uma transferência por DMA. O disparo corresponde ao DREQ (DMA *request*) das explicações apresentadas anteriormente. Para resolver os casos simultaneidade nos disparos, existe o Controle de Prioridade. Este controle pode ser fixo ou rotativo usando o algoritmo *Round Robin*.

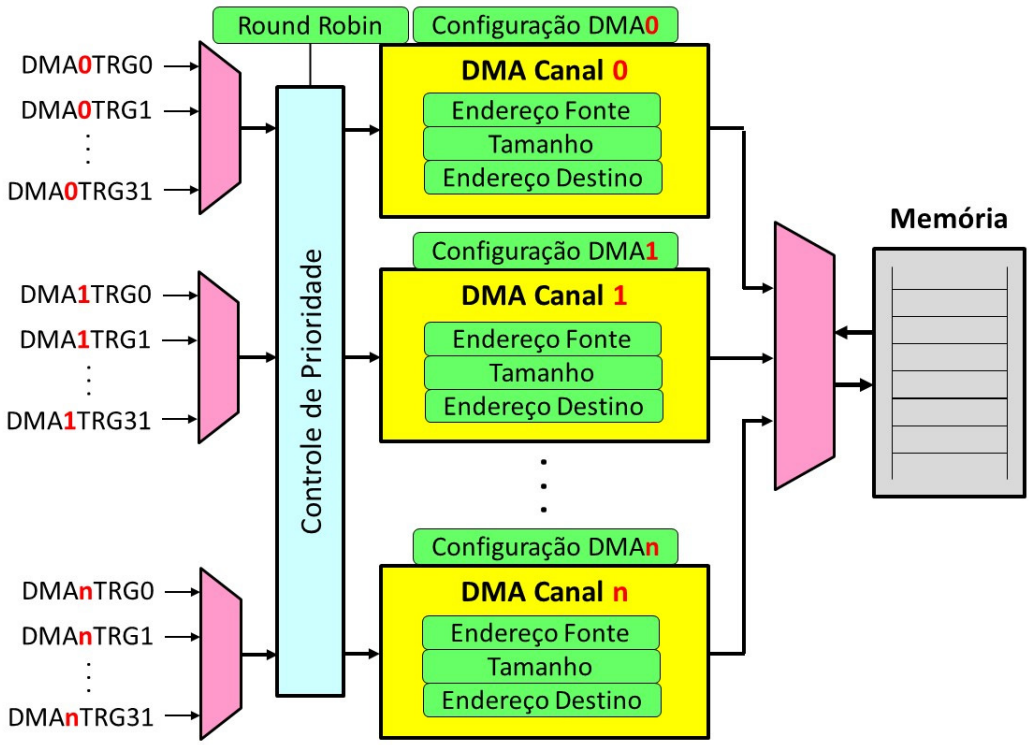


Figura 12.5. Diagrama de blocos do controlador de DMA do MSP430. Nesta figura, alguns detalhes foram omitidos para beneficiar a clareza.

A Figura 12.6 apresenta o diagrama de blocos detalhado com DMA, tomando como exemplo o Canal 1. Os retângulos verdes de cantos arredondados representam o parâmetro que são controlados pelo programador. Podemos ver os registradores de endereço fonte e destino e seus controles de incremento e tipo de transferência (byte ou word). O modo de operação do canal é controlado pelo registrador DMADT e a habilitação do canal é feita com o bit DMAEN. O multiplexador da extrema esquerda, controlado pelos bits DMA1TSEL, dão ao programador a liberdade de selecionar um dos 32 diferentes disparos.

Como se podem ter vários canais de DMA operando simultaneamente, o usuário pode trabalhar com a prioridade fixa ou rotativa ao habilitar o bit Round Robin. Para cada ciclo de DMA que vai acontecer, a CPU precisa ser colocada em HOLD, o que também permite que o controlador acesse a memória, como se pode ver o sinal na parte inferior da figura. Existe ainda uma lógica para compatibilizar o HOLD com a NMI e as operações da interface JTAG.

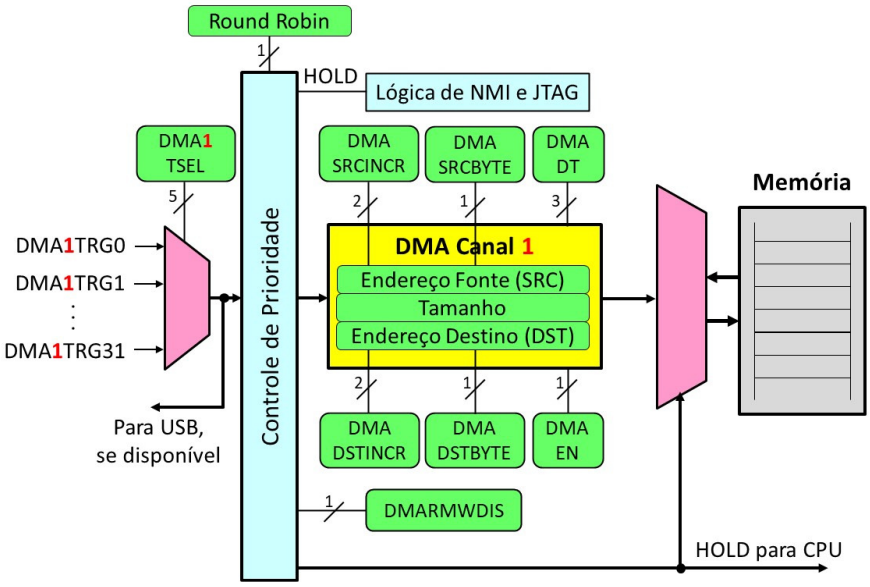


Figura 12.6. Diagrama de blocos detalhado, tomando como exemplo o Cana 1 de DMA.

Os três registradores listado abaixo são muito importantes para o entendimento do DMA. Cada canal possui esses três registradores e a forma como são controlados resulta em grande flexibilidade para a operação do DMA

- DMAnSA → registrador endereço fonte (SA abrevia *Source Address*);
- DMAnDA → registrador endereço destino (DA abrevia *Destination Address*);
- DMAnSZ → registrador total de transferências (SZ abrevia *Size*).

12.4. Modos de Endereçamento do Controlador de DMA

São quatro as opções para os Modos de Endereçamento do DMA e estão listados logo abaixo. A próxima figura apresenta a ilustração de cada um deles. Como os canais são independentes, o usuário pode programar o modo de endereçamento de cada canal de acordo com sua necessidade.

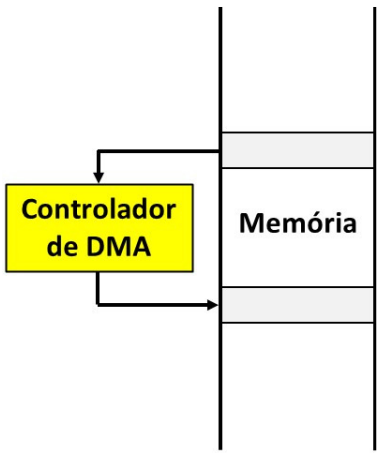
- De endereço fixo para endereço fixo;
- De endereço fixo para bloco de endereços;
- De bloco de endereços para endereço fixo e
- De bloco de endereços para bloco de endereços.

Como já vimos, cada canal de DMA possui um registrador de endereço fonte e um registrador de endereço destino. O usuário pode programar como esses registradores se comportam a cada transferência, ou seja, se são incrementados, decrementados ou se permanecem constantes. O controle é feito pela configuração dos bits indicados na tabela abaixo. Ao controlar o comportamento dos registradores de endereço após cada

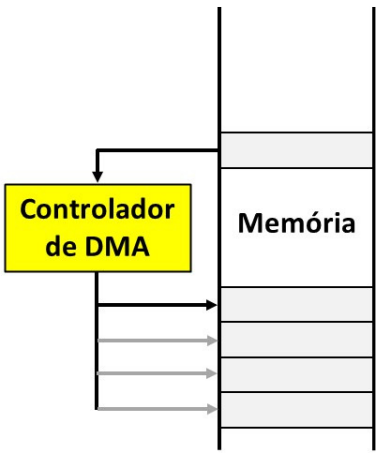
transferência por DMA, se conseguem as 4 opções recém-listadas e ilustradas na Figura 12.7.

Tabela 12.1. Indicação dos bits para selecionar o comportamento dos registradores de endereço após cada transferência por DMA.

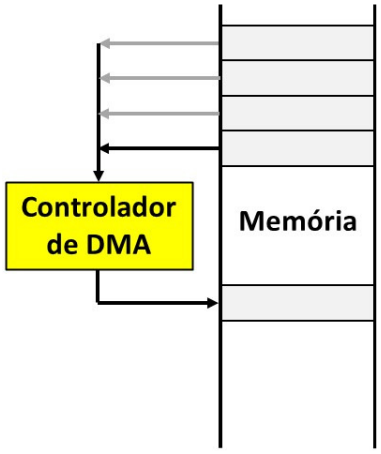
Bits DMASRCINCR (Endereço fonte)	Bits DMADSTINCR (Endereço destino)	Comportamento
0 ou 1	0 ou 1	Fixo
2	2	Decrementa
3	3	Incrementa



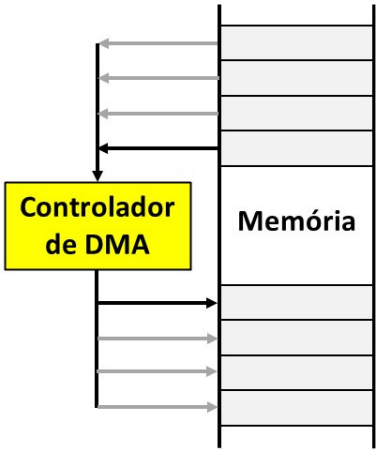
(a) De endereço fixo para endereço fixo.



(b) De endereço fixo para bloco de endereços.



(c) De Bloco de endereços para endereço fixo.



(d) De bloco de endereços para bloco de endereços.

Figura 12.7. Ilustração dos 4 modos de endereçamento do recurso de DMA do MSP430.

Como o controlador de DMA do MSP430 pode trabalhar com bytes ou words, temos as 4 opções de transferências, listadas abaixo.

- de byte para byte;
- de word para word;
- de byte para word (MSByte da word é zerado) e
- de word para byte (transfere apenas o LSByte da word)

Lembro-me de problemas na transferência de byte para byte. Parece que os bytes têm de estar em endereços pares. Conferir com mais cuidado.

12.5. Modos de Endereçamento do Controlador de DMA

Iniciamos apresentando na lista abaixo, os três modos de transferência por DMA. A distinção é feita pela forma de disparo e pela forma como se suspende as atividades da CPU (HOLD).

- **Transferência Simples** → É preciso de um disparo para cada dado transferido. O canal de DMA é desabilitado quando o contador (DMAxSZ) chega a zero. A cada transferência a atividade da CPU é suspensa.
- **Transferência em Bloco** → Um disparo dá início à transferência de dados por DMA que só para quando o contador (DMAxSZ) chega a zero e o canal de DMA é desabilitado. Durante todo este período, a CPU fica inativa.
- **Transferência em Rajadas** → Um disparo dá início à transferência de dados por DMA, porém há o intercalamento com a CPU não fica completamente inativa. Quando o contador (DMAxSZ) chega a zero o canal de DMA é desabilitado.

O leitor pode notar um dilema entre atividade de CPU e transferência por DMA. Por um lado, para transferir os dados necessários, precisamos suspender as atividades de CPU. Por outro lado, enquanto a CPU está suspensa, nenhum processamento é realizado. Podemos então fazer algumas recomendações. Sempre que possível, use o modo Transferência Simples. Neste caso os dados são transferidos e a suspensão da CPU fica distribuída ao longo de toda a transferência. Se os dados precisarem ser transferidos com grande velocidade, então é recomendado o modo Rajada, pois a CPU não é completamente suspensa e consegue fazer algum processamento. Em casos extremos, quando for imperativo uma alta velocidade na transferência dos dados, usamos o modo Transferência em Bloco e pagamos o preço de termos a CPU suspensa durante toda a transferência.

É comum a necessidade de se repetir uma transferência por DMA. Por isso, ao iniciar a operação de DMA, os registradores de endereço fonte, destino e contador são copiados para registradores temporários, o que possibilita a restauração dos valores originais.

- DMAnSA copiado para o registrador T_DMAnSA (temporário);
- DMAnDA copiado para o registrador T_DMAnDA (temporário);
- DMAnSZ copiado para o registrador T_DMAnSZ (temporário);

Agora podemos entender os 6 modos de operação de um canal de DMA, que o usuário pode selecionar com os bits de DMADT, como mostrado na Tabela 12.2.

Tabela 12.2. Modos de transferência por DMA.

DMADT	Modo	Descrição
0	Simples	Um disparo para cada transferência. DMA desabilitado se DMAnSZ =0.
1	Bloco	Um disparo faz a transferência de todos os dados. CPU permanece suspensa. DMA desabilitado se DMAnSZ =0.
2 ou 3	Rajada	Um disparo faz a transferência de todos os dados. É intercalado com atividade da CPU. DMA desabilitado se DMAnSZ =0.
4	Simples repetido	Um disparo para cada transferência. DMA permanece habilitado.
5	Bloco repetido	Um disparo faz a transferência de todos os dados. CPU permanece suspensa. DMA permanece habilitado.
6 ou 7	Rajada repetido	Um disparo faz a transferência de todos os dados. É intercalado com atividade da CPU. DMA permanece habilitado.

Vamos agora tentar formar uma ideia mais clara do DMA. Após o usuário decidir qual canal usar, digamos o canal n, a configuração deve realizar as seguintes etapas.

- Inicializar o registrador DMAnSA com o endereço de origem e indicar no campo DMASRCINCR seu comportamento (incrementa, decrementa ou fixo) a cada transferência. Especificar no campo DMASRSBYTE se a fonte tem o tamanho de byte ou word.
- Inicializar o registrador DMAnDA com o endereço de destino e indicar no campo DMADSTINCR seu comportamento (incrementa, decrementa ou fixo) a cada transferência. Especificar no campo DMADSTBYTE se o destino tem o tamanho de byte ou word.

- Inicializar o registrador DMAnSZ com o total de transferências a ser realizada.
- Indicar no campo DMAnTSEL a origem dos disparos do DMA.
- Selecionar no campo DMADT o modo de transferência a ser realizada por este canal.
- Habilitar o canal de DMA usando o bit DMAEN. Há uma habilitação para cada canal.

Quando o usuário habilita o DMA, os registradores DMAnSA, DMAnDA e DMAnSZ são copiados para os registradores temporários e, a partir de agora, o disparo pode dar início às transferências. A cada dado transferido, o contador DMAnSZ é decrementado. Se diz que a transferência solicitada termina quando esse contador chega a zero. Quando isto acontece, uma *flag* específica para o canal (DMAIFG) vai para nível alto e pode provocar interrupção. Se foi selecionado algum modo sem repetição, quando o contador DMAnSZ chega a zero, o canal de DMA é desabilitado (DMAEN = 0). Se foi selecionado algum modo com repetição, quando o contador DMAnSZ chega a zero, o canal de DMA permanece habilitado (DMAEN = 1), os valores são restaurados a partir dos registradores temporários e o próximo disparo repete as transferências.

É importante comentar como são usados os registradores temporários. Quando o controlador é habilitado (DMAEN = 1), são feitas as cópias listadas abaixo.

- DMAnSZ → T_DMAnSZ
- DMAnSA → T_DMAnSA
- DMAnDA → T_DMAnDA

Como mostrado na tabela abaixo, a cada transferência são alterados o registrador DMAnSZ e os temporários T_DMAnSA e T_DMAnDA. Isto significa que durante as transferências, as leituras dos registradores de endereços sempre retornam os valores originais. O programador não tem como ler o valor dos registradores temporários. Apenas o registrador DMAnSZ pode ser usado para monitorar o progresso das transferências por DMA. Por isso, ao terminar as operações esse registrador é restaurado a partir do temporário. Os registradores de endereços fazem o inverso.

Tabela 12.3. Emprego dos registradores temporários.

Transferências alteram	**Se DMAnSZ =0 sem repetição**	**Se DMAnSZ =0 com repetição**
DMAnSZ	T_DMAnSZ → DMAnSZ	T_DMAnSZ → DMAnSZ
T_DMAnSA	-	DMAnSA → T_DMAnSA
T_DMAnDA	-	DMAnDA → T_DMAnDA

12.5.1. Transferência Simples (com ou sem Repetição)

Este modo é o usado na grande maioria das situações. Para cada transferência é preciso de um disparo, portanto, adequado para transferir dados em uma taxa lenta. É a solução ideal para comunicação serial (UART, I^2C, SPI). A cada transferência a CPU entra em HOLD por 2 ciclos do relógio MCLK.

A Figura 12.8 apresenta o diagrama de estados para este modo. É importante lembrar que os canais de DMA são independentes, então este diagrama apresenta o comportamento de um canal. O diagrama está um pouco simplificado. Foram removidos transições e estados relativos a Abort e NMI (DMAABORT e ENNMI) que podem ser consultados no manual do fabricante.

Como pode ser notado nesta figura, enquanto o controlador de DMA não é habilitado (DMAEN = 0) o canal permanece no estado de *Reset*. Sob estas condições o programador deve configurar o canal que vai usar. Em especial ele inicializa os registradores DMAnSZ, DMAnSA e DMAnDA para indicar a quantidade de transferências, o endereço fonte e o endereço destino, respectivamente. Quando o canal é habilitado (DMAEN = 1), é feita uma cópia desses três registradores para os registradores temporários e o controlador vai para o estado onde fica esperando pelo disparo. A cada disparo, uma transferência é realizada, enquanto a CPU é mantida em *Hold* durante 2 períodos de MCLK. Ao terminar a transferência, o contador DMAnSZ é decrementado e os registradores temporários que guardam os endereços são modificados (incrementar, decrementar ou permanecer fixo) em função do que o usuário programou (Figura 12.7).

Em seguida, se DMAnSZ ainda não chegou a zero, o controlador volta para o estado em que fica esperando o próximo disparo. Quando DMAnSZ chega a zero, se foi programado o modo sem repetição (DMADT = 0), o controlador é desabilitado (DMAEN = 0), a cópia que estava em t_DMAnSZ é usada para restaurar o valor de DMAnSZ e a flag DMAIFG correspondente vai para 1. Porém, se foi programado o modo com repetição (DMADT = 4), então os valores dos três registradores envolvido são restaurados, a flag DMAIFG correspondente vai para 1 e tudo volta a se repetir. É recomendado nova consulta à Tabela 12.3.

Das 32 possibilidades de disparos, uma delas é pelo bit DMAREQ. Este é um disparo por software. Quando o usuário faz DMAREQ = 1, ele provoca um disparo no canal de DMA correspondente. Note no diagrama de estados, que este bit é zerado após a realização de cada transferência.

Quando o usuário desabilita o canal de DMA (DMAEN = 0), o controlador volta para o estado de Reset. O manual apresenta uma certa dualidade neste caso. Parece que se a desabilitação (DMAEN = 0) ocorre enquanto se espera pelo disparo, a unidade vai para o estado de Reset. Porém, se acontece durante uma transferência (como pode acontecer isso se a CPU está em Hold? Verificar.) temos a restauração do valor original do contador DMAnSZ. Ver no manual o ramo marcado com:

[DMADT = {0} AND DMAnSZ = 0] OR DMAEN =0

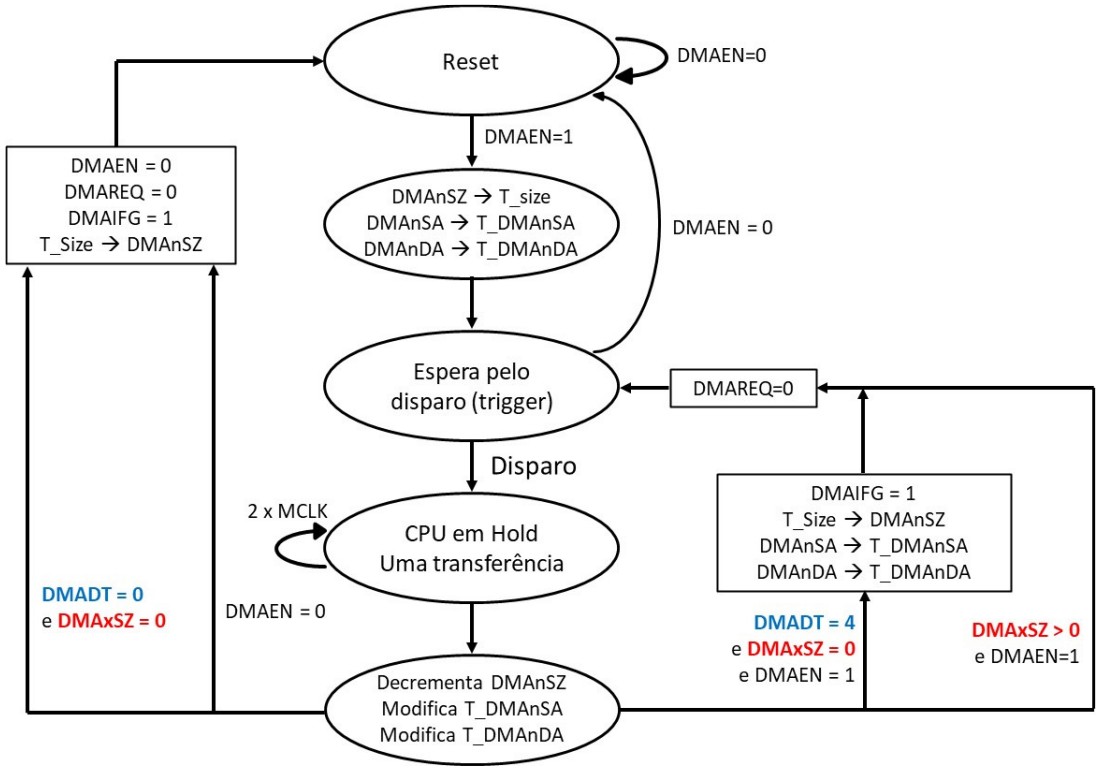


Figura 12.8. Diagrama de estados do modo de transferência Simples, com ou sem repetição. O diagrama está um pouco simplificado. Notar que acrescentei a flag DMAIFG e que removi a parte de DMAABORT e NMI.

12.5.2. Transferência em Bloco (com ou sem Repetição)

Este modo é o usado para transferir blocos de dados de uma forma rápida. O preço a ser pago é a suspensão das atividades da CPU. Como cada transferência por DMA consome 2 períodos de MCLK, o período em que a CPU vai permanecer suspensa é dada pela conta abaixo. Por exemplo, para se transferir 1.024 dados, considerando MCLK = 1.048.576 Hz, a CPU irá permanecer suspensa (em Hold) durante 1,95 ms, ou seja, inativa durante 2.048 períodos de MCLK.

$$T_{susp} = 2 \times DMAnSZ \times T_{MCLK}$$

A Figura 12.9 apresenta o diagrama de estados para este modo. É importante lembrar que os canais de DMA são independentes, então este diagrama apresenta o

comportamento de um canal. O diagrama está um pouco simplificado. Foram removidos transições e estados relativos a Abort e NMI (DMAABORT e ENNMI) que podem ser consultados no manual do fabricante.

Como pode ser notado nesta figura, enquanto o controlador de DMA não é habilitado (DMAEN = 0) o canal permanece no estado de Reset. Sob estas condições o programador deve configurar o canal que vai usar. Em especial ele inicializa os registradores DMAnSZ, DMAnSA e DMAnDA para indicar a quantidade de transferências, o endereço fonte e o endereço destino, respectivamente. Quando o canal é habilitado (DMAEN = 1), é feita uma cópia desses três registradores para os registradores temporários e o controlador vai para o estado onde fica esperando pelo disparo. Quando ocorre o disparo, a CPU é colocada em Hold e as transferências são iniciadas. A cada transferência, o contador DMAnSZ é decrementado e os registradores temporários que guardam os endereços são modificados (incremento, decremento ou fixo) em função do que o usuário programou (Figura 12.7). Somente quando o contador (DMAnSZ) chega a zero as transferências terminam e o controle é devolvido à CPU.

Quando DMAnSZ chega a zero, se foi programado o modo sem repetição (DMADT = 1), o controlador é desabilitado (DMAEN = 0), a cópia que estava em t_DMAnSZ é usada para restaurar o valor de DMAnSZ e a flag DMAIFG correspondente vai para 1. Porém, se foi programado o modo com repetição (DMADT = 5), então os valores dos três registradores envolvido são restaurados, a flag DMAIFG correspondente vai para 1 e o controlador volta para o estado em que espera pelo disparo. É recomendado nova consulta à Tabela 12.3. Se o disparo foi feito por software, ou seja, pelo bit DMAREQ, este bit é zerado ao término das transferências.

Quando o usuário desabilita o canal de DMA (DMAEN = 0), o controlador volta para o estado de Reset. Novamente, o manual apresenta uma certa dualidade neste caso. Parece que se a desabilitação (DMAEN = 0) ocorre enquanto se espera pelo disparo, a unidade vai para o estado de Reset. Porém, se acontece durante uma transferência (como pode acontecer isso se a CPU está em Hold? Verificar.) temos a restauração do valor original do contador DMAnSZ. Ver no manual o ramo marcado com:
[DMADT = {1} AND DMAnSZ = 0] OR DMAEN =0

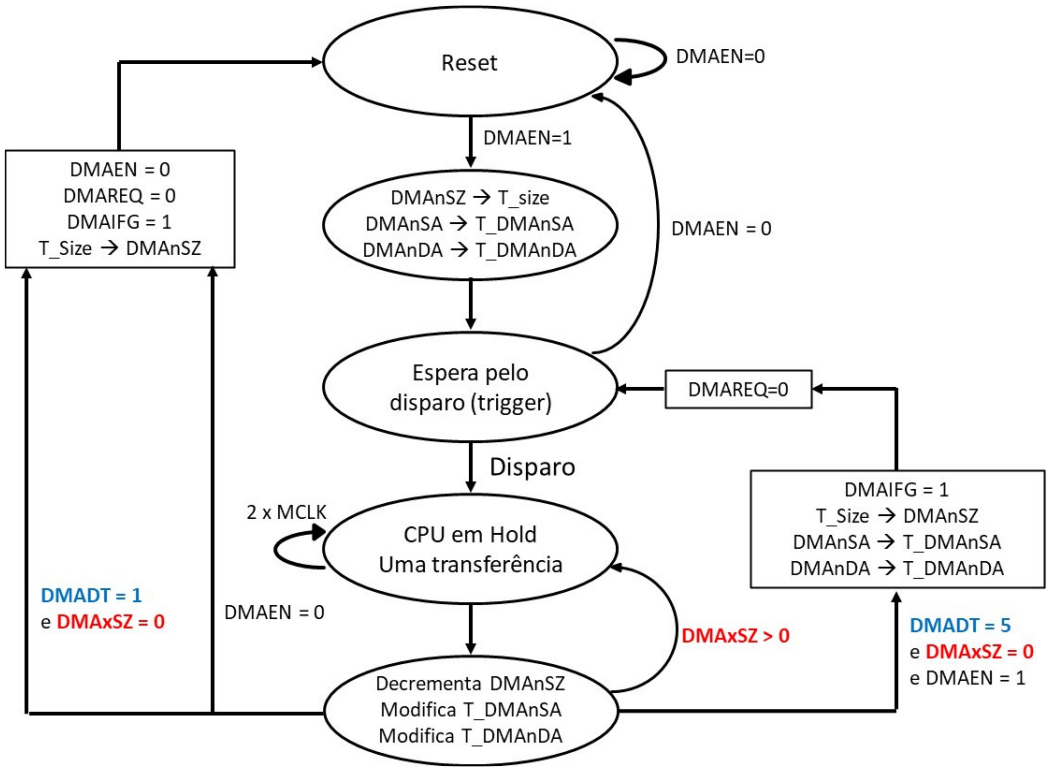


Figura 12.9. Diagrama de estados do modo de transferência em Bloco, com ou sem repetição. O diagrama está um pouco simplificado. Notar que acrescentei a flag DMAIFG e que removi a parte de DMAABORT e NMI.

12.5.3. Transferência em Rajadas (com ou sem Repetição)

Este modo é o usado para transferir blocos de dados de uma forma relativamente rápida, porém sem suspender completamente a atividade da CPU. A cada 4 transferências (8 x MCLK) por DMA, o barramento é liberado para a CPU por 2 períodos de MCLK. Então, a cada 10 períodos de MCLK, 8 são dedicados ao DMA e 2 à CPU, como mostrado na Figura 12.10. Por exemplo, para ser transferir 1.024 dados por DMA, considerando MCLK em 1.048.576 Hz, teremos:

- Duração da transferência = 1,22 ms (1.280 períodos de MCLK)
- CPU operando = 0,244 ms (256 períodos de MCLK)

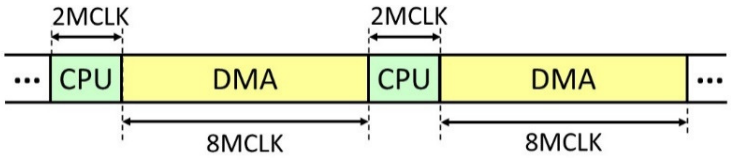


Figura 12.10. Distribuição do barramento entre a CPU e o DMA quando se seleciona o modo Rajada, com ou sem repetição.

A Figura 12.11 apresenta o diagrama de estados para este modo. É importante lembrar que os canais de DMA são independentes, então este diagrama apresenta o comportamento de um canal. O diagrama está um pouco simplificado. Foram removidos transições e estados relativos a Abort e NMI (DMAABORT e ENNMI) que podem ser consultados no manual do fabricante.

Como pode ser notado nesta figura, enquanto o controlador de DMA não é habilitado (DMAEN = 0) o canal permanece no estado de Reset. Sob estas condições o programador deve configurar o canal que vai usar. Em especial ele inicializa os registradores DMAnSZ, DMAnSA e DMAnDA para indicar a quantidade de transferências, o endereço fonte e o endereço destino, respectivamente. Quando o canal é habilitado (DMAEN = 1), é feita uma cópia desses três registradores para os registradores temporários e o controlador vai para o estado onde fica esperando pelo disparo. Quando ocorre o disparo, a CPU é colocada em Hold e as transferências são iniciadas. A cada transferência, o contador DMAnSZ é decrementado e os registradores temporários que guardam os endereços são modificados (incremento, decremento ou fixo) em função do que o usuário programou (Figura 12.7). A cada 4 transferências, a CPU volta a operar por 2 MCLK. Somente quando o contador (DMAnSZ) chega a zero as transferências terminam e o controle é plenamente devolvido à CPU.

Quando DMAnSZ chega a zero, se foi programado o modo sem repetição (DMADT = 2 ou 3), o controlador é desabilitado (DMAEN = 0), a cópia que estava em t_DMAnSZ é usada para restaurar o valor de DMAnSZ e a flag DMAIFG correspondente vai para 1. Porém, se foi programado o modo com repetição (DMADT = 6 ou 7), então os valores dos três registradores envolvido são restaurados, a flag DMAIFG correspondente vai para 1 e o controlador volta para o estado em que espera pelo disparo. É recomendado nova consulta à Tabela 12.3. Se o disparo foi feito por software, ou seja, pelo bit DMAREQ, este bit é zerado ao término das transferências.

Quando o usuário desabilita o canal de DMA (DMAEN = 0), o controlador volta para o estado de Reset. Novamente, o manual apresenta uma certa dualidade neste caso. Parece que se a desabilitação (DMAEN = 0) ocorre enquanto se espera pelo disparo, a unidade vai para o estado de Reset. Porém, se acontece durante uma transferência (como pode acontecer isso se a CPU está em Hold? Verificar.) temos a restauração do valor original do contador DMAnSZ. Ver no manual o ramo marcado com:
[DMADT = {2, 3} AND DMAnSZ = 0] OR DMAEN =0

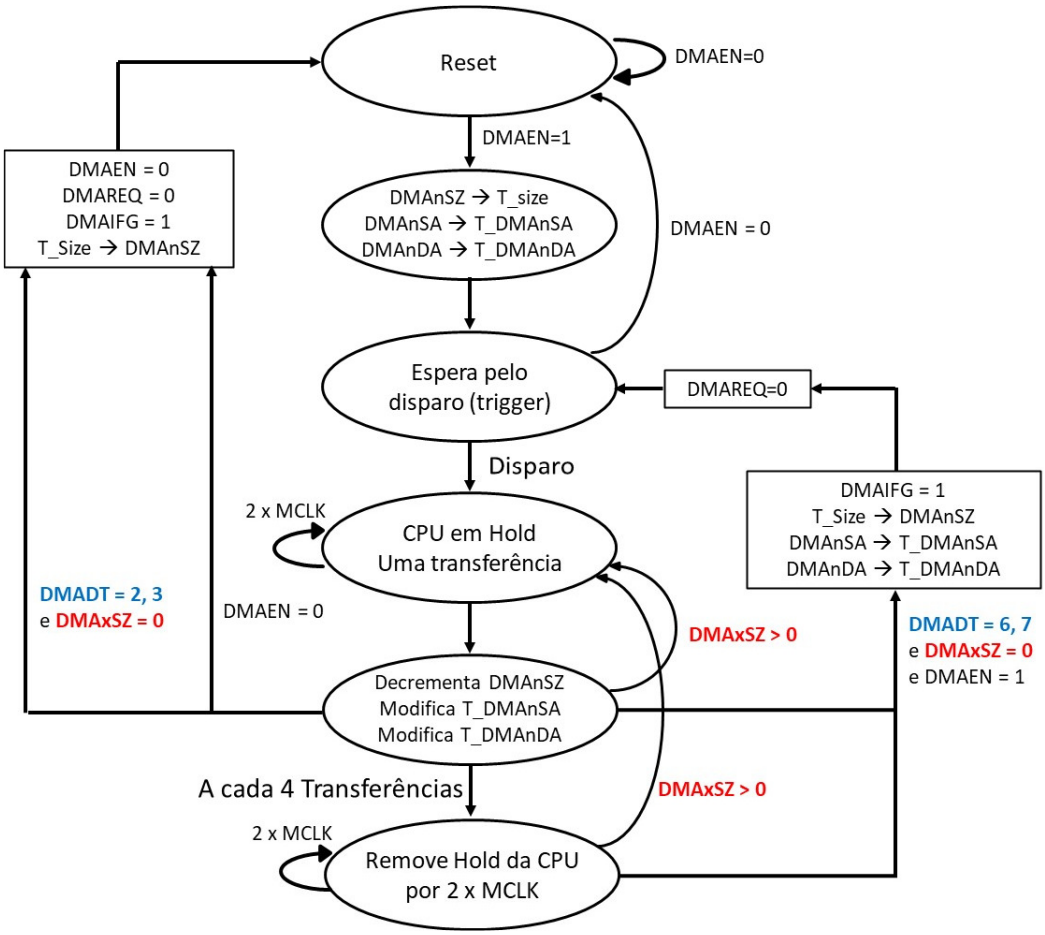


Figura 12.11. Diagrama de estados do modo de transferência em Rajadas, com ou sem repetição. O diagrama está um pouco simplificado. Notar que acrescentei a flag DMAIFG e que removi a parte de DMAABORT e NMI.

12.6. Disparo das Operações de DMA

Os disparos das operações de DMA podem ser por flanco ou por nível. No caso do MSP430 a quase totalidade das operações de DMA são disparadas por flanco de subida, que acontece quando um determinado bit muda de 0 para 1. A opção de disparo por flanco é selecionada quando o bit DMALEVEL está em 0.

O disparo por nível, selecionado com DMALEVEL em 1 só é possível pela entrada externa DMAE0 (P2.6). Neste caso (por nível), as transferências por DMA acontecem enquanto o pino está em nível alto e o DMA permanece habilitado (DMAEN = 1). O sinal de disparo deve permanecer em nível alto para que as transferências em modo bloco ou rajada sejam completadas. Se o pino for colocado em nível baixo, a transferência entra

em estado suspenso. Quando o pino volta para nível alto, ela continua do ponto em que parou.

Ao trabalhar com disparo por nível, é recomendado que se use os modos sem repetição, isto por que a habilitação do DMA é zerada (DMAEN = 0) ao completar a operação. Esta preocupação é porque com os modos com repetição, há o risco de que a CPU fique em *Hold* por um tempo muito longo. Imagine o caso do modo bloco com repetição com um defeito "prendendo" o pino DMAE0 em nível alto.

As transferências por DMA acontecem sem a intervenção da CPU e são independentes dos modos de baixo consumo.

Na arquitetura MSP430 temos 32 fontes possíveis para os disparos das transferências por DMA. A tabela abaixo apresenta o caso particular da versão MSP430 F5529. Logo a seguir é feito um estudo de cada uma das possibilidades de disparo.

Tabela 12.4. Opções de disparo para os canais de DMA (versão MSP430 F5529), veja a continuação.

Disparo	Canal 0	Canal 1	Canal 2
0	DMAREQ	DMAREQ	DMAREQ
1	TA0CCR0 CCIFG	TA0CCR0 CCIFG	TA0CCR0 CCIFG
2	TA0CCR2 CCIFG	TA0CCR2 CCIFG	TA0CCR2 CCIFG
3	TA1CCR0 CCIFG	TA1CCR0 CCIFG	TA1CCR0 CCIFG
4	TA1CCR2 CCIFG	TA1CCR2 CCIFG	TA1CCR2 CCIFG
5	TA2CCR0 CCIFG	TA2CCR0 CCIFG	TA2CCR0 CCIFG
6	TA2CCR2 CCIFG	TA2CCR2 CCIFG	TA2CCR2 CCIFG
7	TB0CCR0 CCIFG	TB0CCR0 CCIFG	TB0CCR0 CCIFG
8	TB0CCR2 CCIFG	TB0CCR2 CCIFG	TB0CCR2 CCIFG
9, ..., 15	Reservado	Reservado	Reservado

Tabela 12.4. Continuação das Opções de disparo para os canais de DMA (versão MSP430 F5529)

Disparo	Canal 0	Canal 1	Canal 2
16	UCA0RXIFG	UCA0RXIFG	UCA0RXIFG
17	UCA0TXIFG	UCA0TXIFG	UCA0TXIFG
18	UCB0RXIFG	UCB0RXIFG	UCB0RXIFG
19	UCB0TXIFG	UCB0TXIFG	UCB0TXIFG

20	UCA1RXIFG	UCA1RXIFG	UCA1RXIFG
21	UCA1TXIFG	UCA1TXIFG	UCA1TXIFG
22	UCB1RXIFG	UCB1RXIFG	UCB1RXIFG
23	UCB1TXIFG	UCB1TXIFG	UCB1TXIFG
24	ADC12IFG	ADC12IFG	ADC12IFG
25, 26	Reservado	Reservado	Reservado
27	USB FNRXD	USB FNRXD	USB FNRXD
28	USB Ready	USB Ready	USB Ready
29	MPY Ready	MPY Ready	MPY Ready
30	DMA2IFG	DMA0IFG	DMA1IFG
31	DMAE0	DMAE0	DMAE0

Disparo pelos Timers (TA0, TA1, TA2 e TB0)

Como pode ser vista na tabela acima, são 8 as possibilidades de disparo pelos *timers*, como listado abaixo:

- TA0CCR0.CCIFG e TA0CCR2.CCIFG;
- TA1CCR0.CCIFG e TA1CCR2.CCIFG;
- TA2CCR0.CCIFG e TA2CCR2.CCIFG;
- TB0CCR0.CCIFG e TB0CCR2.CCIFG;

A transferência é disparada quando o bit CCIFG vai para 1. Essa flag é automaticamente apagada após o início da transferência. Se a correspondente interrupção estiver habilitada (CCIE = 1), então não acontece o disparo do DMA.

Disparo pelas unidades seriais (USCI_Ax e USCI_Bx)

As instâncias das duas unidades seriais podem disparar uma transferência por DMA. O funcionamento é o mesmo, independente de operarem no modo UART, I^2C ou SPI. As explicações abordam as quatro unidades, mas é claro que elas operam de forma independente.

Uma transferência é disparada quando a USCI_Ax ou a USCI_Bx recebe um novo dado. A flag UCAxRXIFG ou UCBxRXIFG é automaticamente zerada quando a transferência é iniciada. Se a interrupção estiver habilitada, ou seja, UCAxRXIE = 1 ou UCBxRXIE = 1, a *flag* UCAxRXIFG ou UCBxRXIFG não dispara a transferência por DMA.

O mesmo acontece com a transmissão de um dado e a *flag* UCAxTCIFG ou UCBxTXIFG.

Disparo pelo Conversor ADC12

O disparo pode ser realizado por uma das flags ADC12IFG, desde que a interrupção correspondente esteja desabilitada. Na conversão de um único canal, a correspondente ADC12IFG faz o disparo da transferência. Quando se usa sequência de canais, o ADC12IFG da última conversão é que realiza o disparo. Em outras palavras, uma transferência é disparada quando a conversão está completa e a flag ADC12IFG é ativada. A ativação de ADC12IFG por software não dispara transferência por DMA. As flags ADC12IFG são automaticamente zeradas à medida que as memórias (ADC12MEMx) são lidas pelo controlador de DMA.

Disparo pelo Multiplicador em Hardware (MPY)

Uma transferência é disparada quando o multiplicador está pronto para receber um novo operando.

Disparos pelo DMA

Aqui temos três possibilidades de disparo:

- Pela ativação do bit DMAREQ (há um bit por canal);
- Pela ativação da flag DMAxIFG e
- Pelo pino DMAE0 (pino P2.6).

Uma transferência pode ser disparada por software com o uso do bit DMAREQ, sendo que cada canal tem seu próprio DMAREQ. O disparo acontece quando se ativa esse bit, que é automaticamente zerado quando se inicia a transferência.

A flag DMAxIFG que vai a 1 quando uma operação de DMA é finalizada pode ser usada para disparar uma nova transferência em um outro canal. A Figura 12.12 apresenta de forma clara como essa flag dispara o canal seguinte. A flag não é zerada por ocasião do início da transferência.

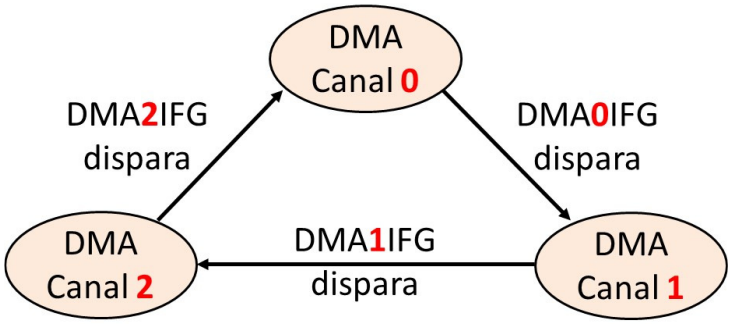


Figura 12.12. Esquema dos disparos entre os canais de DMA. A flag DMAxIFG vai para 1 quando a operação programada no canal é finalizada.

Ainda é possível disparar DMA externamente com o uso do pino DMAE0 (P2.6). Este pino é a única possibilidade de se disparar uma transferência por nível, como foi visto no início deste tópico.

Disparos e Instruções do Tipo Lê-Modifica-Escreve

Instruções do tipo Lê-Modifica-Escreve (sigla RMW do inglês *Read-Modify-Write*) são aquelas que acessam um valor na memória, o modificam e o escrevem de volta na memória. Temos, então, dois acessos à memória. Para que fique bem claro, vamos usar exemplos em *assembly*.

Por exemplo, a instrução `BIS.B #BIT5,&P1OUT` envolve a leitura do registrador (mapeado em memória) P1OUT, sua operação OR com o byte 0x20 e a escrita de volta no registrador P1OUT. A mesma coisa acontece com um incremento, por exemplo, `INC &0x2400`. Há uma grande quantidade de instruções deste tipo e elas são executadas com o uso de diversos ciclos de barramento.

Há o risco de uma operação de DMA ser realizada entre os dois ciclos de barramento de uma instrução deste tipo. Por exemplo, durante a execução da instrução `INC &0x2400`, pode acontecer que após a leitura da posição 0x2400 ocorra uma transferência por DMA que altere a posição 0x2400 e logo em seguida a CPU sobrescreve esta posição com o incremento. Provavelmente, não era esta a intenção do programador. É possível imaginar uma boa quantidade de situações curiosas envolvendo o DMA e instruções do tipo RMW.

Em geral, não precisamos nos preocupar com isso, entretanto, para casos mais específicas, o controlador de DMA oferece ao programador o bit DMARMWDIS.

- DMARMWDIS = 0 → a CPU entra em *Hold* assim que ocorre o disparo da transferência por DMA, e por isso, pode ser que a transferência ocorra durante a execução de uma instrução do tipo lê-modifica-escreve.
- DMARMWDIS = 1 → a CPU, se for o caso, termina a execução da instrução lê-modifica-escreve antes de entrar em *Hold* e permitir a transferência por DMA, ou

seja, as transferências nunca acontecem durante a execução de instruções do tipo RMW.

Prioridade entre os Canais de DMA

Para revolver os casos de dois ou mais disparos simultâneos, o controlador de DMA tem uma prioridade entre os canais, sendo o DMA0 o mais prioritário e o DMA7 o de menor prioridade. Assim, em caso de simultaneidade, o canal de DMA de maior prioridade completa sua transferência, independentemente do modo (simples, bloco ou rajada) selecionado, e somente depois o controlador aceita o canal com a segunda maior prioridade.

O usuário pode escolher entre esta prioridade fixa ou a prioridade rotativa usando o algoritmo Round Robin. Nesse caso, a ordem das prioridades, DMA0 – DMA1 – DMA2 permanece a mesma, porém ela é girada de forma a que o canal que terminou a transferência vá para a última posição. A Figura 12.13 apresenta uma ilustração do funcionamento do algoritmo, onde o sinal "+" indica o canal de maior prioridade no momento, o sinal "-" o de menor prioridade e "Disp" sinaliza o canal que disparou uma transferência por DMA.

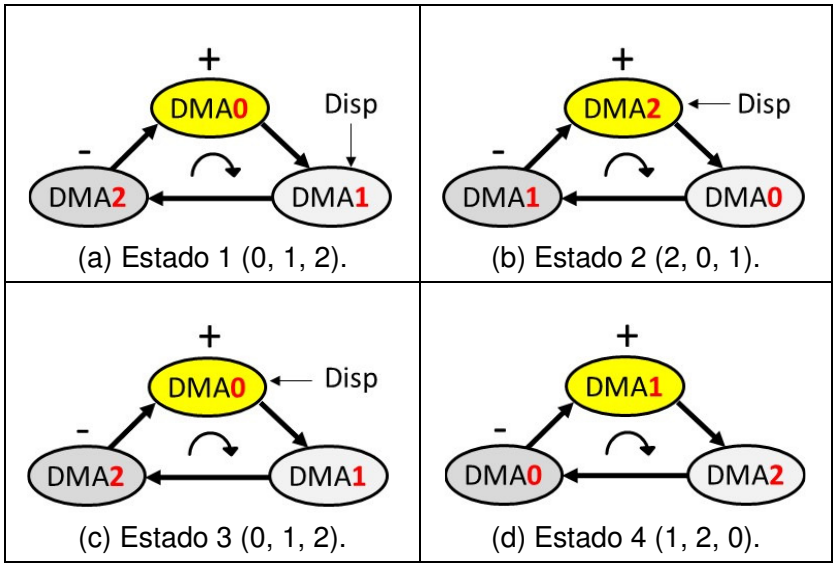


Figura 12.13. Ilustração do algoritmo Roud Robin para alternar a prioridade entre os canais de DMA. O sinal "+" indica o canal mais prioritário no momento e, é claro, o "-" o de menor prioridade.

Interrupções e a Operação de DMA

As interrupções comuns não interrompem as transferências por DMA. As interrupções ficam pendentes até que a transferência seja completada. Somente as interrupções do tipo NMI podem interromper operações de DMA. Para que isto aconteça, o usuário deve habilitar o bit ENNMI.

Por outro lado, as rotinas que atendem às interrupções podem ser interrompidas pelas transferências de DMA. Caso o programador precise que um trecho de código não seja interrompido pelo DMA, ele deve previamente desabiltar o controlador de DMA (DMAEN = 0).

Cada canal de DMA possui sua própria flag para gerar interrupção (DMAIFG). Esta flag é ativada quando o respectivo contador de transferências (DMAxSZ) chega a zero. Para que a interrupção ocorra, a flag DMAIE deve estar em 1. O registrador de vetor de interrupção (DMAIV) deve ser consultado para ver qual das possíveis flags provocou a interrupção. O funcionamento do registrador DMAIV é semelhante ao que já foi visto para os demais recursos e não será repetido aqui.

12.7. Registradores do Controlador de DMA

Nesta seção são descritos os registradores disponíveis para a operação do Controlador de DMA. A Tabela 12.5 apresenta uma listagem completa de todos esses registradores. Na versão F5529 temos apenas 3 canais de DMA: DMA0, DMA1 e DMA2, por isso a tabela ressalta apenas os registradores desses canais. A arquitetura MSP430 pode chegar a 8 canais de DMA e a lógica para nomeação de seus registradores é a mesma.

É de se notar que agora temos alguns registradores de 32 bits. Na verdade, estes registradores disponibilizam ao programador apenas 20 bits para permitir DMA em endereços acima de 64 K.

Tabela 12.5. Registradores do Controlador de DMA

32 bits	16 bits	Acesso	Reset	Ordem
-	DMACTL0	R/W	0	MSB
	DMACTL1	R/W	0	MSB
-	DMACTL2	R/W	0	MSB
-	DMACTL3	R/W	0	MSB
-	DMACTL4	R/W	0	MSB
-	DMAIV	R/W	0	MSB
-	DMA0CTL	R/W	0	MSB

DMA0SA	DMA0SA	R/W	?	MSB
DMA0DA	DMA0DA	R/W	?	MSB
-	DMA0SZ	R/W	?	LSB
-	DMA1CTL	R/W	0	MSB
DMA1SA	DMA1SA	R/W	?	MSB
DMA1DA	DMA1DA	R/W	?	MSB
-	DMA1SZ	R/W	?	LSB
-	DMA2CTL	R/W	0	MSB
DMA2SA	DMA2SA	R/W	?	MSB
DMA2DA	DMA2DA	R/W	?	MSB
-	DMA2SZ	R/W	?	LSB
	...	...	...	...

? = valor indefinido

12.8.1. DMACTL0 – Registrador 0 de Controle do DMA *(DMA Control 0 Register)*

Este registrador permite selecionar a origem do disparo para as operações de DMA, dos canais 0 e 1.

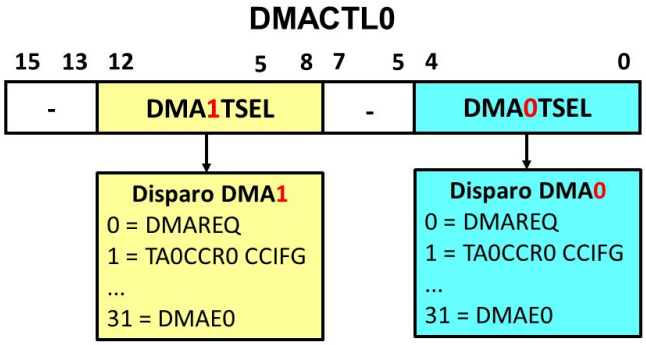


Figura 12.14. Registrador 0 de controle do DMA. Especifica a origem do disparo para os canais 0 e 1.

(R) *Bits* 15, ..., 13: Reservados
Esses bits são reservados e sua leitura sempre retorna zero.

(R/W) *Bit* 12, ..., 8: DMA1TSEL – Seleção do Disparo para o Canal 1 (*DMA1 trigger select*)

Esses bis permitem a seleção da origem do disparo para as transferências por DMA para o canal 1. Veja detalhes na Tabela 12.4.

(R) *Bits* 7, ..., 5: Reservados
Esses bits são reservados e sua leitura sempre retorna zero.

(R/W) *Bit* 4, ..., 0: DMA0TSEL – Seleção do Disparo para o Canal 0 (*DMA0 trigger select*)
Esses bis permitem a seleção da origem do disparo para as transferências por DMA para o canal 0. Veja detalhes na Tabela 12.4.

12.8.2. DMACTL1 – Registrador 1 de Controle do DMA *(DMA Control 1 Register)*

Este registrador permite selecionar a origem do disparo para as operações de DMA, dos canais 2 e 3. Vale lembrar que a versão F5529 do MSP430 só tem os canais 0, 1 e 2.

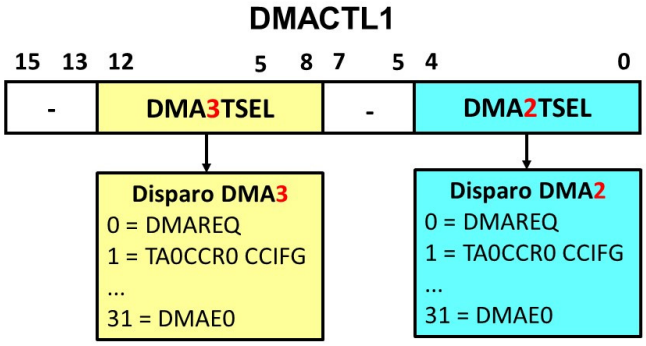


Figura 12.15. Registrador 1 de controle do DMA. Especifica a origem do disparo para os canais 2 e 3.

(R) *Bits* 15, ..., 13: Reservados
Esses bits são reservados e sua leitura sempre retorna zero.

(R/W) *Bit* 12, ..., 8: DMA3TSEL – Seleção do Disparo para o Canal 3 (*DMA3 trigger select*)
Esses bis permitem a seleção da origem do disparo para as transferências por DMA para o canal 3, que não está disponível na vesão F5529. Veja detalhes na Tabela 12.4.

(R) *Bits* 7, ..., 5: Reservados
Esses bits são reservados e sua leitura sempre retorna zero.

(R/W) *Bit* 4, ..., 0: DMA2TSEL – Seleção do Disparo para o Canal 2 (*DMA2 trigger select*)
Esses bis permitem a seleção da origem do disparo para as transferências por DMA para o canal 2. Veja detalhes na Tabela 12.4.

12.8.3. DMACTL2 – Registrador 2 de Controle do DMA *(DMA Control 2 Register)*

Este registrador permite selecionar a origem do disparo para as operações de DMA, dos canais 4 e 5, que não estão disponíveis na versão F5529 do MSP430.

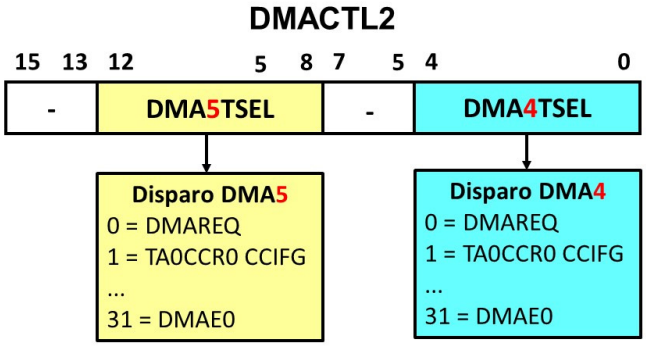


Figura 12.16. Registrador 2 de controle do DMA. Especifica a origem do disparo para os canais 4 e 5.

(R) *Bits* 15, ..., 13: Reservados
Esses bits são reservados e sua leitura sempre retorna zero.

(R/W) *Bit* 12, ..., 8: DMA5TSEL – Seleção do Disparo para o Canal 5 (*DMA5 trigger select*)
Esses bits permitem a seleção da origem do disparo para as transferências por DMA para o canal 5, que não está disponível na versão F5529. Veja detalhes na Tabela 12.4.

(R) *Bits* 7, ..., 5: Reservados
Esses bits são reservados e sua leitura sempre retorna zero.

(R/W) *Bit* 4, ..., 0: DMA4TSEL – Seleção do Disparo para o Canal 4 (*DMA4 trigger select*)
Esses bits permitem a seleção da origem do disparo para as transferências por DMA para o canal 4, que não está disponível na versão F5529. Veja detalhes na Tabela 12.4.

12.8.4. DMACTL3 – Registrador 3 de Controle do DMA *(DMA Control 3 Register)*

Este registrador permite selecionar a origem do disparo para as operações de DMA, dos canais 6 e 7, que não estão disponíveis na versão F5529 do MSP430.

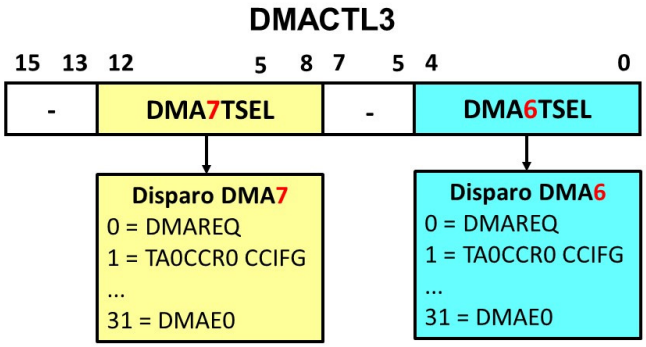


Figura 12.17. Registrador 3 de controle do DMA. Especifica a origem do disparo para os canais 6 e 7.

(R) *Bits* 15, ..., 13: Reservados
Esses bits são reservados e sua leitura sempre retorna zero.

(R/W) *Bit* 12, ..., 8: DMA5TSEL – Seleção do Disparo para o Canal 7 (*DMA5 trigger select*)
Esses bits permitem a seleção da origem do disparo para as transferências por DMA para o canal 7, que não está disponível na versão F5529. Veja detalhes na Tabela 12.4.

(R) *Bits* 7, ..., 5: Reservados
Esses bits são reservados e sua leitura sempre retorna zero.

(R/W) *Bit* 4, ..., 0: DMA4TSEL – Seleção do Disparo para o Canal 6 (*DMA4 trigger select*)
Esses bits permitem a seleção da origem do disparo para as transferências por DMA para o canal 6, que não está disponível na versão F5529. Veja detalhes na Tabela 12.4.

12.8.5. DMACTL4 – Registrador 4 de Controle do DMA *(DMA Control 4 Register)*

Este registrador permite um controle sobre o controlador de DMA especificando se as interrupções NMI podem interromper operações de DMA, qual prioridade a ser usada e o comportamento das operações durante a execução de instruções do tipo lê-modifica-escreve (RMW).

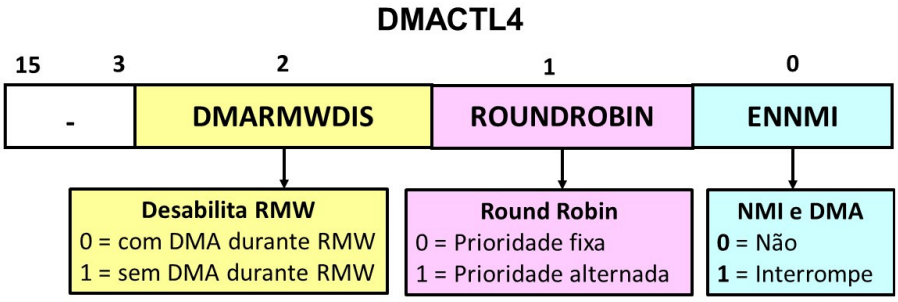


Figura 12.18. Detalhes do Registrador 4 de controle do DMA.

(R) *Bits* 15, ..., 3: Reservados
Esses bits são reservados e sua leitura sempre retorna zero.

(R/W) *Bit* 2: DMARMWDIS – Desabilita Lê-Modifica-Escreve
(*Read-modify-write disable*)
Quando este bit é programado em 1, as operações de DMA não acontecem durante a execução das instruções do tipo RMW.
DMARMWDIS = 0 → DMA pode interromper instruções do tipo RMW.
DMARMWDIS = 1 → DMA não interrompe instruções do tipo RMW.

(R/W) *Bit* 1: ROUNDROBIN – Habilita Prioridade Alternada
(*Round robin*)
Permite que a prioridade entre os canais seja alternada segundo o algoritmo Round Robin.
ROUNDROBIN = 0 → A prioridade é DMA0, DMA1, ..., DMA7.
ROUNDROBIN = 1 → A prioridade é alterada a cada transferência, vide Figura 12.13.

(R/W) *Bit* 0: ENNMI – Habilita Interrupção NMI
(*NMI enable*)
Permite que a NMI interrompa a transferência por DMA. Quando a NMI ocorre, a transferência que estiver em curso é terminada normalmente, porém as próximas não ocorrem. O bit DMABORT é ativado para indicar a ocorrência. Esta opção é muito útil nas transferências no modo Bloco.
ENNMI = 0 → NMI não interrompe uma transferência por DMA.
ENNMI = 1 → NMI pode interromper uma transferência por DMA.

12.8.6. DMAxCTL – Registrador de Controle para o Canal x
(DMA Channel x Control Register)

Existe um registrador deste tipo para cada canal de DMA. Ele permite programar todas as particularidades do canal. Em especial citamos os modos de operação e o incremento ou decremento dos registradores de endereço.

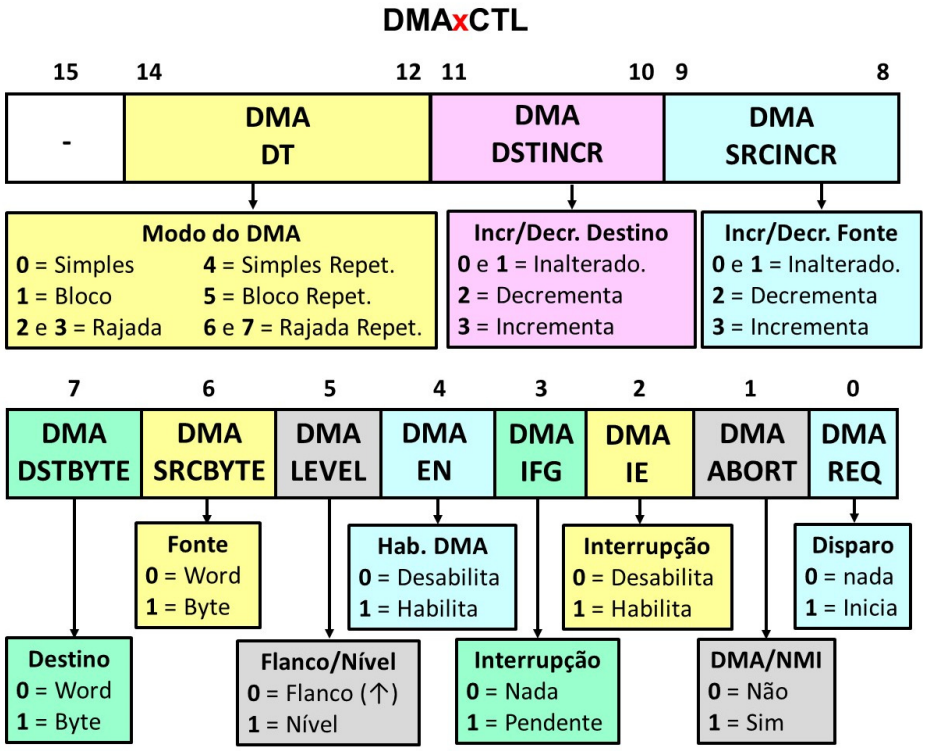


Figura 12.19. Detalhes do Registrador de Controle do Canal x. Existe um registrador deste tipo para cada canal de DMA.

(R) *Bit* 15: Reservado
Este bit é reservado e sua leitura sempre retorna zero.

(R/W) *Bit* 14, ..., 12: DMADT – Modo de Operação do DMA
(*DMA transfer mode*)
Esses 3 bits permitem ao usuário escolher como quer operar o canal de DMA.

DMADT = 0 → Modo Simples	DMADT = 4 → Modo Simples Repetido
DMADT = 1 → Modo Bloco	DMADT = 5 → Modo Bloco Repetido
DMADT = 2 ou 3 → Modo Rajada	DMADT = 6 ou 7 → Modo Rajada Repetida

(R/W) *Bit* 11, 10: DMADSTINCR – Incremento do Endereço de Destino
(*DMA destination increment*)
Esses bits permitem que o programador especifique, após cada transferência, como será alterado o endereço de destino. As opções são de incremento, decremento ou inalterado. O incremento/decremento será de uma ou duas posições de acordo com a seleção de

byte ou word feita em DMADSTBYTE. Vale lembrar que o registrador DMAxDA é copiado para um registrador temporário, que é de fato alterado, ou seja, o registrador DMAxDA permanece inalterado durante toda a operação de DMA.
DMADSTINCR = 0 ou 1 → Inalterado
DMADSTINCR = 2 → Decremento
DMADSTINCR = 3 → Incremento

(R/W) *Bit* 9, 8: DMASRINCR – Incremento do Endereço de Origem
(*DMA source increment*)
Esses bits permitem que o programador especifique, após cada transferência, como será alterado o endereço de origem. As opções são de incremento, decremento ou inalterado. O incremento/decremento será de uma ou duas posições de acordo com a seleção de byte ou word feita em DMASRCBYTE. Vale lembrar que o registrador DMAxSA é copiado para um registrador temporário, que é de fato alterado, ou seja, o registrador DMAxSA permanece inalterado durante toda a operação de DMA.
DMASRCINCR = 0 ou 1 → Inalterado
DMASRCINCR = 2 → Decremento
DMASRCINCR = 3 → Incremento

(R/W) *Bit* 7: DMADSTBYTE – Destino do DMA em Byte
(*DMA destination byte*)
Este bit seleciona se o destino das operações de DMA é um byte ou uma word.
DMADSTBYTE = 0 → Destino é uma word.
DMADSTBYTE = 1 → Destino é um byte.

(R/W) *Bit* 6: DMASRCBYTE – Origem do DMA em Byte
(*DMA source byte*)
Este bit seleciona se a origem das operações de DMA é um byte ou uma word.
DMASRCBYTE = 0 → Origem é uma word.
DMASRCBYTE = 1 → Origem é um byte.

(R/W) *Bit* 5: DMALEVEL – Nível de Disparo do canal de DMA
(*DMA level*)
Este bit seleciona se o disparo do DMA é por flanco ou por nível.
DMALEVEL = 0 → Disparo por flanco de subida (↑).
DMALEVEL = 1 → Disparo por nível alto.

(R/W) *Bit* 4: DMAEN – Habilitação do canal de DMA
(*DMA enable*)
Este bit controla a habilitação de um determinado canal de DMA. Para sua programação, o canal deve ser desabilitado.
DMAEN = 0 → Canal desabilitado.
DMAEN = 1 → Canal habilitado.

(R/W) *Bit* 3: DMAIFG – Flag de Interrupção do Canal de DMA
(*DMA interrupt flag*)
Esta flag vai a 1 quando termina as transferências que foram programadas no canal, ou seja, ele vai a 1 quando o contador de transferências chega a zero.
DMAIFG = 0 → Sem interrupção pendente.
DMAIFG = 1 → Tem interrupção pendente.

(R/W) *Bit* 2: DMAIE – Habilita a Interrupção do Canal de DMA
(*DMA interrupt enable*)
Quando este bit está em 1, ele permite que flag DMAIFG provoque a interrupção do canal de DMA (DMAIFG = 1).
DMAIE = 0 → Interrupção desabilitada.
DMAIE = 1 → Interrupção habilitada.

(R/W) *Bit* 1: DMAABORT – Aborto do DMA
(*DMA abort*)
Esta flag indica se uma operação de DMA foi interrompida por uma NMI.
DMAABORT = 0 → Transferência por DMA não foi interrompida.
DMAABORT = 0 → Transferência por DMA foi interrompida pela NMI.

(R/W) *Bit* 0: DMAREQ – Pedido de DMA
(*DMA request*)
Permite o disparo por software de uma transferência por DMA. Este bit é automaticamente zerado.
DMAREQ = 0 → Nenhum disparo.
DMAREQ = 1 → Dispara uma transferência por DMA.

12.8.7. DMAxSA – Registrador de Endereço de Origem para o Canal x *(DMA Channel x Source Address Register)*

Este registrador contém o endereço para indicar a origem das transferências por DMA. Durante a operação de DMA este registrador é copiado para um registrador temporário que é alterado durante as transferências. Os acessos a este registrador sempre retornam o valor programado. Deve ser notado que este é um registrador de 32 bits.

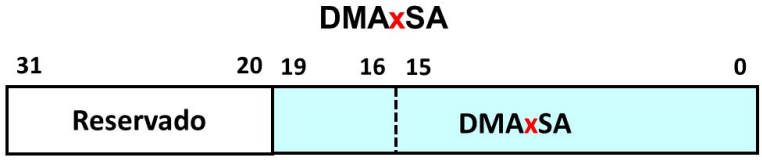


Figura 12.20. Registrador de endereço de origem do canal x de DMA.

(R) *Bits* 31, ..., 20: Reservados
Estes bits são reservados e sempre lidos como zero.

(R/W) *Bits* 19, ..., 0: DMAxSA – Endereço de Origem do Canal x de DMA (*DMA source address*)
Quando operando no modo Simples, estes bits contém o endereço o dado de origem (*source*) a ser transferido. Quando no modo Bloco ou Rajada, estes bits contém o endereço de início do bloco de dados a ser transferido. Este registrador permanece inalterado durante toda a operação de DMA, porém sua cópia temporária é quem sofre as alterações. Quando se escreve uma word neste registrador, os bits 16 até 19 são zerados. Para se acessar os 20 bits deste registrador é necessário o uso das instruções estendidas.

12.8.7. DMAxDA – Registrador de Endereço de Destino para o Canal x *(DMA Channel x Destination Address Register)*

Este registrador contém o endereço de destino das transferências por DMA. Durante a operação de DMA este registrador é copiado para um registrador temporário que é alterado durante as transferências. Os acessos a este registrador sempre retornam o valor programado. Deve ser notado que este é um registrador de 32 bits.

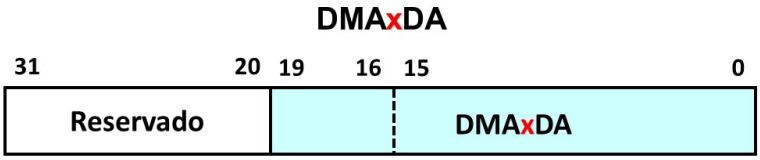


*Figura 12.21. Registrador de endereço de destino do canal **x** de DMA.*

(R) *Bits* 31, ..., 20: Reservados
Estes bits são reservados e sempre lidos como zero.

(R/W) *Bits* 19, ..., 0: DMAxDA – Endereço de Destino do Canal x de DMA (*DMA destination address*)
Quando operando no modo Simples, estes bits contém o endereço de destino do dado transferido. Quando no modo Bloco ou Rajada, estes bits contém o endereço de destino do bloco de dados a ser transferido. Este registrador permanece inalterado durante toda a operação de DMA, porém sua cópia temporária é quem sofre as alterações. Quando se escreve uma word neste registrador, os bits 16 até 19 são zerados. Para se acessar os 20 bits deste registrador é necessário o uso das instruções estendidas.

12.8.8. DMAxSZ – Registrador de Contador de Transferências do Canal x *(DMA Channel x Size Address Register)*

Este registrador contém a quantidade de transferências que ainda falta ser realizada.

Figura 12.22. Registrador com a quantidade de transferência a ainda falta acontecer.

(R/W) *Bits* 15, ..., 0: DMAxSZ – Quantidade de Transferências do Canal x de DMA (*DMA size*)
Este registrador de 16 bits é decrementado a cada transferência realizada (byte ou word). Quando ele chega a zero, é imediatamente recarregado com seu valor anterior. Como os registradores de endereço fonte (DMAxSA) e de endereço destino (DMAxDA) permanecem inalterados durante toda a operação de DMA, o programador deve usar este registrador DMAxSZ para saber quantas transferências ainda faltam acontecer e calcular os próximos endereços envolvidos.
DMAxSZ = 0 → transferência desabilitada;
DMAxSZ = 1 → ainda falta uma transferência (byte ou word);
...
DMAxSZ = 65.535 → ainda faltam 65.535 transferências. Este é o valor máximo.

12.8.9. DMAIV – Registrador de Vetores de Interrupção *(DMA Interrupt Vector Register)*

A arquitetura MDP430 prevê até 8 canais de DMA, cada um com sua flag de interrupção (DMAxIFG). Por isso, faz-se necessário um registrador de vetores de interrupção, para indicar, dentre as interrupções, pendentes, qual é a de maior prioridade. Lembramos que a versão F5529 possui apenas 3 canais de DMA.

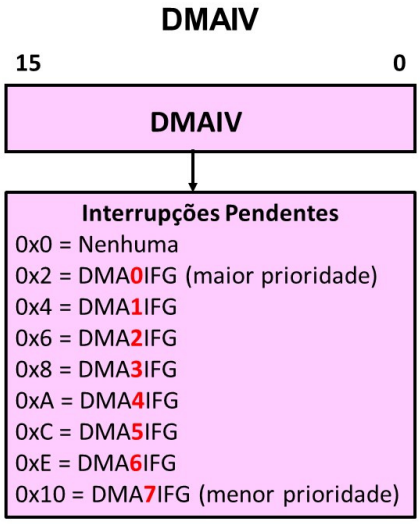


Figura 12.23. Registrador de vetores de interrupção do DMA.

(R/W) *Bits* 15, ..., 0: DMAIV – Registrador de Vetores de Interrupção de DMA (*DMA interrupt vector register*)
A leitura deste registrador retorna um número que corresponde à interrupção pendente de maior prioridade e, em seguida, apaga a flag (DMAxIFG) que solicitava esta interrupção. Sua operação segue a lógica dos demais registradores deste tipo.

12.9. Exercícios Resolvidos

ER 12.1. Este exercício tem o objetivo de ilustrar o conceito de DMA. O problema proposto é o de copiar os elementos do vetor **v0** no vetor **v1**. Ambos são vetores para 1.000 elementos inteiros (16 bits). O vetor **v0** deve ser inicializado com os valores {0, 1, ..., 999} e o vetor **v1** deve ser inicializado com zeros. Vamos ainda, de forma muito simples, usar o TA0R acionado pelo SMCLK (1.048.576) para medir quanto tempo demorou para se fazer cada cópia do vetor. Vamos provar quatro formas de efetuar a cópia:

1) Ensaio usando um laço de programa;
2) Ensaio usando DMA0 no Modo Simples;
3) Ensaio usando DMA0 no Modo Rajada e
4) Ensaio usando DMA0 no Modo Bloco.

Solução:

Vamos construir um programa que faça a cópia do vetor usando cada uma das possibilidades acima e medir o tempo gasto pela quantidade de ciclos do TA0 acionado pelo SMCLK.

Para o ensaio de cópia com um laço de programa, usamos o código abaixo:

```
    for (i=0; i<QTD; i++)           //Laço de cópia
        v1[i]=v0[i];                //Copiar
```

Para os ensaios de cópia com o DMA, será preciso configurar o controlador de DMA, cuja rotina está listada a seguir. A configuração de cada uma das opções é muito parecida. Basta alterar a opção DMADT do registrador de controle DMA0CTL.

- DMADT_0 → (0) Modo Simples;
- DMADT_1 → (1) Modo Bloco e
- DMADT_2 → (2) Modo Rajada

Note que a função de configuração já coloca o DMA para operar no Modo Simples, que tem o campo igual a zero. Assim, para mudar o modo baste sobrescrever este campo com DMADT_1 ou DMADT_2. Para o disparo do DMA0 foi selecionado o DREQ, que é ativado por software. Note que os acessos tanto à fonte quanto ao destino foram configurados para operar em 16 bits (DMADSTBYTE = 0 e DMASRCBYTE = 0). Como se precisa desses campos em zero, eles não aparecem na configuração do registrador DMA0CTL.

```
// Configurar DMA0 para o modo SIMPLES (DMADT=0)
void config_dma0(void){
    DMACTL0 = DMA0TSEL_0;   //DREQ
    DMA0CTL = DMADT_0|      //Modo Simples
              DMADSTINCR_3| //Incrementar endereço fonte
              DMASRCINCR_3| //Incrementar endereço destino
    DMA0SA = v0;            //Endereço fonte
    DMA0DA = v1;            //Endereço destino
    DMA0SZ = QTD;           //Quantidade
    DMA0CTL |= DMAEN;       //Habilitar
}
```

No teste que usa o Modo Simples, para cada transferência, é necessário fazer DREQ = 1, como listado abaixo. Precisaremos de uma quantidade de DREQ igual ao tamanho do vetor.

```
for (i=0; i<QTD; i++)           //Laço de cópia
        DMA0CTL |= DMAREQ;      //Disparar DMA, um disparo por cópia
```

Nos testes que usam o Modo Rajada ou o Modo Bloco, o primeiro DREQ = 1 dá início à transferência por DMA que só para quando o contador (DMA0SZ) chega a zero. Neste

caso, usamos a flag DMAIFG para saber quando a transferência terminou, como listado abaixo.

```
DMA0CTL |= DMAREQ;                    //Iniciar DMA
    while ( (DMA0CTL&DMAIFG) == 0); //Espesar DMAIFG
```

Na listagem completa, note que antes de iniciar cada teste, o TA0 é inicializado para o Modo Contínuo, usando SMCLK (1.048.576 Hz). Assim que o teste termina, o TA0 é paralisado e o valor de TA0R é guardado na variável correspondente.

O tempo que os testes demoram depende da frequência do MCLK, que usualmente é igual a 1.048.576 Hz. Pode ser que a CPU do leitor esteja configurada para um outro MCLK e por isso seu programa resulte em valores diferentes. A tabela abaixo apresenta os resultados encontrados. O termo Ciclos usa como referência o SMCLK = 1.048.576 Hz e o termo Ganho indica quantas vezes o ensaio foi mais rápido, tomando como base o ensaio 1.

Tabela 125. Resultados dos 4 ensaios propostos

Ensaio	Ciclos	Tempo (ms)	Ganho
1	14.006	13,36	1
2	11.006	10,50	1,3
3	2.510	2,39	5,6
4	2.010	1,92	7,0

Abaixo está a listagem completa do programa. Ele deve terminar com o led verde aceso. Neste momento o usuário deve usar o CCS para consultar as variáveis `tp1`, `tp2`, `tp3` e `tp4`, para ver o tempo gasto por cada teste. Se ao final, o led vermelho acender é porque aconteceu algum erro.

Listagem do ER 12.1

```
// ER 12.1
// Uso do DMA para fazer cópia de um vetor de inteiros
// Ensaio 1: Cópia usando laço de programa
// Ensaio 2: Cópia usando DMA Modo Simples
// Ensaio 3: Cópia usando DMA Modo Rajada
// Ensaio 4: Cópia usando DMA Modo Bloco

#include <msp430.h>

#define TRUE    1
#define FALSE   0
```

```
#define QTD   1000      //Tamanho dos vetores

void config_dma0(void); //Configurar DMA – Modo Simmples
char comp_vet(void);    //Comparar os vetores
void inic_vet(void);    //Inicializar vetores
void inic_leds(void);   //Inicializar leds

int v0[QTD],v1[QTD];    //Vetores de para o teste
int tp1,tp2,tp3,tp4;    //Registrar tempo consumido em cada teste

int main(void)
{
    int i;
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
    inic_leds();

    // ENSAIO 1 – Cópia com laço de programa
    inic_vet();
    TA0CTL=TASSEL_2 | MC_2 | TACLR; //Disparar TA0
    for (i=0; i<QTD; i++)           //Laço de cópia
        v1[i]=v0[i];                //Copiar
    TA0CTL=0;                       //Parar TA0
    tp1=TA0R;                       //Ler tempo consumido
    comp_vet();                     //Comparar vetores

    // ENSAIO 2 – Cópia com DMA em Modo Simples
    inic_vet();
    config_dma0();
    TA0CTL=TASSEL_2 | MC_2 | TACLR; //Disparar TA0
    for (i=0; i<QTD; i++)           //Laço de cópia
        DMA0CTL |= DMAREQ;          //Disparar DMA
    TA0CTL=0;                       //Parar TA0
    tp2=TA0R;                       //Ler tempo consumido
    if ( (DMA0CTL&DMAIFG) == 0)     //Completou a tarefa?
                while(1);           //Falha
    comp_vet();                     //Comparar vetores

    // ENSAIO 3 – Cópia dom DMA em Modo Rajada
    inic_vet();
    config_dma0();
    DMA0CTL |= DMADT_2;             //Modo Rajada
    TA0CTL=TASSEL_2 | MC_2 | TACLR; //Disparar TA0
    DMA0CTL |= DMAREQ;              //Iniciar DMA
    while ( (DMA0CTL&DMAIFG) == 0); //Espersar DMAIFG
    TA0CTL=0;                       //Parar TA0
    tp3=TA0R;                       //Ler tempo consumido
    comp_vet();                     //Comparar vetores

    // ENSAIO 4 – Cópia com DMA em Modo Bloco
    inic_vet();
```

```
    config_dma0();
    DMA0CTL |= DMADT_1;             //Modo Bloco
    TA0CTL=TASSEL_2 | MC_2 | TACLR; //Disparar TA0
    DMA0CTL |= DMAREQ;              //Iniciar DMA
    while ( (DMA0CTL&DMAIFG) == 0); //Espersar DMAIFG
    TA0CTL=0;                       //Parar TA0
    tp4=TA0R;                       //Ler tempo consumido
    comp_vet();                     //Comparar vetores

    P4OUT |= BIT7;                  //Led  verde, tudo certo
    while(TRUE);                    //Travar programa
    return 0;
}

// Configurar DMA0 para o modo SIMPLES (DMADT=0)
void config_dma0(void){
    DMACTL0 = DMA0TSEL_0;   //DREQ
    DMA0CTL = DMADT_0 | DMADSTINCR_3 | DMASRCINCR_3;  //Simples
    DMA0SA = v0;            //Endereço fonte
    DMA0DA = v1;            //Endereço destino
    DMA0SZ = QTD;           //Quantidade
    DMA0CTL |= DMAEN;       //Habilitar
}

// Comparar vetores, acender vermelho em caso de erro
char comp_vet(void){
    int i;
    for (i=0; i<QTD; i++){
        if (v0[i] != v1[i]){
            P1OUT |= BIT0;      //Acender vermelho
            return FALSE;
        }
    }
    return TRUE;
}

// Inicializar os vetores
void inic_vet(void){
    int i;
    for (i=0; i<QTD; i++){
        v0[i]=i;                //Inicializar v1 com 0, 1, 2, ..., 999
        v1[i]=0;                //Inicializar com zeros
    }
}

// Inicializar leds
void inic_leds(void){
    P1DIR |= BIT0;
    P1OUT &= ~BIT0;
    P4DIR |= BIT7;
```

```
    P4OUT &= ~BIT7;
}
```

ER 12.2. Este exercício e os próximos ilustram o DMA operando em auxílio à porta serial assíncrona (UART). Neste primeiro programa vamos usar as duas unidades assíncronas para se comunicarem, sendo que a USCI_A0 transmite e a USCI_A1 recebe. A USCI_A0 vai transmitir, elemento por elemento, o vetor **v0**, e a USCI_A1 vai receber esses elementos e armazená-los no vetor **v1**, ambos vetores de inteiros. Para tanto é preciso conectar P3.3 (Tx) com P4.1 (Rx) e P3.4 (Rx) com P4.2 (Tx). Nesta versão vamos usar um laço de programa para transmitir, sendo que a recepção será feita usando o DMA1.

Solução:
A figura abaixo ilustra o esquema a ser usado neste exercício. Um laço de programa, consultando a flag UCTXIFG vai escrevendo os elementos do vetor **v0** no TXBUF da USCI_A0. Já a USCI_A1, cada vez que recebe um elemento, ativa a flag UCRXIFG que dispara uma transferência pelo DMA1, que por sua vez transfere o conteúdo de UCA1RXBUF para o vetor **v1**. A transmissão é feita por bytes e o DMA usado para recepção também opera com bytes.

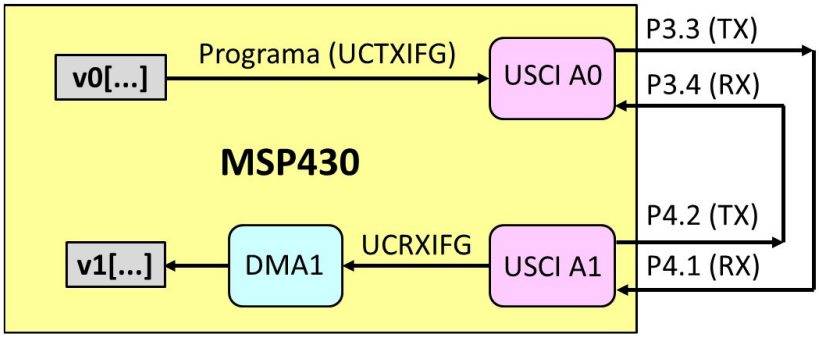


Figura 12.24. Esquema ilustrativo do exercício proposto.

A listagem abaixo apresenta o programa solução. As portas assíncronas foram configuradas para 9.600 bps, 8 bits de dados e 1 stop. Deve ser comentado que é preciso mapear os pinos P4.1 e P4.2, o que é feito na configuração da USCI_A1. Como prevenção, foi adicionada a função `testa_serial()` só para verificar se a conexão do leitor está correta. Caso seu programa fique preso nesta função, verifique a conexão dos pinos.

Note que a configuração do DMA1 o coloca no Modo Simples. O disparo é a flag UCA1IFG, a fonte é UCA1RXBUF e o destino é o vetor `v1`. O laço de transmissão, usando UCTXIFG, envia todos os elementos do vetor. Na recepção, a cada UCRXIFG ocorre uma

transferência por DMA. Ao terminar o laço, o programa testa o DMAIFG, só para ter certeza de que ele terminou.

Listagem do ER 12.2

```
// ER12p02.c
// Serial 9600, 8 bits, 1 stop
// UCA0 Transmite por software
// UCA1 recebe por DMA1

// USCI A0: P3.3=TX e P3.4=RX
// USCI A1: P4.2=TX e P4.1=RX

#include <msp430.h>

#define TRUE    1
#define FALSE   0
#define QTD     200  //Quantidade de transferências
#define UCA1RXBUF_ADR (__SFR_FARPTR) 0x060C

char testa_serial(void);
void config_dma1(void);
void config_serial_A0(void);
void config_serial_A1(void);
char comp_vet(void);
void inic_vet(void);
void inic_leds(void);

volatile int v0[QTD],v1[QTD];

int main(void)
{
    int i;
      WDTCTL = WDTPW | WDTHOLD;     // stop watchdog timer

    inic_leds();                //Inicializar Leds
    config_serial_A0();         //Inicializar USCI_A0
    config_serial_A1();         //Inicializar USCI_A1

    if (testa_serial()==FALSE)  //Conexão está
        while(TRUE);            //correta?

    inic_vet();                 //Inicializar vetores
    config_dma1();              //Inicializar DMA1

    for (i=0; i<QTD; i++){
        while( (UCA0IFG & UCTXIFG)==0); //Laço
        UCA0TXBUF=v0[i];                //de transmissão
```

```
    }
    while( (DMA1CTL&DMAIFG)==0);   //Esperar último DMA

    comp_vet();          //Comparar vetores
    P4OUT |= BIT7;       //Verde indica que terminou
    while(TRUE);       //Travar programa
    return 0;
}

// Testar serial
// Se prender nesta função, confira conexão
// P3.3-->P4.1 e P4.2-->P3.4
char testa_serial(void){
    char i;
    for (i=0; i<QTD; i++){
        while( (UCA0IFG & UCTXIFG)==0);
        UCA0TXBUF=i;
        while( (UCA1IFG & UCRXIFG)==0);
        if (UCA1RXBUF != i){
            P1OUT |= BIT0;  //Vermelho
            return FALSE;
        }
    }
    return TRUE;
}

// Configurar DMA1
// UCA1RXBUF (DMA1)-->v1
void config_dma1(void){
    DMACTL0 |= DMA1TSEL_20;   // Disparo = UCA1RXIFG
    DMA1CTL = DMADT_0      |  //Modo Simples
              DMADSTINCR_3 |  //Ponteiro destino, incrementar
              DMASRCINCR_0 |  //Ponteiro fonte fixo
              DMADSTBYTE;     //Destino opera com bytes
              DMASRCBYTE;     //Fonte opera com bytes
    DMA1SA = UCA1RXBUF_ADR;   //Fonte = UCA1RXBUF
    DMA1DA = v1;              //Destino = vetor v1
    DMA1SZ = QTD;             //Quantidade de transferências
    DMA1CTL |= DMAEN;         //Habilitar DMA1
}

// Configurar USCI A0
// P3.3=TX e P3.4=RX
void config_serial_A0(void){
    UCA0CTL1 = UCSSEL_2 | UCSWRST;
    UCA0IFG = 0;  //Apagar TXIFG e RXIFG
    UCA0BRW = 6;
    UCA0MCTL = UCBRF_13 | UCBRS_0 | UCOS16;
    UCA0CTL0 = 0;
    P3SEL |= BIT3; //Tx serial
```

```
    P3SEL |= BIT4; //Rx serial
    UCA0CTL1 &= ~ UCSWRST; //UCSI sai de Reset
}

// Configurar USCI A1
// P4.1=RX e P4.2=TX
void config_serial_A1(void){
    UCA1CTL1 = UCSSEL_2 | UCSWRST;
    UCA1IFG = 0;    //Apagar TXIFG e RXIFG
    UCA1BRW = 6;
    UCA1MCTL = UCBRF_13 | UCBRS_0 | UCOS16;
    UCA1CTL0 = 0;
    P4SEL |= BIT2; //Tx serial
    P4SEL |= BIT1; //Rx serial
    PMAPKEYID = 0X02D52; //Liberar mapeamentp
    P4MAP1 = PM_UCA1RXD;   //P4.1 = RX
    P4MAP2 = PM_UCA1TXD;   //P4.2 = TX
    UCA1CTL1 &= ~ UCSWRST; //UCSI sai de Reset
}

// Inicializar os vetores
void inic_vet(void){
    int i;
    for (i=0; i<QTD; i++){
        v0[i]=i;     //Inicializar v0 = 0, 1, 2, ...
        v1[i]=0;   //Inicializar v1 = 0
    }
}

// Comparar vetores, acender vermelho em caso de erro
char comp_vet(void){
    int i;
    for (i=0; i<QTD; i++){
        if (v0[i] != v1[i]){
            P1OUT |= BIT0;      //Acender vermelho
            return FALSE;
        }
    }
    return TRUE;
}

// Inicializar leds
void inic_leds(void){
    P1DIR |= BIT0;
    P1OUT &= ~BIT0;
    P4DIR |= BIT7;
    P4OUT &= ~BIT7;
}
```

ER 12.3. Vamos repetir o exercício anterior, mas agora com a USCI_A0 transmitindo pelo canal 0 de DMA e a USCI_A1 recebendo por um laço de programa.

Solução:
A figura abaixo ilustra o esquema a ser usado neste exercício. A recepção será com um laço de programa, consultando a flag UCRXIFG que vai escrevendo os bytes recebidos no vetor `v1`. A transmissão deve usar o DMA0 disparado pelo UCTXIFG. O detalhe é que o disparo do DMA acontece quando esta flag vai de 0 para 1. Para conseguir isso, precisamos escrever o primeiro byte no UCA0TXBUF. Assim, a quantidade de transferências é subtraída de um e o endereço a ser programado na DMA0 é o do `v0[1]`. O resto do programa funciona da forma esperada.

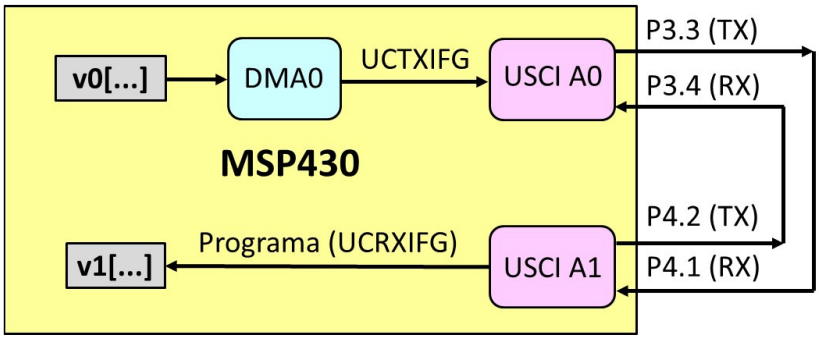


Figura 12.25. Esquema ilustrativo do exercício proposto.

A listagem abaixo apresenta o programa solução. Note que para evitar que a listagem ficasse grande, as funções que eram comuns ao programa anterior foram omitidas.

Listagem do ER 12.3

```
// ER12p03.c
// UCA0 Transmite por DMA0
// UCA1 recebe por software

// USCI A0: P3.3=TX e P3.4=RX
// USCI A1: P4.2=TX e P4.1=RX

#include <msp430.h>

#define TRUE    1
#define FALSE   0
#define UCA0TXBUF_ADR (__SFR_FARPTR) 0x05CE
#define QTD 200 //Quantidade de transferências
```

```
char testa_serial(void);
void config_dma0(void);
void config_serial_A0(void);
void config_serial_A1(void);
char comp_vet(void);
void inic_vet(void);
void inic_leds(void);

volatile char v0[QTD],v1[QTD];

int main(void)
{
    int i;
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
    inic_leds();                //Inicializar Leds
    config_serial_A0();         //Inicializar USCI_A0
    config_serial_A1();         //Inicializar USCI_A1

    if (testa_serial()==FALSE)  //Conexão está
        while(TRUE);            //correta?

    inic_vet();                 //Inicializar vetores
    config_dma0();
    UCA0TXBUF=v0[0];            //Escrita para disparar DMA0
    for (i=0; i<QTD; i++){
        while( (UCA1IFG & UCRXIFG)==0);   //Laço de
        v1[i]=UCA1RXBUF;                  //recepção
    }
    comp_vet();         //Comparar vetores
    P4OUT |= BIT7;      //Verde indica que terminou
    while(TRUE);        //Travar programa
    return 0;
}

// Configurar DMA0
// v0 (DMA0)--> UCA0TXBUF
void config_dma0(void){
    DMACTL0 = DMA0TSEL_17;    //Disparo = UCA0TXIFG
    DMA0CTL = DMADT_0      |  //Modo Simples
              DMADSTINCR_0 |  //Ponteiro destino, fixo
              DMASRCINCR_3 |  //Ponteiro fonte, incrementar
              DMADSTBYTE;     //Destino = byte
              DMASRCBYTE;     //Fonte = byte
    DMA0SA = &v0[1];          //Segundo elemento do vetor
    DMA0DA = UCA0TXBUF_ADR;   //Endereço de UCA0TXBUF
    DMA0SZ = QTD-1;           //Transferir um a menos
    DMA0CTL |= DMAEN;         //Habilitar
}

// Copiar as funções abaixo do ER 12.2
```

```
char testa_serial(void){...}
void config_serial_A0(void) {...}
void config_serial_A1(void) {...}
void inic_vet(void){...}
char comp_vet(void) {...}
void inic_leds(void){...}
```

ER 12.4. Neste exercício vamos usar DMA tanto para transmitir, como para receber. É pedido para repetir o exercício anterior, mas com a USCI_A0 transmitindo pelo canal 0 de DMA0 e a USCI_A1 recebendo pelo canal 1 de DMA1.

Solução:
A figura abaixo ilustra o esquema a ser usado neste exercício. Tanto a transmissão como a recepção usarão canais de DMA. É a união das operações de DMA dos dois exercícios anteriores. Não é mais necessário um laço de programa. Uma vez iniciado a transmissão ela ocorre toda por DMA e a única obrigação do programa é verificar o fim das transferências usando DMAIFG do DMA1.

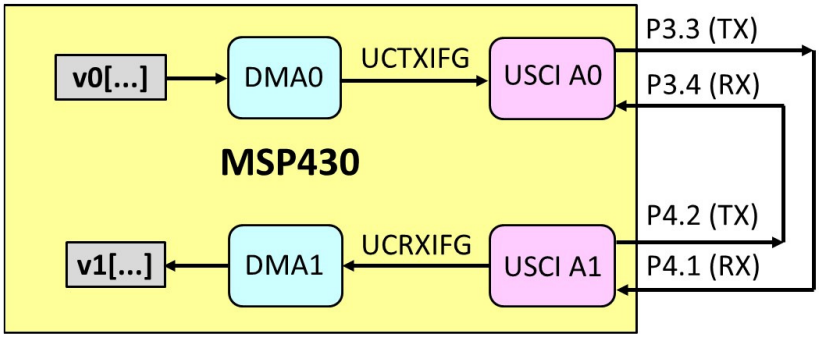


Figura 12.26. Esquema ilustrativo do exercício proposto.

A listagem abaixo apresenta o programa solução. Note que para evitar que a listagem ficasse grande, as funções que eram comuns ao programa anterior foram omitidas.

```
// ER12p04.c
// UCA0 Transmite por DMA0
// UCA1 recebe por DMA1

// USCI A0: P3.3=TX e P3.4=RX
// USCI A1: P4.2=TX e P4.1=RX

#include <msp430.h>

#define TRUE    1
#define FALSE   0
#define QTD 10 //Quantidade de transferências
```

```
//#define UCA0RXBUF_ADR (__SFR_FARPTR) 0x05CC
#define UCA0TXBUF_ADR (__SFR_FARPTR) 0x05CE
#define UCA1RXBUF_ADR (__SFR_FARPTR) 0x060C
//#define UCA1TXBUF_ADR (__SFR_FARPTR) 0x060E

char testa_serial(void);
void config_dma0(void);
void config_dma1(void);
void config_serial_A0(void);
void config_serial_A1(void);
char comp_vet(void);
void inic_vet(void);
void inic_leds(void);

volatile char v0[QTD],v1[QTD];

int main(void)
{
    char i;
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
    config_serial_A0();
    config_serial_A1();

    if (testa_serial()==FALSE)
        while(TRUE);

    inic_vet();                 //Inicializar vetores
    config_dma0();
    config_dma1();
    UCA0TXBUF=v0[0];
    while( (DMA1CTL&DMAIFG)==0);
    comp_vet();         //Comparar vetores
    P4OUT |= BIT7;      //Verde indica que terminou
    while(TRUE);        //Travar programa
    return 0;
}

// Configurar DMA0
// v_tx (DMA0)--> UCA0TXBUF
void config_dma0(void){
    DMACTL0 = DMA0TSEL_17;    //Disparo = UCA0TXIFG
    DMA0CTL = DMADT_0      |  //Modo Simples
              DMADSTINCR_0 |  //Ponteiro destino, fixo
              DMASRCINCR_3 |  //Ponteiro fonte, incrementar
              DMADSTBYTE   |  //Destino opera com bytes
              DMASRCBYTE;     //Fonte opera com bytes
    DMA0SA = &v0[1];          //Segundo elemento do vetor
    DMA0DA = UCA0TXBUF_ADR;   //Endereço de UCA0TXBUF
    DMA0SZ = QTD-1;           //Transferir um a menos
```

```
    DMA0CTL |= DMAEN;          //Habilitar
}

// Configurar DMA1
// UCA1RXBUF (DMA1)-->v1
void config_dma1(void){
    DMACTL0 |= DMA1TSEL_20;   // Disparo = UCA1RXIFG
    DMA1CTL = DMADT_0       |  //Modo Simples
              DMADSTINCR_3 |  //Ponteiro destino, incrementar
              DMASRCINCR_0 |  //Ponteiro fonte fixo
              DMADSTBYTE   |  //Destino opera com bytes
              DMASRCBYTE;     //Fonte opera com bytes
    DMA1SA = UCA1RXBUF_ADR;   //Fonte = UCA1RXBUF
    DMA1DA = v1;              //Destino = vetor v1
    DMA1SZ = QTD;             //Quantidade de transferências
    DMA1CTL |= DMAEN;         //Habilitar DMA1
}

// Copiar as funções abaixo do ER 12.2
char testa_serial(void){...}
void config_serial_A0(void) {...}
void config_serial_A1(void) {...}
void inic_vet(void){...}
char comp_vet(void) {...}
void inic_leds(void){...}
```

12.10. Exercícios Propostos

EP 12.1. No ER 9.6 enviamos um vetor de 10 bytes pela porta I2C. Repita este exercício, mas agora transmitindo esses 10 dados por DMA.

EP 12.2. Este exercício é um pouco difícil. O trabalho com o LCD com a interface PCF8574 é grande devido à necessidade de se usar I2C o tempo todo. Uma forma de se simplificar isso é criar uma cópia (um buffer) do LCD em memória. Por exemplo, para um LCD 2x16, poderíamos criar uma matriz com duas linhas e 16 colunas. Assim, os programas atualizam esse buffer, que periodicamente é transferido para o LCD. Pense em um buffer um pouco mais inteligente que pudesse ser enviado periodicamente para LCD por DMA.

EP12.3. Vamos usar o ADC para realizar 100 conversões da entrada A0 e transferir por DMA para o vetor `int vet[100]`. O ADC deve operar no modo Simples com Repetição.

EP12.4. Vamos usar o ADC para realizar 100 conversões da entrada A0 e transferir por DMA para o vetor `int vet[100]`. O ADC deve operar no modo Sequência de Canais com Repetição.

EP12.5. O ER 7.18 propunha uma forma de se gerar um sinal senoidal a partir de um conversor DAC construído com PWM. Aperfeiçoe este exercício com o emprego de DMA. Note que agora não se demanda mais por CPU.

Colocar num apêndice
Gabarito para configurar os registradores do DMA

	15 – 13	12 – 8	7 – 5	4 – 0
DMACTL0	-	DMA1TSEL	-	DMA0TSEL
	-		-	
DMACTL1	-	DMA3TSEL	-	DMA2TSEL
	-		-	
DMACTL2	-	DMA5TSEL	-	DMA4TSEL
	-		-	
DMACTL3	-	DMA7TSEL	-	DMA6TSEL
	-		-	

	15 – 3	2	1	0
DMACTL4	-	DMARMWDIS	ROUNDROBIN	ENNMI
	-			

	15	14	13	12	11	10	9	8
DMAx CTL	-	DMADT			DMADSTINCR		DMASRCINCR	
	7	6	5	4	3	2	1	0
	DMA DSTBYTE	DMA SRCBYTE	DMA LEVEL	DMA EN	DMA IFG	DMA IE	DMA ABORT	DMA REQ

	31 – 17	16 – 19	15 – 0
DMAxSA	-		DMAxSA
	-		
DMAxDA	-		DMAxDA
	-		
DMAxSZ	-	-	DMAxSZ
	-	-	